

POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione



POLO REGIONALE DI COMO

Corso di Laurea Specialistica in
Ingegneria Informatica

METHODOLOGY AND PATTERNS FOR DEVELOPING COMMUNITY-BASED WEB APPLICATIONS WITH A MODEL-DRIVEN APPROACH

(METODOLOGIA E PATTERN PER LO SVILUPPO DI APPLICAZIONI WEB FONDATE SU
COMUNITÀ CON UN APPROCCIO ORIENTATO AI MODELLI)

Relatore: prof. Piero Fraternali

Tesi di Laurea di:

Lorenzo Frattini
matricola: 681085

Matteo Silva
matricola: 675751

Anno Accademico 2006/2007

A mio padre, che per tutta la
vita avrebbe desiderato essere
qui, in questo momento.

A mia madre, che mi ha dato la
fiducia e il supporto senza i
quali non avrei mai potuto
raggiungere questo traguardo.

Lorenzo

Alla mia famiglia, al mio amore
e a chi mi ha fatto *rinascere*.

Matteo

Summary

With this document we show in detail our work of thesis, aimed to bringing model-driven development techniques to the issue of designing community-based web applications, with special regard to the WebML development process.

In order to achieve this, we first benchmarked researches and results coming from the scientific community, according to many different disciplines. Therefore we analyzed most popular social networks and community-based web applications in order to identify high-level patterns to be reused in different contexts, regardless specific purposes and application domains. Starting from this, we reviewed the entire WebML process, with the addition of a specific phase for the community design and giving precise indications on how to cope with it.

Finally we provide a case study, borrowed from the professional context we worked in during our stage at Web Models s.r.l., in which we exemplify a community-based application design according to our results.

Sommario

Con questo documento illustriamo il nostro lavoro di tesi, volto a portare la tecnica di sviluppo model-driven alla progettazione di applicazioni web basate su comunità di utenti, con preciso riferimento al processo di sviluppo WebML.

A tal fine abbiamo dapprima effettuato un benchmarking orientato a raccogliere ricerche e risultati provenienti dalla comunità scientifica. Quindi abbiamo esaminato le più famose applicazioni web di tipo social-network fondate su comunità di utenti, con lo scopo di identificare pattern di alto livello che potessero essere riutilizzati in diversi contesti, a prescindere da scopi e domini applicativi specifici. Partendo da questo abbiamo rivisto l'intero processo di sviluppo WebML, introducendo una fase specifica di progettazione della comunità, per la quale forniamo indicazioni operative.

Infine illustriamo un caso di studio, tratto dal contesto professionale all'interno del quale ci siamo trovati ad operare durante il nostro stage presso Web Models s.r.l., attraverso il quale esemplifichiamo il progetto di un'applicazione web community-based compatibilmente con i risultati ottenuti.

Table of contents

Definitions.....	7
Introduction.....	8
Web 2.0.....	8
Virtual communities.....	10
Virtual communities and Web 2.0.....	10
Critical issues in virtual communities: a top-down approach.....	11
The business perspective.....	12
The communication perspective.....	13
The engineering perspective.....	15
State of the art.....	16
Scientific fields of research.....	16
Conferences.....	17
Frameworks.....	18
Algorithms	20
Tools and software libraries.....	21
Conclusions.....	22
Patterns.....	24
Basic concepts.....	25
Member.....	25
Item.....	27
Relationship.....	29
Group.....	30
Activity.....	32
Message.....	35

The community model.....	36
Motivation.....	36
Pay-Perform-Reward chain.....	36
Pay-Perform-Reward runtime.....	39
Front-end patterns.....	42
Navigation patterns.....	42
Contribution patterns.....	46
Social patterns.....	53
Back-end patterns.....	62
Front-end patterns in social networks.....	69
Process.....	70
Model-Driven Development.....	71
Model-Driven Architecture.....	71
Model-driven development process.....	72
WebML process.....	75
WebML.....	76
WebRatio.....	77
Extending the WebML process.....	80
Motivations.....	80
Community development lifecycle.....	81
Inputs and outputs of the process.....	82
Development roles.....	83
Communication expert.....	83
Social network expert.....	84
Requirements specification.....	84
Identification of users.....	85
Functional requirements.....	88
Personalization requirements.....	90
Non-functional requirements.....	91
Community design.....	91
Community state variable design.....	92
Community state variable taxonomy.....	95
Mapping state variable transitions using active rules.....	96
Example 1.....	98
Example 2.....	103

Methodology and patterns for developing community-based web applications with a model-driven approach

Implementation.....	106
Data model extension.....	107
Back-end patterns.....	112
Overview.....	112
Evaluation.....	113
Notification.....	114
Payment.....	115
Permission check.....	115
Relevance adjustment.....	116
Reputation adjustment.....	117
Reward.....	118
Syndication.....	118
Case study.....	120
Context.....	121
Requirement analysis.....	122
Mle in a nutshell.....	122
Community requirements elicitation.....	123
User hierarchy extension.....	123
Design community services by patterns.....	125
Community design.....	127
The moderating-fostering strategy.....	127
Community state variables.....	128
Mapping state variables transitions.....	133
Define the update functions.....	135
Define the ban process.....	137
Conclusions.....	138
Results.....	139
Further works.....	140
Ringraziamenti.....	142
Lorenzo.....	142
Matteo.....	142
Bibliography.....	144

Definitions

In this chapter we introduce some basic definitions before proceeding to a presentation of our research on virtual communities, with explicit reference to the so-called Web 2.0. We focused our study on understanding the current state of the art of the field, in order to outline the main scientific areas involved and research groups currently working on this topic.

Introduction

When we discuss about virtual communities we can look at them from different points of view; we could focus our attention on psychological aspects, or even on technological problems, or furthermore we could only want to understand how to exploit virtual communities for improving sells of an e-commerce firm. It is trivial to assert that virtual communities are complex objects, that need to be analyzed by an cross-disciplinary approach. The main question is: which are the open issues related to virtual communities and who is in charge to solve them?

For trying to answer this question, we start from definitions of virtual communities, then we outline which scientific disciplines are studying them, in order to understand which actors are involved and which challenges are still open.

Web 2.0

The term “Web 2.0” was forged during a brainstorming session between O'Reilly and Media Live International, trying to identify the common traits of the web industry that survived the so-called “bursting of the dot-com bubble”, in the fall of 2001¹. They outlined eight principles that summarize the Web 2.0 trend:

1. “*The web as a platform*”: the web has not to be intended any longer as a simple repository of informations, but as a platform for more complex services.
2. “*Harness collective intelligence*”: the network effects from user contributions are one of the most important resources to be exploited.
3. “*Blogging and the wisdom of the crowds*”: people should be intended not just as an audience, but rather as main actor that actively decides what is important and what is not.
4. “*Data as the next Intel Inside*”: data owning, privacy, intellectual property and

content licensing are very relevant issues for web sites, and they can be turned into opportunities for companies.

5. “*End of software release cycle*”: the software development lifecycle needs to be revised; operations should become core competencies and user must become co-developers, emphasizing the beta-stage of development as the daily evolution of applications.
6. “*Lightweight programming models*”: applications should be loosely coupled, interoperable. Syndications of contents should be preferred to coordination in order to allow “remixability” of contents and services.
7. “*Software above the level of single device*”: software has not to be designed for the personal computer usage only; more and more devices are being connected to the web platform and the desktop paradigm is progressively turning into ubiquitous computing.
8. “*Rich user experience*”: usability and human-computer interaction are very relevant factors for succeeding in the Web 2.0 age. Users prefer easy-to-use, essential and effective interfaces.

According to McKinseyⁱⁱ, this trend can be classified into nine different categories, that we hereby grouped into *tools* (as instruments and technologies) and *paradigms* (that move design focus from users to community).

Tools	Paradigms
RSS Podcasts Wikis Mash-ups Web services	Peer-to-peer networking Social networking Collective intelligence

From the previous classifications we can outline four common factors:

- meta-description of contents;
- decentralization of production and services;
- active role of users;
- perception of democracy.

Virtual communities

The term *virtual community* was first introduced in 1993 by Howard Rheingoldⁱⁱⁱ. Current literature proposes many definitions and, for the intrinsic interdisciplinary nature of the subject, every discipline underlines a particular aspect. We report the following, by Tardini & Cantoni^{iv}:

«A virtual community is a group of people to whom interactions and communications play an important role in creating and maintaining significant social relations».

The definition refers to the concept of *social relation* that we will use to define the concept of *social network* as the entire set of social relations. Thinking a virtual community as a social network will be useful for the comprehension of the following paragraphs, because many efforts of researches go in this direction.

Before continuing, it is useful to remark the main aspects of a virtual community, according to Preece^v:

- *People*: the actors of social interactions, they perform tasks and plays role, such as, for instance, leaders or moderators.
- *A shared purpose*: the focus of social interactions could be a special matter of interest, a job or a service.
- *Policies*: rituals, rites, protocols and laws.
- *Computer systems*: software and hardware infrastructures needed to let people communicate, share tasks and feel together what is also called “social software”.

Virtual communities and Web 2.0

Before going on, we'd like to underline that the concept of virtual community is not subsequent to that of Web 2.0. The Web 2.0 definition, as we mentioned before, was forged in 2001; until then the phenomena of virtual communities was already affirmed (the first case of virtual community go back to 1979 with the USENET case^{vi}). Furthermore, it is not a mistake to state that virtual communities are the engine that boosts the first generation web towards the second.

Critical issues in virtual communities: a top-down approach

Nowadays, the success of many web initiatives is closely related to their capability to grow a community around themselves; this entails a number of difficulties in terms of design, development and maintenance of a web application.

The point is that the production of the web application is only the result of a work that includes many critical issues that need to be opportunely taken into consideration before starting designing the application. We tried to study such issues from different perspectives, in order to identify their level of abstraction and the mutual relationships that link them together.

In order to achieve this, we propose a framework according to which it is possible to study virtual communities from three different points of view:

- *The business perspective.* This “perspective” includes all those critical issues which are related to the capability of an organization to take advantage from the virtual community in its core business (we extended the term “business” also for non-lucrative initiatives such as foundations).
- *The communication perspective.* This “perspective” covers all the communication theory-related requirements that need to be achieved in order to reach business goals. This includes all the literature related to virtual communities, common ground and other studies about collaborative activities.
- *The engineering perspective.* This “perspective” includes all those frameworks that have been proposed for monitoring, maintaining and supporting virtual

communities. At this level we also include all the technological support for addressing marketing strategies and tracking of customers' behavior.

The business perspective

In order to better identify critical issues at this level, we distinguish between web initiatives not-for-profit and for-profit. In both cases for addressing business goals it is necessary to study virtual communities in order to understand group dynamics, interactions and mutual influences among participants.

Not-for-profit initiatives

In the case of not-for-profit initiatives, the virtual community acts as main source for contents; community is the unique selling point of the initiative and the main goals of organizations should be as follows:

- Encourage people to publish contents
- Tailor a thick social network, promoting social collaboration
- Monitor the social network
- Monitor the quality of contents

For-profit initiatives

When dealing with web initiatives for profit instead, a virtual community plays two different roles: the role of *consumers* and the role of *promoter* for goods and/or services.

The first assume that the community is considered as a “pool” of consumers or simply as a portion of the organization's market; by sampling the behavior of community members' organizations are facilitated in improving their marketing strategies, proposing products and services by one-to-one marketing and forecasting sales.

The promoter role has to be intended as the potential for the community to promote a particular product or a service. Promotions can be both *internal* and *external* depending on the extent to which members are capable of persuading people inside or outside the community to buy or use a particular product or service.

Summarizing, we can state that critical aspects related to web initiatives for-profit are:

- Encourage people to promote

- Monitor participants' reputation
- Monitor business transactions
- Monitor community rumors

The communication perspective

Literature of communication theory has led to a better comprehension of groups dynamics within virtual communities. Business requirements are closely tied to the organization's capability of preventing community from collapse, but it's still necessary to outline causes that can lead a community towards the failure.

Studies conducted by Preece^{vii} affirm that the success of a virtual community depends on two main factors:

1. *Sociability*: the capability of people involved to interact with others.
2. *Usability*: the ease of using tools for human-computer-interaction.

There are many contributions about sociability, such as *common ground*^{viii} and *social presence*^{ix} theories; we hereby summarize some relevant aspects.

Sociability component	Description	Critical aspects	Negative impact on the community
Purpose	“What draws people to a community”	→ Non-clearness of the purpose	→ Less attraction → Hostility → Frustration → Loss of trust → Loss of commitment
People	“The mind and the heart of the community”	→ Lurkers → Killers → Participants turnover → Critical mass	→ Less participation → Less discussion improvements → Less discussion continuity → Less attraction
Policies	“The rules that dominate community behavior”	→ Join requirements → Communication style → Moderation rules → Privacy rules → Copyright	→ Personal responsibility → Non-conformance → Lack of democracy → Fear to speak

Methodology and patterns for developing community-based web applications with a model-driven approach

Sociability component	Description	Critical aspects	Negative impact on the community
			→ Violations

While sociability is mainly related to psychological aspects, usability involves both psychological theories and technological issues. We restrict our acceptance of usability to the context of web applications, borrowing some concepts from the MiLE+^x framework proposed by TEC-Lab¹ researchers and from Jakob Nielsen's heuristics^{xi}.

Usability component	Description	Critical aspects	Negative impact on the community
Content	Information published within the community platform	<ul style="list-style-type: none">→ Accuracy→ Currency→ Coverage→ Objectivity→ Authority→ Clarity→ Language errors	<ul style="list-style-type: none">→ Loss of trust→ Less participation
Navigation	Patterns used to dispose contents	<ul style="list-style-type: none">→ Segmentation→ Orientation→ Accessibility	<ul style="list-style-type: none">→ Frustration→ Less participation
Semiotics	Signs and symbols used	<ul style="list-style-type: none">→ Labels ambiguity→ Labels overlapping	<ul style="list-style-type: none">→ Frustration→ Less participation
Graphics	Graphic design and layout	<ul style="list-style-type: none">→ Content positioning→ Colors and font rules	<ul style="list-style-type: none">→ Frustration→ Less participation
Cognitive aspects	Simplicity to understand and memorize information	<ul style="list-style-type: none">→ Information overload→ Quick reading→ Orientation	<ul style="list-style-type: none">→ Loss of commitment→ Less participation
Technology	Set of software and hardware used to serve the community	<ul style="list-style-type: none">→ Correctness→ Availability→ Interoperability→ Performance	<ul style="list-style-type: none">→ Loss of trust→ Less participation→ Frustration

1 Technology Enhanced Communication Laboratory – <http://www.tec-lab.ch/>

The engineering perspective

We found lot of researches about social network modeling and analysis, in particular for addressing the following issues:

- Recommendation systems
- Participation fostering
- Path recognition

The level of maturity of this branch of research is good; we found loads of algorithms and tools specifically designed to support social network analysis. A full list of them is available at the INSA² website, with many links to interesting publications. The main bottleneck of such techniques is the scalability due to the complexity of algorithms used to analyze social behavior.

Another branch of research is focused on ontologies and semantic representation of knowledge. In this direction some researches are focused on the development of intelligent agents capable to use semantic definitions for automatically compose complex services.

The latest relevant research is the Web3D project³, which is aimed to propose a framework for web applications into 3D virtual environments.

² International Network for Social Network Analysis – <http://www.isna.org/>

³ <http://www.web3d.org/>

State of the art

In the following paragraphs we present a synthetic report about the most up-to-date results related to the specific matter of virtual communities. We focused our analysis on the relationships between topics of interest and scientific fields of research; according to this and to the framework we proposed in the previous chapter, we report the most relevant affirmed guidelines, frameworks and algorithms proposed by the scientific community. Furthermore, we propose a map of the most authoritative contributions to the matter and their related conferences.

Scientific fields of research

It was not easy to make a complete, easy-to-access report about the specific areas of interest covered by this topic; we hereby present the methodology adopted for addressing such research.

First, we defined a coherent subset of scientific fields of research matching the following prerequisites:

- *Authority*: we referred to fields of research recognized by the entire scientific community, so that the classification of every discipline is defined by a widely shared description.
- *Internationality*: due to the fact we conducted a research among international groups of research and contributions, we needed a classification of the scientific fields of interest independent from the educational system of each country.

In order to satisfy these constraints we referred to the classification method proposed by ACM⁴; such classification is widely adopted to classify many scientific publications and has a formal description for each discipline involved in the computer science field.

4 The 1998 ACM Computing Classification System – <http://www.acm.org/about/class/ccs98-html>

Then, starting from all the publications gathered around the topic of virtual communities, we tried to outline the main topics and the scientific fields of research involved.

Conferences

We hereby outline the International conference from whose proceedings we gathered publications and contributions around the topic of virtual communities.

Organization / Institution	Conference name	Conference description
ACM	ACM-HT	Conference on Hypertext and Hypermedia
ACM	ACM-WikiSym	International Symposium on Wikis
ACM	ACM-WI	Web Intelligence
ACM	ACM-SIGMIS	Computer Personnel Doctoral Consortium and Research Conference
Czech Society for Operational Research	EURO	European Conference on Operational Research
CEUR	CEUR	International Workshop on Description Logics
ESCW	ESCW	European Semantic Web Conference
FIP	ICSOC	International Conference on Service Oriented Architectures
IASTED	PDCN	Parallel and Distributed Computing and Networks
IASTED	EuroIMSA	Internet and Multimedia Systems and Applications
IEEE	CEC	IEEE Conference on E-Commerce Technology
IEEE	EEE	Conference on Enterprise Computing, E-Commerce and E-Services
IEEE	AINA	Advanced Information Networking and Applications
IEEE	RCIS	Conference on Research Challengers in Information Science
IJCAR	IJCAR	International Joint Conference on Automated Reasoning

Methodology and patterns for developing community-based web applications with a model-driven approach

KES	KES-AMSTA	KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications
Knowledge System Institute	SEKE	International Conference on Software Engineering and Knowledge Engineering
Microsoft Latin and Caribbean Collaborative ICT Research	CRIWG	International Workshop on Groupware
School of Computer Science and Engineering at the University of New South Wales	APWeb/WAIM	International Conference on Web-Age Information Management
Shidler College of Business University of Hawaii at Manoa	HICSS	Hawaii International Conference on System Sciences
System and Information Sciences Notes	ICABS	International Conference on Adaptive Business Systems
The Chinese University of Hong Kong The University of Arizona	PAISI	Pacific Asian Workshop on Intelligence and Security Informatics
The HCI International Conference	HCI	International Conference on Human-Computer Interaction
User Modeling	UM	User Modeling
CMP	WEB2CON	Web 2.0 summit
Web3D	Web3D	Web3D technologies for the World Wide Web

Frameworks

The following set of resources proposes framework, patterns and models for designing applications, focusing on different issues.

Methodology and patterns for developing community-based web applications with a model-driven approach

Name	Focus	Fields of research
Kolloch's framework ^{xii}	Participation fostering	Social and behavioral science
Bishop's framework ^{xiii}	Participation fostering	HCI, Social and behavioral science
Visualizing personal relations in online communities ^{xiv}	Participation fostering	Computer-supported collaborative work, HCI, Social and behavioral science
User and community-adaptive reward mechanism for sustainable online community ^{xv}	Participation fostering	Computer-supported collaborative work, Social and behavioral science
A system dynamics approach to study virtual communities ^{xvi}	Social Network Analysis	Computer-supported collaborative work, Social and behavioral science
The Design of Gugubarra 2.0: a tool for building and managing profiles of web users ^{xvii}	Participation fostering	Computer-supported collaborative work, Social and behavioral science
Knowledge contribution in problem solving virtual communities: the mediating role of individual motivations ^{xviii}	Participation fostering	Social and behavioral science
The Effect of Individual Needs, Trust and Identification in Explaining Participation Intentions in Virtual Communities ^{xix}	Participation fostering	Social and behavioral science
Applying Trust, Reputation and Intuition Aspects to Support Virtual Communities of Practice ^{xx}	Participation fostering	Social and behavioral science, Probability and statistics
A Social Hypertext Model For Finding Community In Blogs ^{xxi}	Social Network Analysis	Analysis of algorithms and problem complexity, Social and behavioral science
Virtual community recommender using the technology acceptance model and the user's needs type ^{xxii}	Recommendation system	Computer-supported collaborative work, pattern recognition

Algorithms

We collected a set of algorithms developed in order to support the measuring of some aspects such as reputation, contents quality and data mining.

Name	Focus	Fields of research
Extraction of reliable reputation information using contributor's stance ^{xxiii}	Social Network Analysis	Probability and statistics
Different aspects of social network analysis ^{xxiv}	Social Network Analysis	Probability and statistics, Social and behavioral science
Relationship Algebra for Computing in Social Networks and Social Network Based Applications ^{xxv}	Social Network Analysis	Pattern recognition, Social and behavioral science
Mining and Visualizing the Evolution of Subgroups in Social Networks ^{xxvi}	Social Network Analysis	Pattern recognition, Social and behavioral science
Analyzing and Visualizing Gray Web Forum Structure ^{xxvii}	Social Network Analysis	Pattern recognition, Probability and statistics, Social and behavioral science
Mining Virtual Community Core Members Based on Gene Expression Programming ^{xxviii}	Social Network Analysis	Pattern recognition, Social and behavioral science
Web Information Retrieval in Collaborative Tagging Systems ^{xxix}	Content traversal	Computer-supported collaborative work, Pattern recognition, Probability and statistics
Measuring quality of articles contributed by online communities ^{xxx}	Recommendation system	Probability and statistics
C ² – A collaborative recommendation system ^{xxxi}	Recommendation system	Computer-supported collaborative work, Probability and statistics
WMR – A graph-based algorithm for friend recommendation ^{xxxii}	Recommendation system	Computer-supported collaborative work, Probability and statistics
Labeled Link Analysis for Extracting User Characteristics	Recommendation system	Probability and statistics, Pattern recognition

in E-commerce Activity Network ^{xxxiii}		
Understanding Leadership Behavior in Human Influence Network ^{xxxiv}	Social Network Analysis	Social and behavioral science, Pattern recognition, Probability and statistics

Tools and software libraries

We found no product specifically supporting the Web 2.0 oriented software design. We found different tools, mainly UI-oriented software libraries.

Name	Type	Licensing	Description	Technology
Aflax ⁵	UI Software library	Open source (MPL)		AJAX + Flash
MochiKit ⁶	UI Software library	Open source (MIT)		Javascript
Ext JS ⁷	UI Software library	Open source (LGPL)	A fork of Yahoo! User Interface Library	Javascript
Scriptaculous ⁸	UI Software library	Open source (MIT)		Javascript
Yahoo! User Interface Library ⁹	UI Software library	Open source (BSD)	Provides implementation of the Yahoo! interaction patterns. It mainly relies on AJAX technology	Javascript
Elgg ¹⁰	Content Management System	Open source (GPL)	A CMS supporting some common features of Web 2.0 applications	PHP

In addition to the tools reported above, we hereby summarize APIs provided by some of the most popular social network web applications. In the following table we basically

5 <http://www.aflax.org/>

6 <http://www.mochikit.com/>

7 <http://extjs.com/>

8 <http://script.aculo.us/>

9 <http://developer.yahoo.com/yui/>

10 <http://elgg.org/>

indicate, for each platform analyzed, whether its API provides or not the possibility to get/set contents, social relationships and to send/fetch point-to-point messages to other members.

Platform	Content		Social relationships		Messages		Notes
	Retrieve / search	Upload / update	Read / check	Set	Fetch	Send	
Del.icio.us ¹¹	yes	yes	no	no	no	no	Tagging is also supported
Facebook ¹²	yes	yes	yes	yes	no	no	Query language provided (FQL)
Flickr ¹³	yes	yes	yes	no	no	no	
Friendster ¹⁴	yes	partially	yes	no	no	no	
Hi5 ¹⁵	yes	partially	yes	no	no	no	
Last.fm ¹⁶	yes	yes	yes	no	no	no	Events cannot be fetched/updated via API
Ma.gnolia ¹⁷	yes	yes	no	no	no	no	Tagging is also supported
Twitter ¹⁸	yes	yes	yes	yes	yes	yes	
YouTube ¹⁹	yes	no	yes	no	no	no	Query language provided

Conclusions

According to the results of our research, the issue of designing an online community appears to be treated mainly at a guideline level.

Many tools and algorithms have been developed for addressing specific topics related to social network analysis. Significant contributions have been made by Preece^{xxxv},

11 <http://del.icio.us/doc/api>

12 <http://wiki.developers.facebook.com/index.php/API>

13 <http://www.flickr.com/services/api/>

14 <http://www.friendster.com/developer>

15 <http://www.hi5networks.com/developer>

16 <http://www.audioscrobbler.net/data/webservices/>

17 <http://ma.gnolia.com/support/api>

18 <http://twitter.com/help/api>

19 <http://code.google.com/apis/youtube/overview.html>

showing how a community should have to be developed under a social-behavioral approach. Bishop^{xxxvi}, Cheng-Vassileva^{xxxvii} show how to foster members' participation proposing specific human-computer interaction patterns; in general there are many contributions to the HCI field: tools for online communities have quite consolidated interaction patterns, as accurately shown by Welie^{xxxviii}.

Our feeling is that this particular matter of research is not currently aimed towards new results, but is rather focused on wedging concepts from different disciplines (such as social network analysis, human-computer interaction, knowledge engineering, etc.) turning them into practice.

Patterns

In this chapter we outline some basic concepts distilled from our previous analysis of the virtual community state of the art. Therefore we propose a model for addressing common issues when designing generic-purpose virtual communities.

Our study brought us to identify two main classes of patterns: *front-end patterns* and *back-end patterns*. The firsts are behavioral patterns related to members' activities; we furthermore distinguish them between *navigation*, *contribution* and *social patterns*, depending on the kind of activity that they can use for. Back-end patterns, instead, are collateral, maintenance activities and workflows that the system should activate as reactions to front-end activities. Later in this chapter we discuss such patterns, explaining motivations and providing, where possible, examples to better understand them.

Basic concepts

There is a set of recurrent concepts that characterize different virtual communities regardless the purpose they've been designed for. We tried to outline them in order to obtain a common vocabulary made of reusable and easy-to-extend definitions.

We adopted an object-oriented approach in order to simplify the specification of such concepts and to exploit the constructs like abstraction and polymorphism.

The concepts we outlined and the we are about to describe are the following:

- Member
- Item
- Relationship
- Group
- Activity
- Reinforcement
- Message

Member

Unlike data-centric applications, where people are identified as simple users, in a community centered approach, people are classified as members. This distinction may seem shallow, or purely “philosophical” but encloses many relevant factors for the modeling approach.

The terms user recalls the concept in which applications' actors are only lucrative agents which use “something” without adding any value of their constructive interaction within the application; this interpretation lacks in term of expressiveness

because it doesn't consider the fact that, in a virtual community environment, members change the state of the application context by their behavior. In such a context, members are characterized by their behavior, credibility and are directly responsible for their actions within the group.

In order to model these complex aspects with an object-oriented approach, it is necessary to simplify the reality with a good degree of approximation: we can represent a member by a profile according with it is possible to define a *member reputation index*. The implementation of this index can be open to many solutions, we don't want to find here a possible implementation but, for sure, each proposed implementation has to provide a *change reputation* operation according to which it is possible to adjust any member's *state* on the basis of a particular *judgment criterion* determined by community designers and/or managers. While reputation is a quantitative index, the *state* represent a formal, qualitative definition according to which it's possible to aggregate members in terms of ranges of reputation. For instance a member whose *reputation index* ranges between 1000 and 3000 could be judged by community rules as a lurker, thus this member stays, because of her activity, in a lurking state.

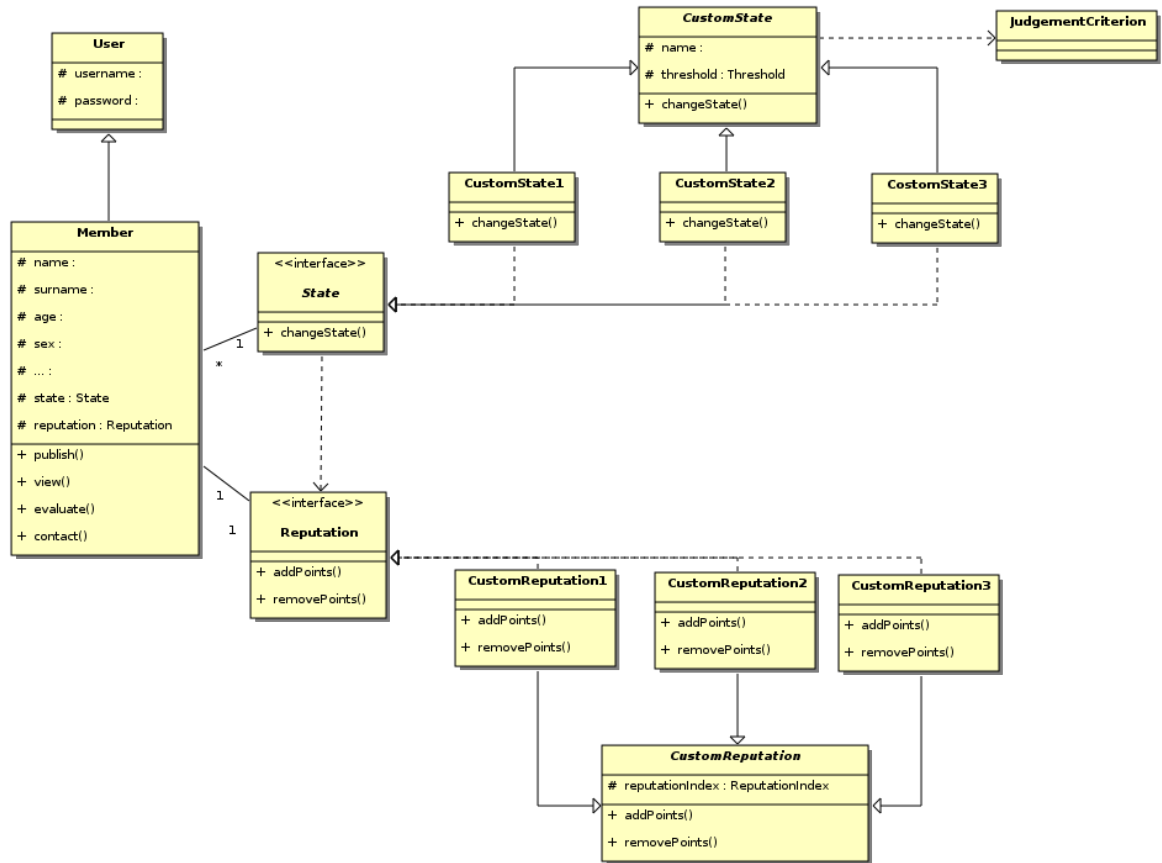


Figure 1: Representation of the concept “Member” through a class diagram

Item

We call *item* a generic element of interest within a virtual community. Many communities base their activity on the plain sharing of items; could they be photos, videos, bookmarks or reviews of concerts and albums, items, if good in quality, subject the community to a network externality. In other words, an item is something whose the simple fact of being there impacts positively on the value of the whole virtual community.

An item has always an *owner* (the member who published it) and a *publication date*, that indicates the moment since when it starts existing; in some cases items can also be a container for other items and organized into hierarchies.

Let's take, for example, an online forum community in which members exchange asynchronous messages around different topics of interest. Generally, in such communities, a discussion is started by a member with a question, a poll or a sentence

to which other members are invited to reply. In this case we can see every single message as an item while contributed answers are other items contained into the one who generated them.

Of course, especially when items are directly contributed by members, a very relevant issue is *quality* that could be measured using a *ranking system*; in the forum example stated above, a helpful, clear and straight-to-the-point message is much more valuable for the community than hundreds of orientation statements like “what's up?” “hello”, “anybody in here?” and so on. We will deal with this aspect later, but leaving apart the concept of quality, we can affirm that the presence of every message is surely valuable for the community.

A more generic way to classify items is relying on the use of *flags* to describe some aspect about them. Flags need to be designed in advance but may offer a quicker way to classify them at runtime. This could be useful, for example, when dealing with socially-contributed contents: in these cases there is always the possibility for members to upload offensive/explicit/inadequate content that should be subject to moderation: using a flagging mechanism can let community members be in charge of check them off.

Another common aspect in virtual communities is the matter of item *licensing*. We will not deal with this aspect now, but it's important to outline it in order to better clarify most common concepts.

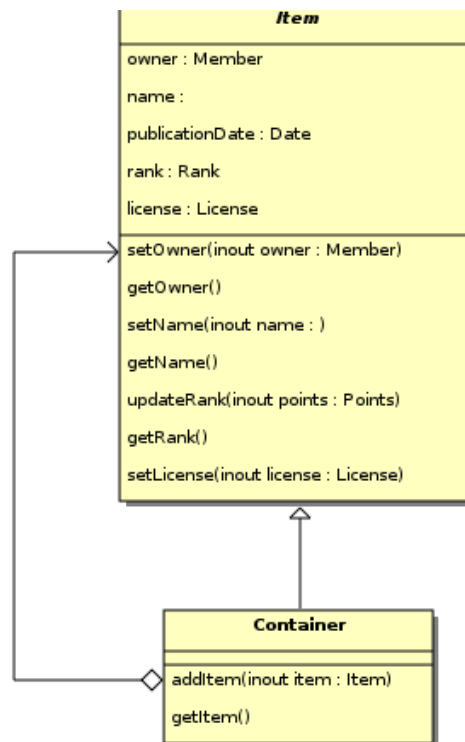


Figure 2: Representation of the concept “Item” through a class diagram

Relationship

The term relationship is open to many interpretations, for our case a relationship can be considered as follows:

1. Member-to-member relationship
2. Member-to-item relationship
3. Item-to-item relationship

We call them respectively *member relationship*, *hybrid relationship* and *item relationship*. These three types of relationships are characterized by the same features:

- *typology*
- *direction*
- *explicitness*

By *typology* of a relationship we mean the “sense” of the relationship. For example two

members could be tied by a friendship or by a formal relation such as manager-employee. Examples of member-to-item relationship typology could be the owning, the usage, etc.

Direction describes the flow between the relationship endpoints. For instance one member can consider another one a “friend” while the other could not manifest the same feeling. The necessity to describe direction of relationships is fundamental for analyze subgroups dynamics.

Finally *explicitness* means two possibility: *explicit relationship* and *implicit relationship*. Explicit relationship are relations tailed by members by formal actions, for instance a member who shares his Google doc with someone explicitly establishes a relationship between himself and the invited person. In the latter case, an implicit relationship has not a performer but it exists for similarity. For similarity we intend how two ore more objects are close by a similarity rule. For instance a member who likes Beethoven compositions is more similar to someone who likes Chopin than someone who listens only to techno-music. So the two who like classical music could be potentially friends and implicitly they belong to the classical music lovers group.

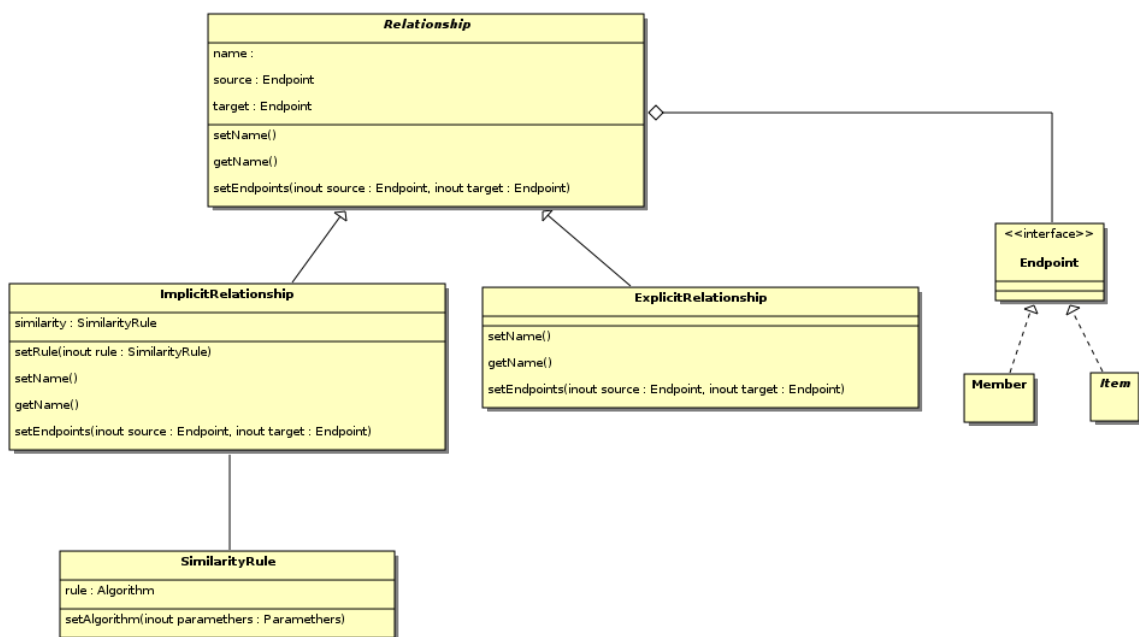


Figure 3: Representation of the concept “Relationship” through a class diagram

Group

In the classical design techniques, a **Group** represents a static collection of people which will access to similar functionalities and data into the application domain. By

this assumption, designers have to predict the users typology at design time, and at the same time they have to define the access policies by site views and modules.

We argue that this approach is not enough to map the real community behavior because from a conceptual point of view it doesn't consider the sub-group dynamics typical of a virtual community environment. More precisely it has the following limitations:

- The access policies are “static” and aren't oriented to map the community dynamics
- The access policies are defined by the application designers and not by the community participants

We propose to extend the actual group definition, in order to consider two main phenomena:

- In a community, members set relationships within themselves forming aleatory subgroups that aren't predictable at design time because they are subject to non deterministic clues, such as people character, external factors tailored to the real life... etc.
- In a community users are implicitly aggregated by similarity, these relationships form implicit groups which became very useful for many different reasons such as community behavior monitoring, groups dynamics tracking, ad-hoc marketing strategies... etc.

To model these new requirements we propose to define the concept of group as an abstract object, this object can be implemented by two possible definition of group:

1. *functional group*: This group is the classical group defined at design time, such specialization is used to distinguish member on the base of their functional role in the application. Typical examples of functional groups are for instance the group of Administrators, the group of registered users, the group of editors and so on...
2. *behavioral group*: This group is defined by the community itself, they are defined at run-time on the base of the members activities and behavior. They could be differentiated into two sub definitions: *explicit group* and *implicit group*.

Explicit group is a group whose existence is controlled by the members, it is a formal declaration made by some members who share a common space into the community environment. Typical examples of *explicit groups* are “classical music lovers” on Last.fm, or generally groups of interest which famous examples are available on Flickr,

Wikipedia, or Google Groups.

Implicit group is a group which existence is given by *neighborhood criterion*, according to which members are aggregated by similarity. A typical implementation of an *implicit group* is available on Last.fm, the system proposes to the logged member a set of “potential” friends whose musical taste is similar to the logged person; such persons form an unaware group.

Such typology of groups assumes an high relevancy specially in busyness context, but requires a specific studying in order to define useful and scalable adaptive filters in order to implement what we call *neighborhood criterion*.

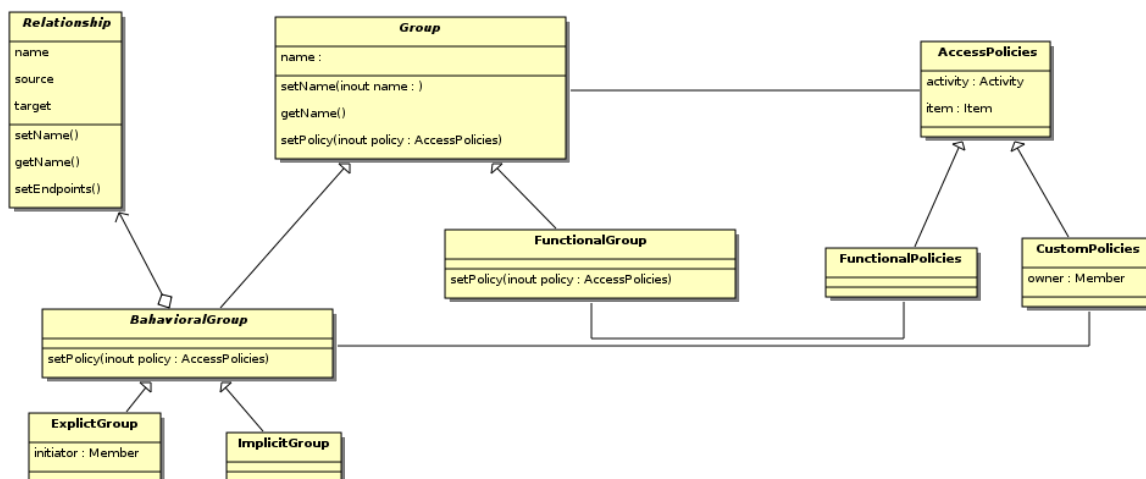


Figure 4: Representation of the concept “Group” through a class diagram

Activity

We’ve already underlined how items can add value to a community. Thinking about a community in its whole, what is really valuable is the act itself of contributing items that is performed by members, usually on a voluntary basis. Because in general items lose their interestingness in the course of time, without such activities the community would lose its main purpose and collapse.

More generally an activity is the performance of an action that is, to a certain extent, relevant for the development of the community. Many activities could have a positive impact on a community: contributions enrich it, invitations broaden it, tagging improves metadata accuracy, rating helps in determining the real satisfaction of members about items, and so on; we call such activities *reinforcing activities* because they support the development of the community and, because of this, need to be properly fostered. Conversely, some activities may explicitly go against the growth of

the community or, more often, dispel the interest of members toward the community itself; we call such activities *controlled activities*.

A relevant concept, especially for reinforcing activities, is that of *reward*. As already explained in the previous chapters, social and behavioral sciences agree that users feel more likely to contribute to a community when they feel they are appropriately rewarded. This is why is very important to foster participation specifically designing a reward system to passionate users and attract their contributions.

A very clear example of these concepts is noticeable in the Yahoo Answers community, in which members can submit questions to the community around a topic of choice. Any user is allowed to propose a question or to answer other's ones. Every member has a personal score; in order to prevent a measureless, chaotic posting of questions, the act of asking is in facts a controlled activity, because members need to spend part of their points for submitting a question to the community. Conversely, the act of giving answers to a question is fostered (every member gets a score reward only for giving an answer to a question): according to our definition, this can be considered as a reinforcing activity.

The *reinforcing* versus *controlled* classification is aimed to distinguish activities from a “perceived value” point of view, but we can still distinguish activities by purpose. All those activities aimed to add a tangible contribution for the community in terms of knowledge (a post on a blog, a file uploaded, a section on a wiki, etc.) are *contribution activities*, while all the activities oriented to enlarge the community mass or aimed to increase community sense of belonging (members' exchange of messages, invitations to external people to join the community or specific subgroups, etc.) are *social activities*.

As stated before, an essential issue when dealing with socially-contributed contents is determining the quality of every item. Evaluation can be an explicit activity, inferred from the activity performed over an item, or a combination of them.

Reinforcement

The concept of reinforcement of an activity resumes “how good” is to be object of that activity, directly impacting on reputation of users and relevance of items.

Let's take for example the YouTube community, and let's think the establishment of a subscription relationships. YouTube members can upload their own video clips: that assimilates every member's personal page to a video channel, and this is exactly how YouTube names members' personal pages. The act of subscribing to a member's

channel means that, to a certain extent, the content she provides is interesting. If many users do the same, that make the user a good user, because that means that her contribution are very valuable for the community.

Let's take now, for example, the Flickr photo sharing community. Within this community, users are allowed to communicate both commenting each others' photos and through asynchronous messages. This could expose users to undesired comments, messages, so providing a mechanism for blacklisting members is in general a good practice.

From our point of view, these two examples are very similar: they are both social relationships, because both of them consist of activities made by a member (A, performer) towards another (B, object). The main difference is exactly what we call *reinforcement*: the first social activity expresses a good opinion for B, because it entails a certain esteem of A towards B; because of this, the activity of “subscribing” has a positive impact on B's reputation ($reinforcement > 0$). For the activity of blacklisting the opposite holds while the latter state a bad opinion for the object member, so it has a negative impact on B's reputation ($reinforcement < 0$).

The same concept can be also extended to the case of *contribution activities*, for whom, instead of members, objects consist of items. Instead of talking about *reputation*, we use the term *relevance*, but the meaning is absolutely comparable.

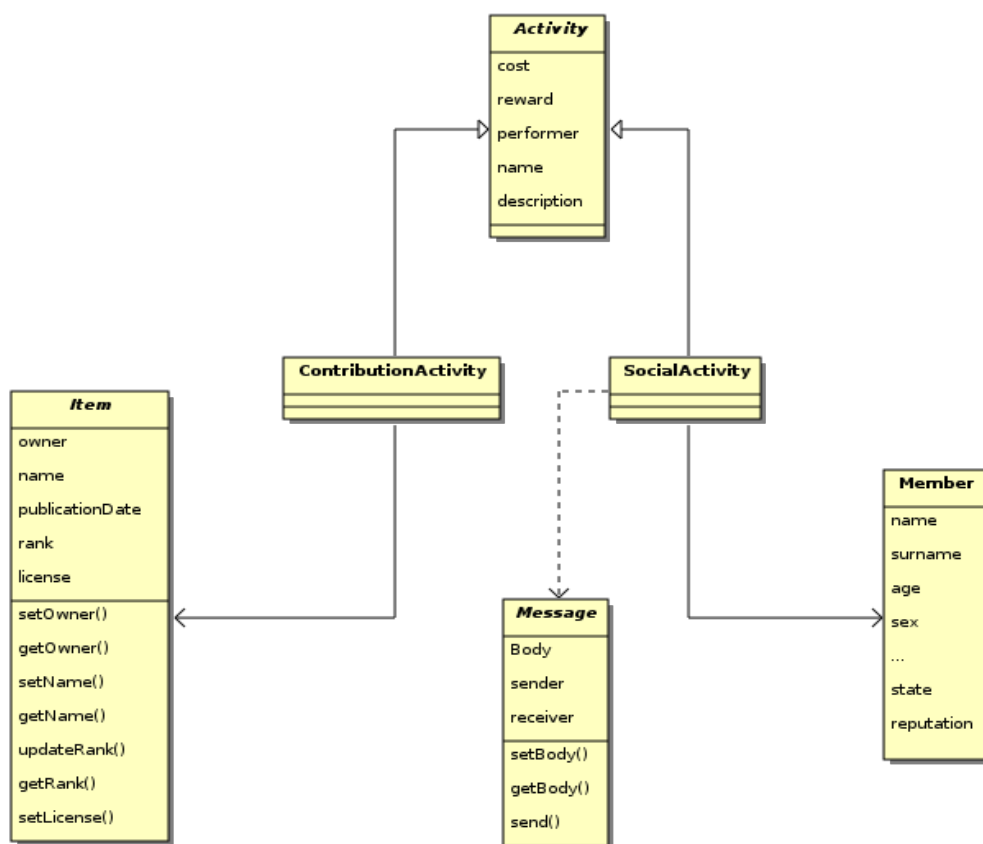


Figure 5: Representation of the concept “Activity” through a class diagram

Message

We call message a generic point-to-point communication between members. Differently from items, messages do not add value to the community, but only let users communicate beside their activity within the community.

Even if the necessity of such concept may be unclear when thinking at the examples cited above, its usefulness is easier to realize when dealing with communities within which main items of interest are not themselves “messages” as happens for forums or questions. In the Flickr Photo Sharing community, for instance, members can communicate via private messages that are automatically forwarded to each recipients' mailbox, for setting and reinforcing mutual ties and relationships. Also Google Docs allows users, when working cooperatively on a document, to exchange real-time text messages via a Jabber client directly embedded into the system interface. This are only two examples: many other social communities offer similar services for letting users communicate independently from the purpose of the community.

The community model

Motivation

Our intent was to figure out a reusable and easy-to-extend schema capable of representing the main aspects related to a virtual community, adaptable to different applicative contexts.

The model we propose can be adopted for developing from scratch new online community or can be integrated into existing traditional web applications (such as plain portals or similar).

Pay-Perform-Reward chain

We consider a virtual community as a complex event dependent system whose behavior is dominated by members activities. Such activities are defined by a specific definition which describes the purpose, the costs, the value, the rules and the interested items involved.

For describing this behavioral model, we introduce the Pay-Perform-Reward chain. The PPR chain represents the core concept that inspired us to develop this model, we call it chain, because it is a sequence of correlated events that happens every time a member interacts with the community, or more easily every time which a member performs an activity.

The chain consists of seven steps:

1. Permission check
2. Affordability check

3. Payment
4. Activity performance
5. Evaluation
6. Reward
7. Reputation adjustment

The meaning *permission check* step is trivial: before a member accesses to a particular activity a *community control system* evaluates if the requirer has the agreement to do that. If the requirer is allowed, then the system provides to check if she has the means to pay for the execution of that activity. We call such step *affordability check*: it is not necessary for every kind of activity, but it is needed for every activity that requires a payment. By imposing a *payment*, the user is invited to be prudent in order to don't run trough her credit and reputation unnecessarily.

After the payment the member is allowed to *perform* the activity, it includes a huge heterogeneity of actions, for instance it could be the sending of a post, the upload of a a song or the invitation for someone to join a discussion. Each activity performed has a potential added value for the community, which we simply call *contribution*.

That contribution is subject to the community *evaluation* which fix the real value of the contribution in terms of quality and popularity, the evaluation can be explicit, so the members have a particular activity according to which they can assign a score to the contribution, or it can be implicit for instance it can be calculated on the base of the popularity of the contribution or furthermore it can be calculated by specific algorithm like HITS²⁰ or other custom methodologies.

Next, the community system provides to *reward* the member for her/his effort, the reward can be implemented in many ways, it could be a simple performer's credit update by adding or removing a fixed score or it can be weighted considering many aspects such as the member reputation, the reputation of the evaluators or other weights defined by customs rules.

Finally, the community systems provides to *adjust* the performer's reputation by evaluating the her/his credit updated by the action performed.

The steps listed below are to be considered a pure abstraction of what happens in a virtual community, it is not a mandatory pattern; in facts it can be whether simplified or complicated to meet the community designers' needs. The PPR chain can be

²⁰ Hypertext Induced Topic Selection (HITS) is like analysis algorithm that rates Web pages for their authority and hub values. Authority value estimates the value of the content of the page; hub value estimates the value of its links to other pages.

changed by adding or removing steps in order to better follow the community designer's policies, anyway we argue that in order to maintain a minimal moderation of the community three steps are encouraged to be implemented:

1. Payment
2. Activity performance
3. Reward

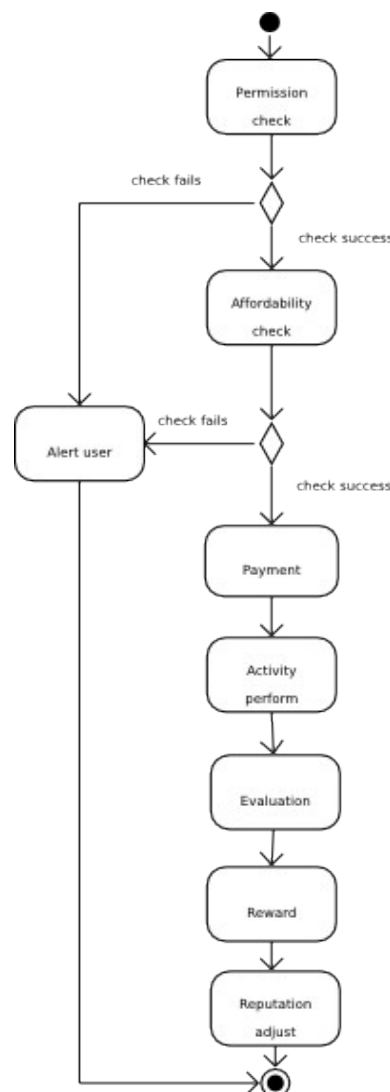


Figure 6: The PPR chain

In this paragraph we don't explain how to physically implement such features, because we'd like them to be completely open to any interpretation. Our model is only a conceptual representation of a possible community environment.

Pay-Perform-Reward runtime

In order to implement a system based on the PPR chain, it is necessary to fix some conceptual components that are needed to build a PPR run-time.

As we mentioned before, the core concepts of the PPR is the activity, every activity has a proper definition according to which it is possible to understand its nature (what's the action do), its permissions (who is allowed to perform it), its cost (how much a member has to pay to perform the activity), its correlated items (what are the entity involved in the activity) and finally its default potential value (its a default reward accreditation which will be adjusted by the community evaluations).

Any activity definition, described by an unique id, is stored in a proper archive we call *activity repository*. Such repository is useful in order to maintain a complete overview of the activity definitions; by this way community designers are able to quickly access any activity and configure permission policies and fostering strategies.

The next relevant component is the *item pool*, it is the container of every content published or loaded in the community environment. As described before, an *item* is a complex object, it could be a *container* of other items or it could be a simple atomic object. For instance, by our assumption, following our item definition, a blog is both an item and a container while the posts are simple atomic items.

Members access to the *item pool* through specific activities which are defined into the activity repository, the access permissions of that items are defined both at design time by the community designers and obviously at run-time by the item owner. We call the first *imposed policies* while the latter are called *custom policies*, generally we refer to them as *access policies*.

An access policy is a rule which formally describes who is allowed to perform a specific set of activities onto a specif item. This simple “who-how-on what” statement is defined by the relationships between *group*, *activities* and *items*.

As we mentioned in the previous paragraph, a group could be both a group defined at design time by the designers or a group defined by the community members, these latter groups we called *behavioral groups* can be formed explicitly by the members by a proper activity or they can be formed by a semi-automated system which builds implicit groups on the base of intelligent aggregation criterion. These two specializations of custom group are respectively named *explicit group* and *implicit group*.

A custom group is an aggregation of relationships, a *relationship* is an atomic link

between member-member, member-item or item-item. Each relationship is stored in the *relationship pool*. Such pool represent the network among members and it can be exploited for many purpose such as recommendation systems, sub-groups dynamic analysis and other complex inspecting techniques derived from the social network analysis.

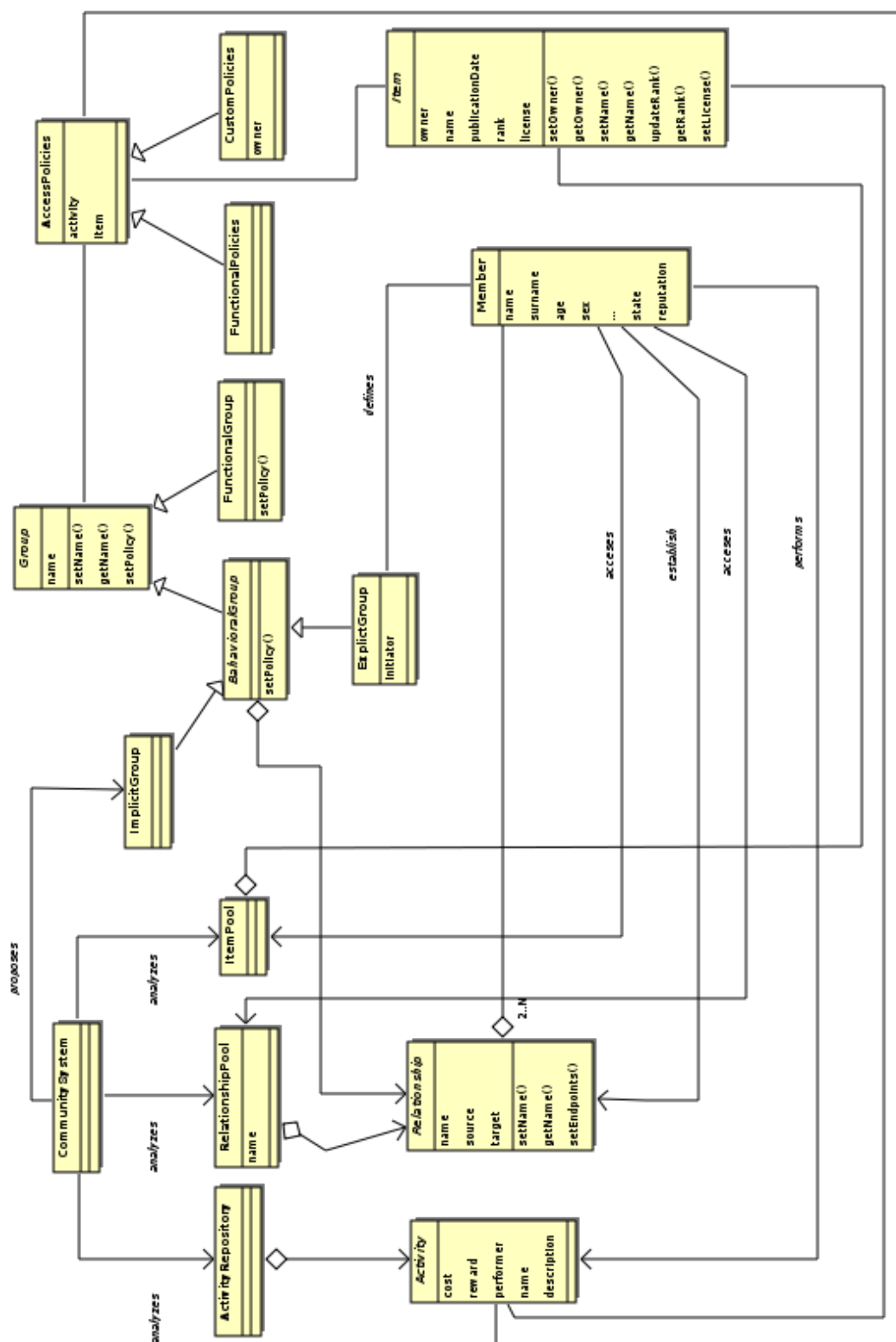


Figure 7: Conceptual class diagram for the PPR runtime

Front-end patterns

Navigation patterns

Navigation patterns describe activities directly related to the exploration of contents published within a community catalog. Regardless the way such contents are published, there are modalities of being introduced to contents that personalization requirements could need to address.

Aggregation

Name

Aggregation

Also known as

Mash-up.

Problem

User may need to create and manage fully customizable mash-up pages, containing independent blocks whose contents and/or services are provided by external platforms.

Solution

To give the user the possibility through proper interface commands to embed such contents relying on formal representations. Furthermore, where depending on members' preferences, users should be able to sort, customize and remove every single imported element.

Related patterns

Exportation, Permission check, Syndication.

Discussion

Most popular communities have quite specific functionalities that are often made available to other applications through the use of Web Services; of course a programmatic interface should be developed ad-hoc for every external service that the community needs to support.

This pattern has not a specific relevance to the virtual community in exam, but can be useful for integrating new services provided on other platforms.

Examples

The iGoogle page presents a clear example for this pattern. It offers to members the capability of aggregating content from different sources through RSS summaries, furthermore it offers many ready-to-use “widgets” that a user can dispose freely within her personal page.

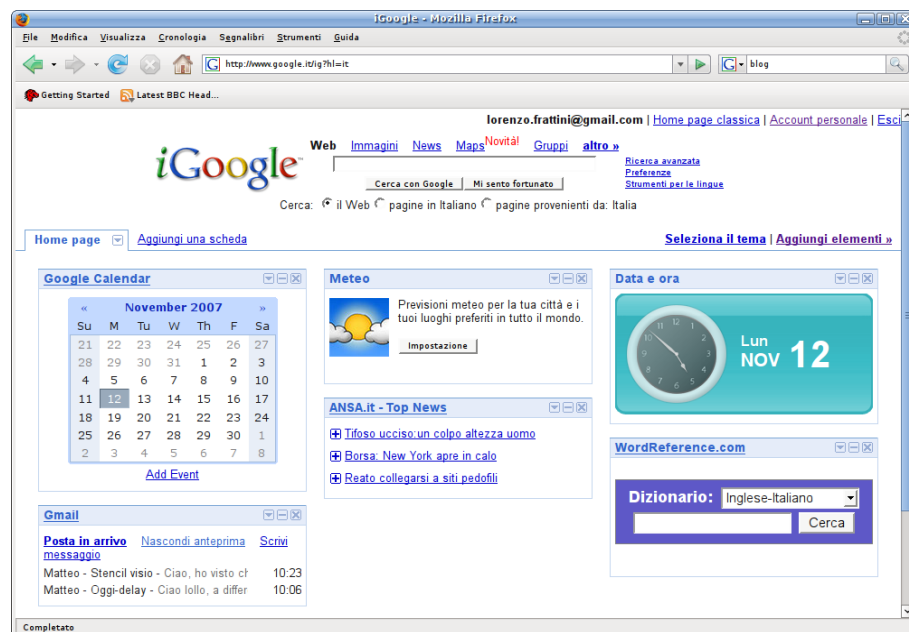


Figure 8: iGoogle mash-up

Browse by connections

Name

Browse by connections.

Problem

To give the possibility to browse contents and relationships also when connections are indirected, and there are several degree of separation between them.

Solution

Members within communities set relationships that can be mapped onto social networks in which every member corresponds to a node, while relationships correspond to directed edges between nodes. Sometimes applications let users browse relationships not only when directly established, but also if users are indirectly connected and a number of edges need to be traversed to link one to another.

Related patterns

Permission check, Relationship setting.

Discussions

The representation of a social network can be onerous in terms of computation and is in general a critical issue for systems scalability; nonetheless it can be a very powerful strategy to make the community grow through a more effective awareness of the community they are part of. This pattern can be exploited also for proposition of contents, and implemented either by showing the effective distance between members or, more often, hiding this detail to the final user.

Examples

Linked-in is a social network site used for professional networking. Through the Linked-in network users can maintain a list of contacts of people they know and trust in business. This allows members to gain an introduction to someone else through a mutual trusted contact; this can be useful to find jobs, people and business opportunities, recommended by anyone in the user's contact network.

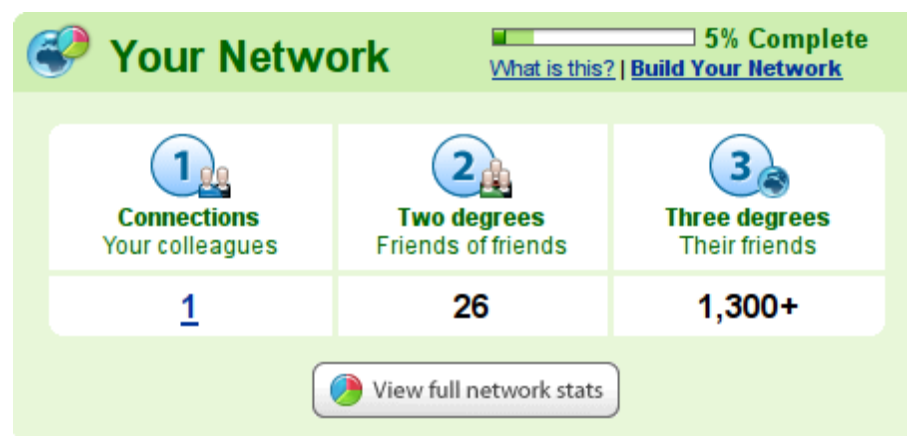


Figure 9: Browsing by connections on Linked-in

Browse by tag

Name

Browse by tag.

Also known as

Folksonomy.

Problem

Sometimes, especially for big catalogs of items, users feel the need to access and search content according to a more “personal” perception rather than a rigid organization like the one imposed by the plain content topology.

Solution

The solution is to give users a list of tags represented by significant keywords, in order to access a subset of contents related to that specific keyword. More important tags should be easily recognizable by users in order to help them to predict the relevance of descriptions; this could be achieved sorting tags, indicating the number of item contained within each label or highlighting each tag with a different graphic appearance (for instance: text size) depending on the number of occurrences it has.

Related patterns

Organization, Permission check.

Discussion

Tags provide a very democratic description, and fast-access mechanism when browsing items, even if a user doesn't have a deep knowledge about the structure of the content map. This advantage is counterbalanced by an increased difficulty for users to mind-

Methodology and patterns for developing community-based web applications with a model-driven approach

map the web application structure; this could cause a lost of orientation.

Examples

Del.icio.us is a web application that allows users to store and share bookmarks describing them with tags. Each user has its own tag container which works as introductory act for exploring stored bookmarks. Every tag can is presented to the user with a size that is proportional to the number of bookmarks it has been used for.

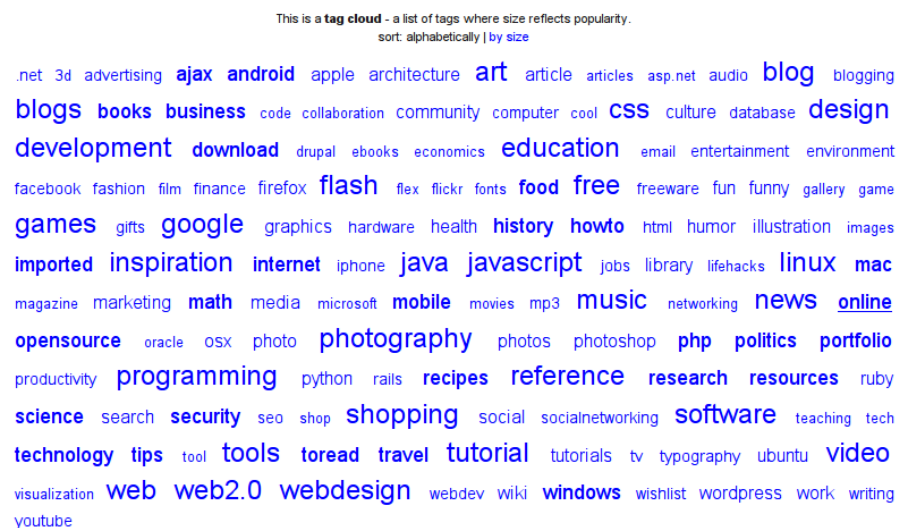


Figure 10: Most popular tags on Del.icio.us

Contribution patterns

Contributions are those activities that members perform, which entail an added value for the community. Such added value can consist of a tangible addition to the item catalog, but could also consist of an improvement to the community capability of understanding the real quality of items that already exist.

Exportation

Name

Exportation.

Problem

Export contents to other web applications.

Solution

The application platform of the community needs to provide specific functionalities aimed to achieve the performance of some tasks made available as web services API by the target applications.

Related patterns

Importation, Permission check, Relevance adjustment, Reputation adjustment.

Discussion

Most popular communities have quite specific functionalities that are often made available to other applications through the use of Web Services API; of course a programmatic interface should be developed ad-hoc for every external service that the community is willing to support.

Exportation can be figured as a promotion effort performed by members which, depending on community policies, could be treated as a normal reinforcing activity and rewarded.

Examples

Many blogs include, beside every post, widgets for posting such entry to *Digg*, *Slashdot*, *Del.icio.us* and similar.

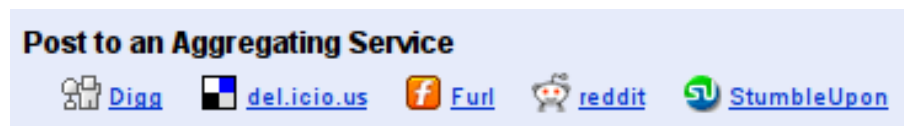


Figure 11: Exportation of a video on YouTube

Flagging

Name

Flagging.

Problem

Sometimes communities need a quick method to let users mark an item or a member to draw attention to it, in general for moderation purposes.

Solution

To implement a flagging strategy that 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 and/or bothering users.

Related patterns

Notification, Permission check, Reputation adjustment.

Discussion

This pattern should be intended as a part of a broader workflow, that more in general describes the process of reviewing content for moderation. Because virtual communities can grow fast, self-sustainment is a requirement; nonetheless, due to the nature of the evaluation of some criteria that cannot be inferred automatically (like the possibility for an item to be offensive), giving the community the possibility to point out an entity for a moderation review, could be effective.

The main advantage of this pattern is the easiness by which members can point out dysfunctional behaviors, signaling them to moderators. Conversely, the main disadvantage is the potential abuse that such activity could be subject to, which can bring to a stall of the moderation system.

A variant of this pattern is the *ban* activity which allows a user to autonomously deny to another one the performance of any activity towards her or her items.

Examples

A clear example is the flag mechanism that users can use to mark potentially disturbing images on Flickr. This action entails a request of review from moderators for the user that published the original picture.

Organization

Name

Organization.

Also known as

Grouping, Tagging.

Problem

Users should be able to organize contents within the application into self-contained subsets, reflecting their very personal mind-map of contents.

Solution

The solution is to give users the possibility to describe contents using a simple, flexible containing mechanism that links in a many-to-many relationship containers with

items. This could be used either for describing a user's personal set of items or for describing the whole application catalog: the latter case entails the realization of a collaborative mechanism that gives users the possibility to describe contents using simple, reusable and significant containers.

Related patterns

Browse by tag, Notification, Permission check, Relevance adjustment, Reward, Syndication.

Discussion

This pattern is a generalization of the tagging mechanism, that could also be used for creating more complex subsets of items. This approach can be effective for meta-describing items, also if they're complex, because in a way, it could bring to a shift of the content topology from physical structure to community perception. Conversely, especially when the container hierarchy is deep and unbalanced, this may sensibly impact on the orientation capability of users during the exploration.

In a container-mechanism, every user should be able to add her own container for every item of interest or, if in a collaborative environment, to add items to container declared by others. Because the relationship from containers to items is many-to-many, every container acquires importance and representativeness as its amount of content grows. This is very relevant especially when the domain of the aggregation is the whole item catalog.

A very common implementation of this approach is the *tagging* mechanism. Tagging is a collaborative container mechanism, that implements a flat structure in which tags (short labels) are direct containers for items. Every item can be described with more than a tag, which means that it can be referred by more than one single container.

Grouping is another variant in which the container-item relationship is 1:N.

Examples

On Flickr users are allowed to create their photo sets as aggregations of their own photos. Photos can be added by to a set by a drag-n-drop selection from a list of all photos contributed.

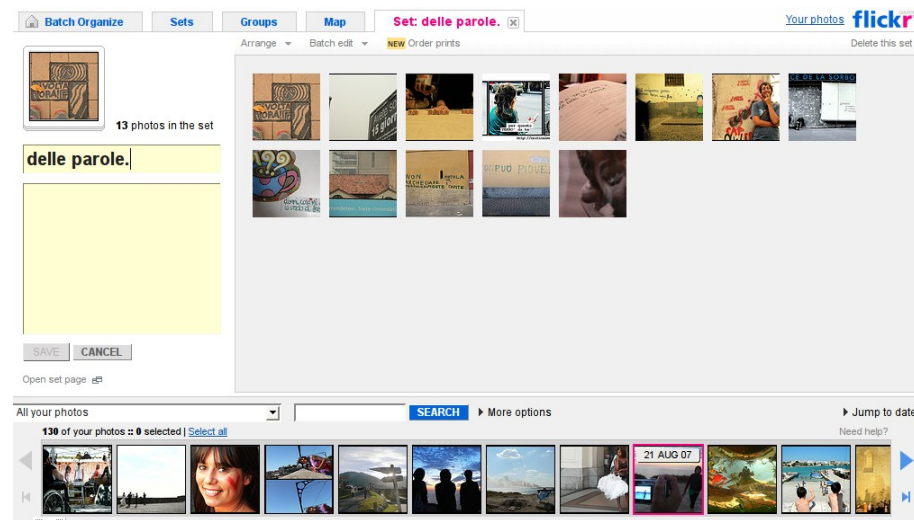


Figure 12: Organization of photos on Flickr

Publication

Name

Publication.

Problem

Members need to publish content within the application catalog.

Solution

An activity should be explicitly designed for letting users upload their contents. Publication should be considered, at least in principle, as a reinforcing activity, because the number of items directly impacts on the value of the community.

Related patterns

Organization, Payment, Permission check, Reward, Syndication.

Discussion

This pattern is applicable whenever it's needed to give members of a community, regardless the possibility to freely publish items in the community. Publication is an activity of contribution subject to permission and that could be, depending on the community policy, configured as a reinforcing and/or controlled activity.

A number of operation needs to be performed at back-end side, when performing a publication:

- If the publication is configured as a controlled activity, the member performs a *payment* to gain access to the publication activity

- If the publication is a reinforcing activity, the performer gets a *reward*.
- Syndication for items and/or containers should be updated

Examples

Almost every virtual community has a contribution activity that corresponds to the performance of an item publication. Here are some examples:

- Posting a blog entry on Blogger.com
- Uploading photos on Flickr photo sharing
- Listening to music on Last.fm
- Rating a piece of news on Slashdot
- Answering an existing question on Yahoo! Answers

Rating

Name

Rating.

Problem

Community members should be allowed to give their opinion about the quality of items published by others.

Solution

To implement a rating mechanism that allows users to explicitly state quality of an item in a scale that need to be properly set.

Related patterns

Evaluation, Permission check, Reward.

Discussion

The act of rating contributes in determining the actual *relevance* of an item within the community. Each rating should be weighted on the *reputation* of the user who gave it, in order to balance the rating on the authority that the community recognize to that single member.

This is a pure evaluation activity, corresponding to the plain invocation of the *evaluation* pattern in back-end. Furthermore, because of the constructive nature of the activity, rating can be rewarded.

Examples

YouTube gives registered users the possibility to mark any video with a number of stars in a scale from 1 to 5. Such rating is used to infer video interestingness and to propose top rated ones.



Figure 13: Rating of a video on YouTube

Recommendation

Name

Recommendation.

Also known as

Suggestion.

Problem

Members could wish to recommend an item to others (members or non-members).

Solution

To implement a pattern similar to *invitation*, but with in addition the possibility to specify the object of the suggestion (the item that the member is going to recommend).

Related patterns

Invitation, Notification, Permission check, Reward.

Discussion

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 normal reinforcing activity and rewarded.

The suggestion message can be forwarded to another member of the community, through the *notification* pattern.

Examples

In YouTube any video can be recommended to friends by simply choosing the “share” feature right under the vision box. The recommendation is sent as an ordinary mail message.

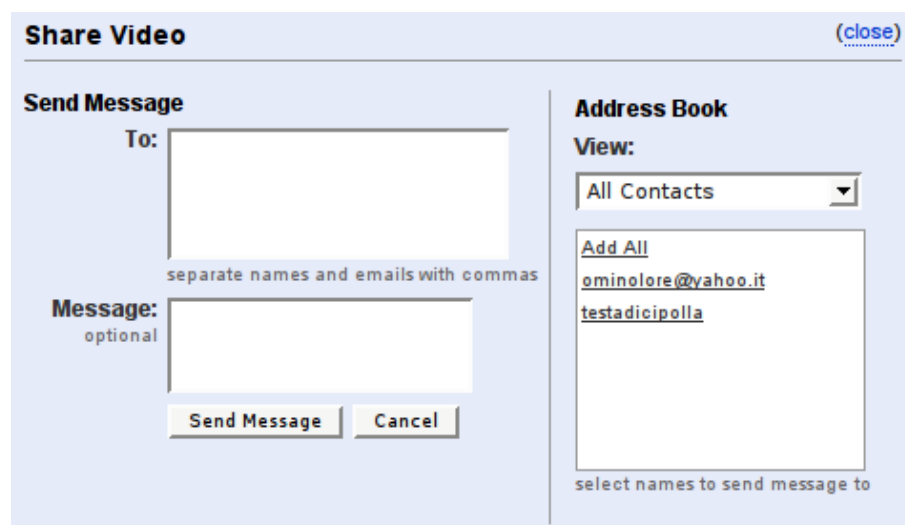


Figure 14: Recommendation of a video on YouTube

Social patterns

Social patterns describe those activities that may not directly add a tangible value to the community in terms of contributions, but which, being aimed to social interactions and demonstrate members' involvement.

Group creation

Name

Group creation.

Problem

Users need to create groups of interest in order to set a shared environment for communications and joint activities.

Solution

The community platform should give members the capability of setting up a shared environment capable of managing the sharing of items and the communication between members and where members can be assigned to specific roles.

Related patterns

Payment, Permission check.

Discussion

Forming groups of interest is a common feature for virtual communities. In general

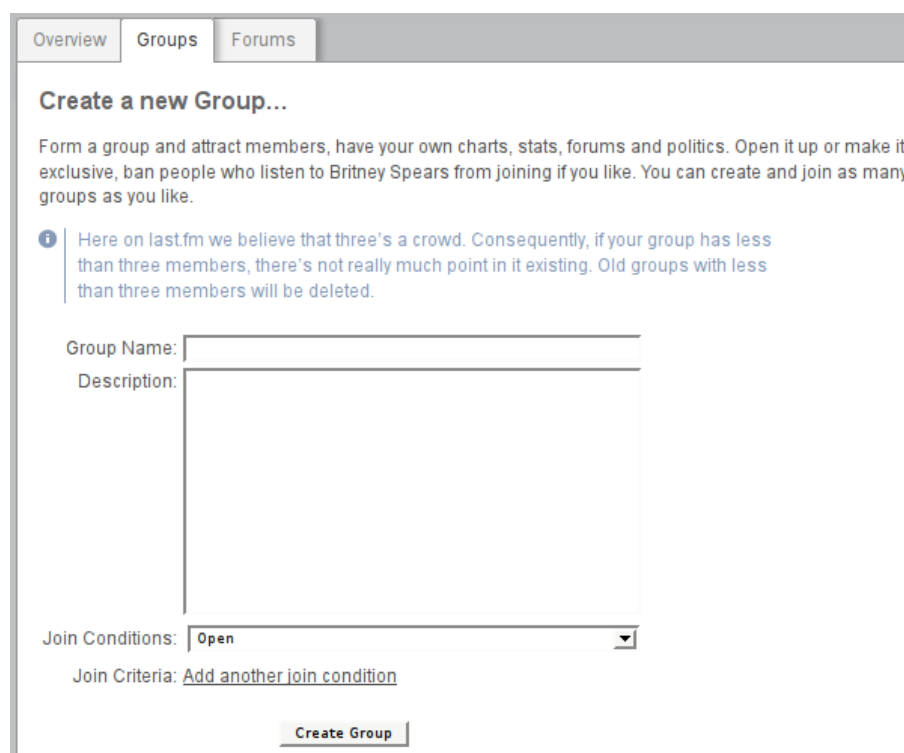
groups are characterized by a join policy (determining how members can participate to the group) and roles (determining who is in charge to do what).

Regardless the specific context of implementation, spontaneous groups are an instruments whose main goal is to establish a perception of the community, focusing both on sociability and contributions. Activity within forums could be also used for inferring information about a user's amount of activity and, more indirectly, to calculate her reputation within the community.

The creation of a group can entail different back-end patterns. The creation of a group, for example, could be a controlled activity in order to limit the amount of groups created within the community. Furthermore the creation of a group could be subject to permission check, in case the activity is not open to everybody.

Examples

Last.fm allows users to spontaneously form groups with an open join policy towards other members.



The screenshot shows the 'Create a new Group...' form on the Last.fm website. At the top, there are three tabs: 'Overview', 'Groups', and 'Forums'. The 'Groups' tab is selected. Below the tabs, the title 'Create a new Group...' is followed by a paragraph explaining the purpose of groups. A blue information icon is followed by a note about the minimum number of members required for a group. The form includes input fields for 'Group Name' and 'Description'. Below these is a 'Join Conditions' dropdown menu set to 'Open', and a link to 'Add another join condition'. A 'Create Group' button is at the bottom.

Overview Groups Forums

Create a new Group...

Form a group and attract members, have your own charts, stats, forums and politics. Open it up or make it exclusive, ban people who listen to Britney Spears from joining if you like. You can create and join as many groups as you like.

i Here on last.fm we believe that three's a crowd. Consequently, if your group has less than three members, there's not really much point in it existing. Old groups with less than three members will be deleted.

Group Name:

Description:

Join Conditions: Open

Join Criteria: [Add another join condition](#)

[Create Group](#)

Figure 15: Group creation on Last.fm

Group participation

Name

Group participation.

Problem

Users need to join groups of interest formed by others in order to participate to a shared environment with communications and activities.

Solution

The community platform must give members the possibility to join groups.

Related patterns

Invitation, Notification, Permission check, Reputation adjustment.

Discussion

Joining to a group may appear as a trivial activity, but entails a number of back-ends activities that needs to be taken into account indeed. A group participation could be autonomously and asynchronously performed by a member, but it could also be performed against the acceptance 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.

If many users participate to a group, that means that probably it is a good group in quality and, in principle, the community should recognize the value of the foundation of such group to the member who started it; back-end should then activate the *reputation adjustment* pattern, by which the calculation of her reputation is updated.

Examples

Within Flickr, group participations of members may occur on a spontaneous basis or against an invitation explicitly sent by another member of the community that already joined the group.

Invitation

Name

Invitation.

Problem

Invite non-members to join the community.

Solution

Implement an invitation mechanism for letting members invite people from the outside.

Related patterns

Payment, Reward, Notification.

Discussion

Invitation is a potentially reinforcing activity, because it directly brings to a growth of the community; because of this, it could be designed with a reward for the inviter against the occurred subscription of the invited user. Furthermore the acceptance of an invitation should be notified to the member who performed it.

In general this pattern can be viewed as a workflow, described by the following activities:

1. A member (A) invites a friend (B) to join the community
2. The system forwards a message to (B)
3. If (B) accepts the invitation:
 1. The system rewards the user (A)
 2. The system notify the user (A) of (B)'s acceptance
4. If (B) rejects the invitation:
 1. The system notifies the user (A) of (B)'s refusal.

A possible variant of this patter is applicable for inviting members of the community to join groups of interest within the community itself.

Examples

The Gmail invitation mechanism easily allows users to invite others to join the service. At the act of invitation an automated message is forwarded to the recipient by the system. If the addressee accepts the invitation and subscribes to Gmail, the member who invited her receive a notification of the friend's subscription, and invites her to be the first who send her a message.

Permission setting

Name

Permission setting.

Problem

A member needs to be in charge of setting permission for granting/forbidding the performance of some activities to a set of other members on items he published. The set of members represent a subset of the social relations she set during her activity within the virtual community.

Solution

To implement a mechanism that is capable of setting permissions related to items, activities and dynamic aggregations of users.

Related patterns

Permission check.

Discussion

This pattern, intended for setting permissions on items, goes beyond what is currently achieved by the WebML User-Group-Module pattern, because it relies on the representation of aggregative relationships, that describe the kind of relations that a user establishes during her activity within the virtual community. Such relationships, differently from groups, don't express a group definition, but an aggregation of individuals that have been socially related with the member in object. This allows members to set different permission on an item for friends, familiars and strangers, in respect to different activity definitions.

Examples

In the Flickr photo sharing community every activity that members are generally allowed to perform on other's photos is subject to permission settings that the owner of a photo sets for members anyhow related to her. 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.

Relationship setting

Name

Relationship setting.

Methodology and patterns for developing community-based web applications with a model-driven approach

Problem

Members establish social relationships within virtual communities.

Solution

Designing a mechanism representing the act of setting any relationships performed by a member towards another within the community.

Related patterns

Evaluation, Permission check, Reputation adjustment.

Discussion

In social networks members can form groups, set relationships like friendship, nearness, familiarity, ban other users blacklisting them and so on. Any member, regardless her group/role within the community, can establish relationships with others. The concept of *relationship setting* encompasses all connections that a member can set with other members of the community during her activity.

The number and the type of relationship settings a user is object of, can impact her reputation: positive relationships, like friendship, tend to increase it, while negative ones, like banning/blocking, reduce it.

Examples

- The act of adding somebody to a member's friend list on MySpace.
- The act of subscribing to a user's channel on YouTube.
- The act of banning a user who is bothering with an offensive activity on a member's items.

Social visualization

Name

Social visualization.

Problem

Members feel more likely to contributed if this can help them to improve their status within the community.

Solution

Design an effective social visualization mechanism.

Related patterns

Evaluation, Reward, Reputation adjustment.

Discussion

Social visualization can be an effective solution for fostering participation within a community. Visualizing levels of participations of all community members, users can be stimulated to engage in responsible and reciprocal behavior.

Social visualization can be related to many factors, like the number of contributions of a member, the amount and intensity of social relationships she established, quality of contributions or more comprehensive indexes, like reputation. This pattern could also be used to promote the purchase of non-free services among members.

Example

Comtella is a virtual community for sharing papers. Social visualization in Comtella is made according to four dimensions: size, color, brightness and shade. Every user is represented by a star in a night sky whose appearance is determined by a combined effect of reward and reputation.



Figure 16: Social visualization in Comtella

The Flickr community offers two different kind of membership: basic membership, free of charge, and the “pro” membership, which cost 24.95\$ per year. Social visualization is used to distinguish users type of memberships.

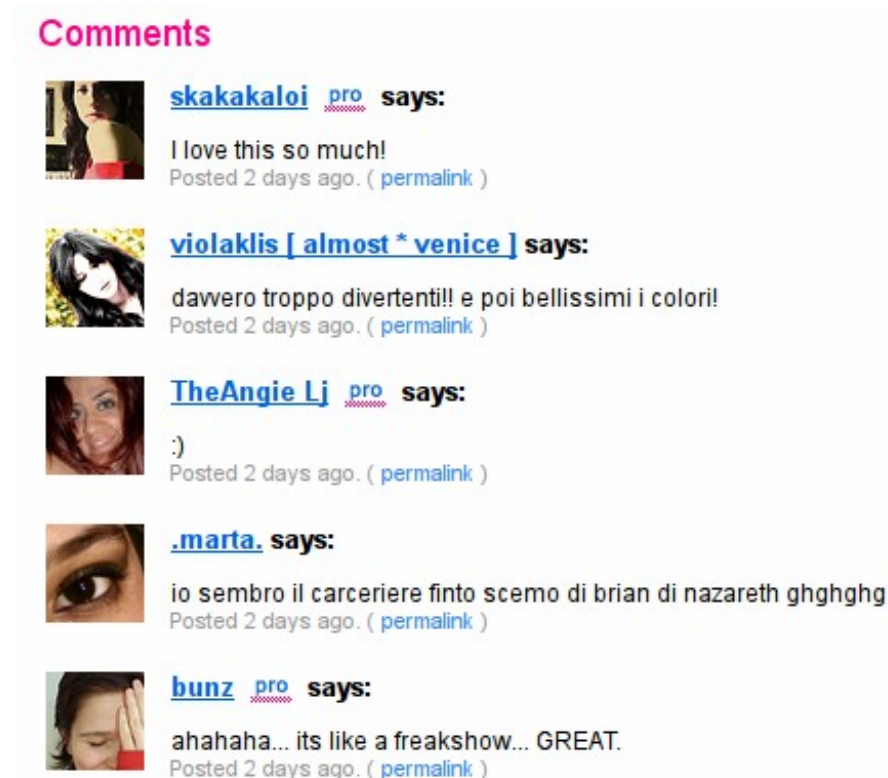


Figure 17: Social visualization for type of membership on Flickr

Talk

Name

Talk.

Problem

Members of a community need to communicate in order to establish social relationships and to develop a certain sense of belonging. To that end, communication between users should not be limited to contribution (like happens for comments), but should be intended regardless their activity on items.

Solution

Users should be able to exchange point-to-point messages. This in general does not add any value to the community, but letting users communicate beside their activity can help the construction of a sociability within the community itself.

Related patterns

Notification, Permission check.

Discussion

This pattern can be implemented allowing an asynchronous communication between members, implementing a built-in mailbox, or a synchronous exchange of instant messages. Communication is in general intended as point-to-point; multicast and broadcast patterns are in general more critical because they expose community members to the risk of undesired communications, so their usage should be limited to administrative groups of users.

A good practice when implementing this pattern, is to forward messages also to third party communication services, in order to perform a more effective communication, independent from the fruition of the web application.

Examples

In Google Docs, when working cooperatively on a document, members can exchange real-time text messages via a GTalk/Jabber client embedded into the web application interface.



Figure 18: Talk pattern though instant messages on Gmail

Back-end patterns

Back-end patterns are directly related to the way community reacts to any activity that members perform. They often can be seen as workflows, which regulates front-end activities, or as processes that are activated as the system needs to adapt to those state variations caused by members' activities.

Evaluation

Name

Evaluation.

Problem

Activities performed on somebody's items (like comments, ratings, flagging, etc) needs to impact on both the item relevance and the member's reputation.

Solution

To implement a back-end mechanism that considers activities performed on items as more or less explicit evaluations of the items' quality. Because in general, community members gets more authority as their reputation grows, evaluations should to be weighted on such reputation. All implicit/explicit evaluations must concur to the calculation of the item relevance, and also impact on the reputation of the owner of the item.

Related patterns

Relevance adjustment, Reputation adjustment.

Discussion

Evaluation is a crucial issue when dealing with socially-contributed contents.

Usually social networks use mechanisms that combine explicit (declared by members) and implicit (inferred by members' activity) evaluations to elicit the interestingness of

an item to the community and users reputation within the community. An *explicit evaluation* consist of a declaration, like a rating, by which a member arbitrarily quantify the quality of an item, usually picking it from a scale. An *implicit evaluation* is an evaluation inferred by an activity of a member on the item; for example the purchase of an item indicates, in a way, that the item has a relevance. In both cases evaluation of an item needs to be weighted on the reputation of the member who performed it.

Not all activity impact the same on the “interestingness” of an item. Viewing, for example, is a weaker indicator of quality compared to the addition of such item to somebody's list of favorites; we say that “viewing” and “adding to favorites” are implicit evaluations with different *intensity*.

Furthermore not every activity can be considered as a positive evaluation: activities like flagging a content like “offensive”, for example, could be considered as negative evaluations: because of this, they could negatively impact on the item's relevance as on publisher's reputation.

Notification

Name

Notification.

Problem

The system may need to notify members against the performance of some activities performed by others (for example being added as somebody's list of contacts), the appear of a situation (for example a low credit balance), etc.

Solution

This problem could be addressed designing a specific built-in messaging system within the application. An option is put such system side by side with a popular delivery system, letting notification being forward towards standard mailboxes and/or instant messaging system (like Jabber/GTalk).

Related patterns

Invitation, Suggestion, Relationship setting, Talk.

Discussion

According to the definition of virtual community, communication is fundamental for a community to exist. We think that, in a complex system, the system itself should be intended as part of the community, because is the main and very first entity the user interfaces with.

Examples

In the Flickr photo community, users gets notified within an internal mailbox every time other users set social relationships with them, subscribing to their photo stream.

Payment

Name

Payment.

Problem

Some activities within the community, those we called *controlled activities*, 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.

Solution

Any member should be endowed with a credit, that is subject to adjustment against the performance of every controlled activity. Such activities must have an explicit indication of the cost in their definition, meaning the amount of credit that the user needs to spend in order to perform it.

Related patterns

Reward.

Discussion

Credit and cost of an activity don't necessarily need to 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. Of course this outlay is strictly related to the modalities according to which the credit balance of a user is recharged.

Examples

Gmail offers to any user the possibility to invite friends to join the service through an invitation mechanism. The number of invitations available to a user is limited: every user gets a fixed number at the act of subscribing, then every invitation can be used only one time.

Permission check

Name

Permission check.

Problem

Not every activity available within the community can be performed by any member.

Solution

Design a permission check process, to verify the possibility to perform any activity against every member request.

Related patterns

Permission setting.

Discussion

Permissions can be set by users to restrict activities on some members they are in relation with (see *permission setting*). This evaluation of permission goes beyond what is currently achieved by the WebML User-Group-Module pattern, because within virtual communities people not only are part of group definitions, but also form group of individuals on which they refers when setting access policies to contents.

Relevance adjustment

Name

Relevance adjustment.

Problem

Some items within a community are more interesting than others.

Solution

Design a specific relevance criteria, aimed to infer the “interestingness” of an item from the activities that community members perform on it.

Related patterns

Evaluation, Reputation adjustment.

Discussion

Activities that members of a community perform on items can be seen as implicit evaluations of their quality and/or interestingness. We call this measure item *relevance*. Such measure impacts, by definition, on the reputation of the member who

respectively published them.

Relevance vary along with time: at the time t_0 , when the item j is first contributed, its value depends only on the publisher's reputation.

$$Relevance_j(t_0) = f(Reputation_{owner}(t_0))$$

Later, as community starts evaluating through members' activities, it depends on the combined effects of evaluations received, evaluators' reputations and, if subject to obsolescence, time.

$$Relevance_j(t) = f(Reputation_{owner}(t), Evaluation_j(t), t)$$

Reputation adjustment

Name

Reputation adjustment.

Problem

If community shows a particular trust toward an users, then means that his authority is recognized.

Solution

Define a reputation index and define mechanism to adjust reputation when necessary.

Related patterns

Evaluation.

Discussion

The definition of reputation a user is not a simple issue. First of all because reputation is time variant and changes over time. Secondly, because factors that impact on reputation are several, and all of them are also time variant. We can say that, in general, the reputation of the i -th user at the time t , depends somehow on two factors: her *sociability*, the extent to which other members within the community establish social relationships with her, and *feedback*, a measure of the quality that the community recognizes for items contributed by the i -th member.

$$Reputation_i(t) = f(Sociability_i(t), Feedback_i(t))$$

When a member of the community contribute with an items, the relevance of such items depends in general on the publisher's reputation.

Reward

Name

Reward.

Problem

Participation of members needs to be properly rewarded.

Solution

Design a reward mechanism.

Related patterns

Evaluation, Reputation adjustment.

Discussion

Successful communities are those in which members are happy to participate. By participation we mean activities which benefit the community and demonstrate involvement in the community, like contributing items, rating and commenting items contributed by others, logging into the system viewing items contributed by others. The design of a reward mechanism is aimed to foster this kind of participation.

By reward we generally intend a point reward that is paid to a member as she performs a constructive activity within the community. Such activity can give the member a certain amount of points, that are collected until they reach a threshold, when the reward is revealed.

A possible variant of this pattern is *Reward compensation* a reward mechanism that is not strictly connected to the performance of an activity, but which is given to a member depending on reputation.

Examples

The Yahoo! Answers application implements a point reward mechanism that rewards members at any login onto the web application.

Syndication

Name

Syndication.

Problem

Formal description of contents with syndications system and microformats has become

an issue of growing interest for improving the degree of interoperability of applications. Many different dialects and standards have been proposed and some of them are widely adopted in many context; they often consist of XML-based statements that describe contents at different levels of abstractions.

Solution

This pattern is used to provide a machine-readable description of items reflecting the logic organization of contents. Description can be implemented according to the Atom/RSS specification. This pattern should be actuated against every modification and/or reorganization of contents.

Related patterns

Publication, Organization.

Discussion

Achieving machine-readability of contents, interoperability towards external applications and agents is improved. This is currently a widespread practice with RSS/Atom summaries and podcast for direct content delivery.

Front-end patterns in social networks

We corroborated our front-end patterns also checking their suitability to describe services available across most popular social network applications. In the following table we report a brief summary. Back-end patterns have been excluded due to the objective impossibility to access such information.

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

Process

In this chapter we introduce the basics of model-driven development and the WebML process, used for modeling complex web sites at conceptual level. Then we propose an extension of the WebML process in order to model community-based web applications.

Model-Driven Development

Model-Driven Architecture

The Model-Driven Architecture (MDA) is a software design approach, proposed and promoted by the Object Management Group²¹, that increases the power of models in the development process. It is model-driven because it provides a mean for using models to understand, design, construct, deploy, operate, maintain and modify software systems.

MDA provides a set of guidelines for structuring specification expressed as models. Using the MDA methodology, system functionalities may first be defined as platform independent model (PIM²²). Given a Platform Definition Model (PDM) corresponding to the Web, the PIM may then be translated to one or more platform-specific models (PSMs) for the actual implementation. The translation from PIM to PSMs, are normally performed using automated tools, like model transformation tools like, for example, tools compliant to the OMG standard named QVT²³.

The MDA process is related to multiple standards, including Unified Modeling

-
- 21 Object Management Group (OMG) is a consortium, originally aimed to define standards for distributed object oriented systems, and now focused on modeling (programs, systems and business process) as well as model-based standards in some twenty vertical markets. Founded 1989 by eleven companies (including Hewlett-Packard Company, Apple Computer, American Airlines and Data General), OMG mobilized to create a cross-compatible distributed object standard. The goal was a common portable and interoperable object model with methods and data that work using types of development environments on all types of platforms.
- 22 A platform-independent model or PIM is a model of a software or business system that is independent from the specific technological platform used to implement it .
- 23 The overall process is described in a document produced and regularly maintained by the OMG and called the MDA Guide (OMG. MDA Guide Version 1.0.1, 2003).
-

Language (UML²⁴), Meta-Object Facility (MOF²⁵), XML Meta-data interchange (XMI²⁶), Enterprise Distributed Object Computing (EDOC²⁷) and the Software Process Engineering meta-Model (SPEM).

Note that the term “architecture” in a model-driven approach does not refer to the system components under the model will be developed but it rather to the architecture of the various standards and model forms that serve as the technology basis for MDA.

One of the main aims of the MDA is to separate design from system architecture and realization technologies; facilitating that, design and architecture can alter independently. The design addresses the functional (use cases) requirements while architecture provides the infrastructure through non-functional requirements like scalability, reliability, performance and security are realized.

MDA envisages that the platform independent model (PIM), which represents a conceptual design realizing the functional requirements, will survive the changes in realization technologies and software architectures.

Model-driven development process

The specification of the Model-Driven Engineering standard software process (MDD) has required an analysis of how the model driven engineering approach is applied to a software project.

The MDD approach does not imply a radical change in the software development

-
- 24 The Unified Modeling Language (UML) is a non-proprietary specification language for object modeling. UML is a general-purpose modeling language that includes a standard graphical notation used to create an abstract model of a system, referred to as a UML model. UML is extensible as it offers a profile mechanism for customization.
 - 25 The Meta-Object Facility (MOF) is an Object Management Group (OMG) standard for the Model Driven Engineering. The official reference page may be found at [OMG's Meta Object Facility](#). MOF originated in the Unified Modeling Language; the OMG was in need of a Meta-modeling architecture to define the UML.
 - 26 The XML Meta-data interchange (XMI) is an OMG standard for exchanging meta-data information via Extensible Markup Language (XML). It can be used for any meta-data whose meta-model can be expressed in a Meta-Object Facility (MOF). The most common use of XMI is as an interchange format for UML models, although it can also be used for serialization of models of other languages (meta-models).
 - 27 The UML profile for Enterprise Distributed Object Computing (EDOC) is a standard of the Object Management Group in support of an open distributed computing using model-driven architecture and Service-Oriented Architecture (SOA). The basis of EDOC is the “Enterprise Collaboration Architecture” meta-model that defines how roles interact within the communities in the performance of collaborative business process.

process, in fact the activities to perform stay almost the same. They still are requirements elicitation, software analysis, architecture and detailed design, implementation, testing and installation. The main differences respect the traditional software development approaches are the following:

- Models are the key elements of the software development and the means to express the results of the different activities of a software development project.
- The level of abstraction of software development is raised from the code level to the model level. Code is derived from well-defined and complete models.
- Separation of concerns is a fundamental principle to organize the levels of abstraction and structure of the software models to be produced. Thus, platform independent specifications are separated from the specific models (including code); functional requirements and their implementation are separated from non functional aspects, etc.
- Standards and formalism (usually expressed as meta-models) are extensively used for modeling the different software aspects. This allows models storage and manipulation: querying, providing views, transforming from standard source to a standard target model, etc.
- The relationships, traceability, derivation and transformation among different models (which represents different levels of abstraction or different concerns/views in a single level) are automatized as much as possible.

The MDD process is defined outlining the different phases as shown in the following figure. The eight phases are hereby described below:

1. *Capture user requirements.* The goal of this phase is to elicit, agree and document the customer requirements that the software system needs to fulfill. This includes establishing a common understanding with the customer on functional and non-functional requirements. This phase includes the following activities: formalization of the customer requirements in an Application Model, derivation of an initial Application PIM and of an initial functional requirements specification from the common infrastructure of reusable assets.
2. *PIM context definition.* The goal of this phase is to clearly define the scope of the software system to be developed. The result is an unambiguous definition of the system, its goals and scope following a black box approach. Main activities are: establishment of the system goals and business principles; description of the external actors that interact with the system and their key behavior; definition of the business event and exchanged business objects.

3. *PIM requirements specification.* The goal of this phase is to build a clear and complete model of the customer requirements and to have a unique requirements description that all subsequent models will use. In order to model the system functional and non-functional requirements, the main activities of this phase are: refinement of the PIM context; identification of services, events and business objects produced and consumed by the system and the actors interacting with it by specification of use cases, non functional requirements and atomic requirements; identification and modeling of the relationships between functional and non functional requirements.
4. *PIM analysis.* The goal of this phase is to model the internal view of the system without any technological consideration and maintaining the separation of concerns between functional and non-functional aspects. The main activities of this phase are: description of the system functionalities (the objects, the functions, the system boundary, the behavior, etc.); the description the system's QoS aspects and their application to the functional elements of the model; management and traceability with the PIM requirements.
5. *Design.* The goal of this phase is to model the detailed structure and behavior of the solution (software application) that fulfills the system functional and non functional requirements. This implies making decisions on how the system will be implemented and which architectural style, patterns, standards and platforms will be used. Following an MDA approach, the design is performed in two steps: specification and design of the application and design of the platform-specific solution by refining the platform-independent solution.
6. *Coding & integration.* The goal of this phase is to develop and verify the software code that implements the software design fulfilling the software requirements. This phase includes activities such as: developing the components and classes (according to the models used as inputs); defining the organization of the code, executing unit tests, and integrating components and subsystems. Following a MDA approach, the code is intended to be automatically produced from the PSM through transformation engines.
7. *Testing.* The goal of this phase is to demonstrate that the final software system satisfies its requirements. This phase includes activities such as: plan tests; prepare test model, test cases and test scripts; execute tests; correct defects and document the testing results. The models are traceable to PIM models (specially to PIM Requirements) and following the MDA approach, test models will be refined from the PIM and test cases and test scripts will be automatically produced from test model through transformation engines.

8. *Deployment*. The goal of this phase is to ensure a successful transition of the developed system to the final users (including resources, environment, schedule planning and execution). This phase includes activities such as: create a deployment model (derived from the PSM deployment model and adopted to the specific execution environment of the customer); create the product manuals; maintain records of the product that is being delivered to the client; provide the installation of the product in the client environment.

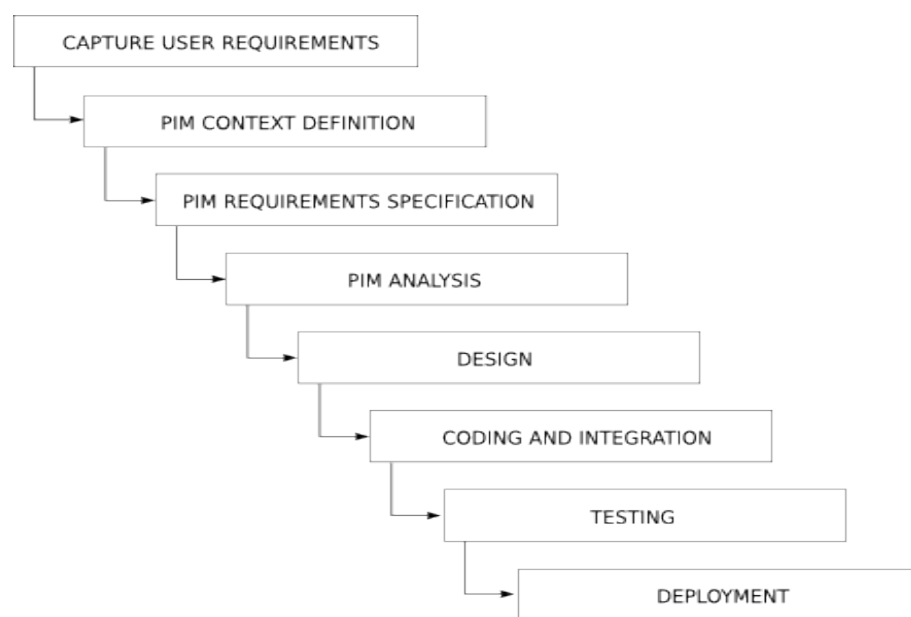


Figure 19: The MDD process

WebML process

The WebML process is an ad-hoc adaptation of the model-driven development specific for the production of Web application through the WebML^{xxxix} notation (an overview of such notation is provided in the next section).

The WebML process is built on appropriate notations and concepts for data and hypertext modeling, on the separation between the different aspects of the structure, navigation and presentation. Moreover, the activities in this process can be automatized using CASE (Computer Aided Software Engineering) tools.

The phases of the WebML process are described below. As it is possible to see, the lifecycle is completely adherent to the one proposed by the model-driven development process specification we introduced in the previous section.

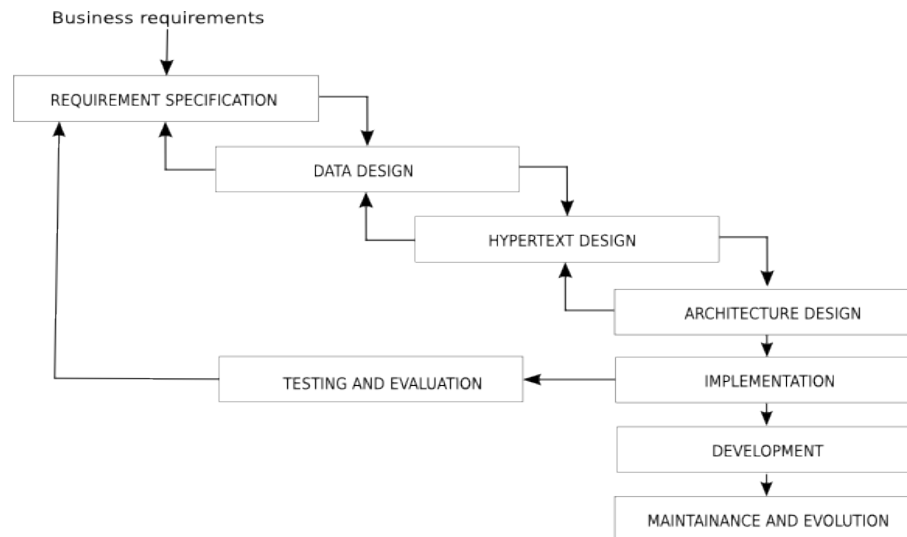


Figure 20: The WebML process

WebML

Web Modeling Language (WebML) is a notation for specifying complex Web site at the conceptual level. It addresses the high-level, platform independent specification of data intensive Web applications and targets Web sites that require such advanced features as the one-to-one personalization of the content and the multichannel availability of information.

WebML concepts are associated with an intuitive graphic representation, which can be easily supported by CASE tools and effectively communicated to non-technical members of the team.

Moreover WebML supports an XML syntax that can be fed to software generation for automatically producing the implementation of the Web site.

The specification of a site in WebML consists of four different perspectives (models):

Data model: It allows to represent in an ER based way the data content of the site. The designer has to specify entities and relationships and also, thanks to a simplified OQL²⁸ query language, some derived information.

²⁸ The Object Query Language (OQL) is a query language standard for object-oriented databases modeled after SQL. OQL was developed by the Object Data Management Group (ODMG). Because of its overall complexity no Vendor has ever fully implemented the complete OQL. OQL has influenced the design of some of the newer query languages like JDOQL and EJBQL, but they can't be considered as different flavors of OQL.

Hypertext model. It describes one or more hypertexts that can be published in the Web site. The main elements of the WebML hypertext model are site views, areas, pages, units, operations, links and session/application variables. A site view is a graph of pages, possibly grouped into areas, allowing users of a given group to perform their specific activities (e.g. users browse the information, while content managers update it). Pages contain content units connected by links, which represent atomic pieces of information to be published.

Presentation model. It expresses the layout and the graphic appearance of the output device and of the rendition language. It's possible to specify a presentation for a specific page or a generic one for all the application pages. This model allows to define different layouts for different site-views in order to make the user recognize quickly which part of the site she is interacting with.

Personalization model. It allows to share specific or individual content that can be used both in the composition of the units and in the definition of presentation specification.

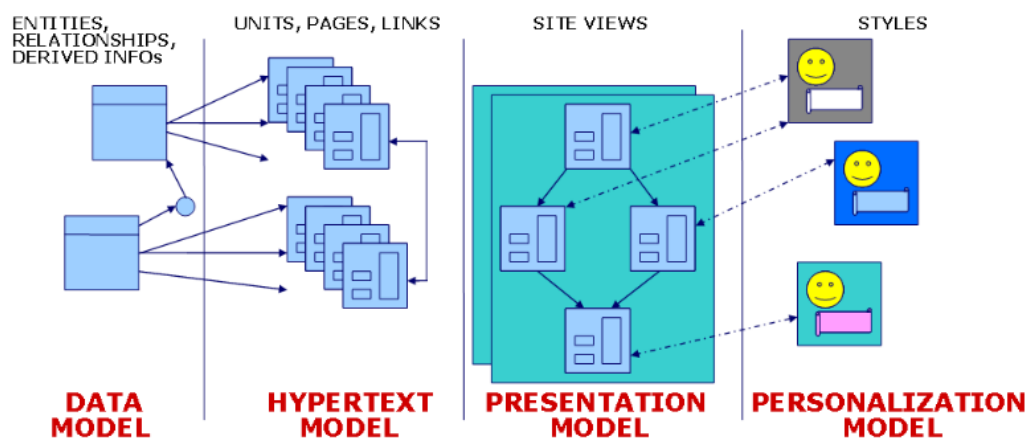


Figure 21: The four WebML perspectives

WebRatio

WebRatio is a CASE tool representative of the model driven design approach . This approach claims that more and more effort should be spent on the application modeling and reusable implementation should be automatically or semantically produced from high level models.

WebRatio supports the WebML design process producing the implementation of the

relational database and of the application page templates. For this reason its architecture includes a graphic interface and a code generator. The graphic interface helps editing ER and WebML schemas while the customizable code generator transforms ER specification into relational table definition for any JDBC/ODBC compliant data sources, and transforms WebML specifications into page templates for the Java2EE and .NET platforms.

WebRatio has five main features:

1. *Data design*. The developer can design the Entity-Relationship data schema with a graphical user interface that allows to draw and specify the properties of entities, relationships and attributes;
2. *Hypertext design*. The developer designs site views specifying the properties of areas, pages, units and links. This features also includes the web service primitives ;
3. *Data mapping*. This feature permits declaring a set of data sources to which the conceptual data schema has to be mapped and translated into relational databases.
4. *Presentation design*. It offers functionalities for defining the presentation style of the application, allowing to create personalized style sheets and associate them with pages.
5. *Code generator*. This features automatically translates the hypertext design into a running Web application built on top the platforms mentioned above.

The tool also supports other functionalities for automatically checking errors already at the design level automatically producing the project documentation, managing the collaborative work and the project version, doing direct and reverse database engineering.

The figure reported below summarizes the design flow of WebRatio and also represents its architecture. It consists of two layers:

- *Design layer*. It provides functions for the visual editing of specification, supporting the first four functionalities listed previously.
- *Runtime layer*. It implements the Model-View-Controller Web Application Framework (MVC2).

These layers are connected by the code generator, which exploits XML transformation to map the visual specification edited in the design layer into application code executable from the runtime layer.

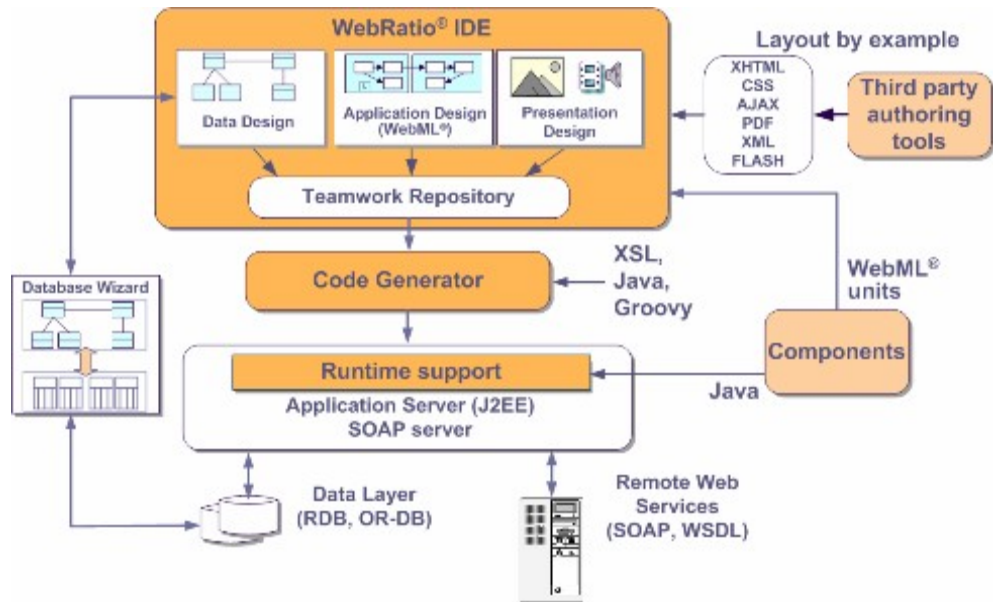


Figure 22: WebRatio architecture

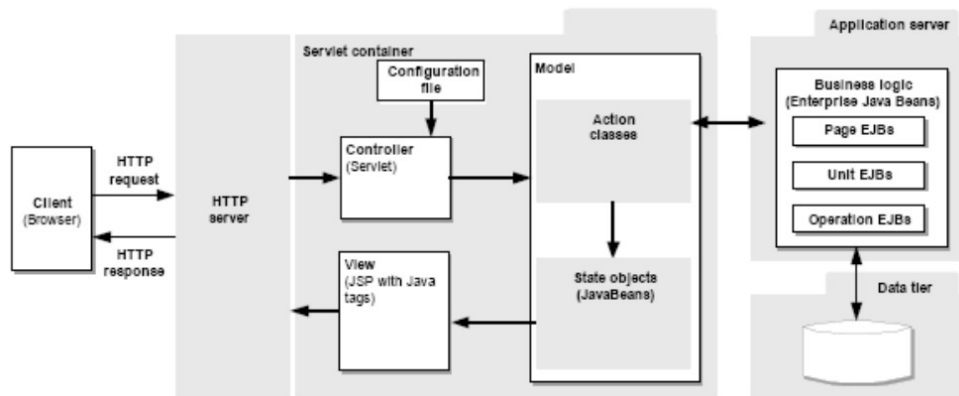


Figure 23: Mode-View-Controller pattern for Web applications (MVC2)

Extending the WebML process

Motivations

The WebML process is open to extensions, which means that projects lifecycle can be adjusted in order to best-fit specific context needs.

A clear example of this, is given in the work for *Process Modeling on Web Application*^{xl}, aimed to the specification and the automated development of web applications in workflow environments. This extension proposed a critic review of the WebML process, starting from the identification of new typologies of requirements, arriving to the definition of new phases of the WebML process itself.

The necessity to review the WebML process started from the intuition that some applications were not purely data-centric, as those for what the WebML process was originally designed. The need to model process-centered applications led to many changes in the way requirements were gathered, in the way they were elicited and expressed and also in how an application had to be modeled in term of data model and hypertexts.

It's not our purpose to go further in depth with this discussion; nonetheless we would like to underline how the WebML process can be specialized according to specific application domains.

Like it has been done for workflows, we propose a review of the WebML process, aimed to bring a model-driven development method to the design of complex, community-based applications.

In order to achieve this, we have studied in-depth the WebML process in order to

identify lacks of representativeness for crucial aspects of virtual communities, reviewing both functional and non-functional requirements and extending the development lifecycle. Our study has taken into considerations the specific literature of virtual communities, in order to achieve an aware definition of activities, roles, state objects and open issues for addressing effective requirements elicitation and modeling of the web application.

An overview of the WebML development process adaptation we made for the community centered applications is depicted below.

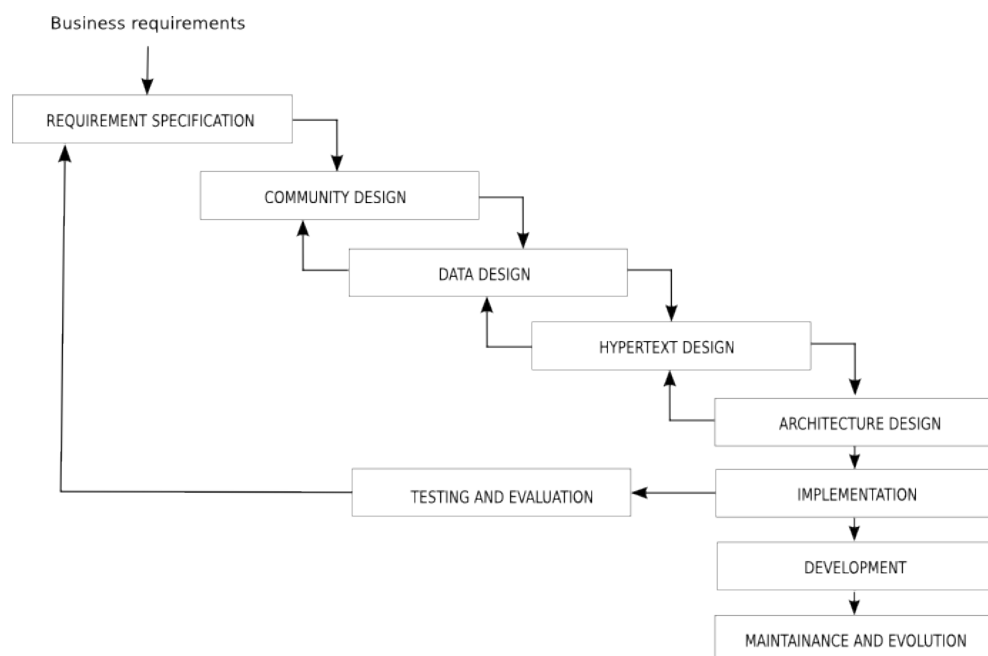


Figure 24: The extended WebML process

Community development lifecycle

Just like software, a virtual community has its own lifecycle too. From an engineering point of view, it is important not to confuse the community lifecycle with that of the software needed to support the community. This distinction may sound trivial, but is very important not to confuse the different levels of abstraction from which a designer needs to face for the developing of a successful virtual community environment in order not to underestimate peculiar aspects strictly related to a specific perspective of the problem. In facts, while the community development process output is the community, in the software development process the output is the application.

According to the Community-Centered Development (CCD^{xli}) process proposed by Preece, the development of a virtual community needs to take into consideration both aspects related to the community design (social processes, policies, etc.), and others which are purely related to the application design. Unfortunately, although Community-Centered Development has highlighted the necessity to merge those two design perspectives, it doesn't offer a development model to do that.

Observing the WebML process definition, we argued that it was suitable for importing the community-development perspective into the software lifecycle by adding a community design phase prior the data design.

Inputs and outputs of the process

The WebML process distinguishes inputs between *business* and *environmental requirements*. The definition of such requirements covers many aspects whose good elicitation is under the responsibility of the application analyst. With the advent of Web 2.0 applications and to face the increasing relevance of virtual communities, the business requirements definition needs to be reviewed, in order to understand the link between a company's business goals and its capability of building up a community around itself.

Many business requirements need to be related to the community design. The point is that such requirements need a interdisciplinary awareness in order to fully-comprehend the complexity that comes from the multiplicity of the involved scientific matters, from social network analysis to HCI and social/behavioral sciences. This may differ from the classical background that a software analyst usually has.

Before start analyzing the business requirements in order to produce software specifications, it is necessary to understand how to design the virtual community that will be hosted by the application. To do that the analyst has to interview the customers to clarify how to translate business requirements into *community requirements* and then to understand how to translate them into software specifications.

Such intermediate step allows to specify social processes, social state objects, policies and roles according to which the community has to be set up for achieving the business requirements.

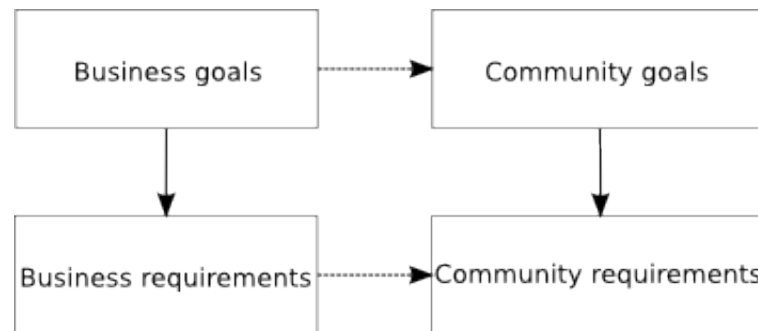


Figure 25: Goals and requirements

Community requirements analysis needs to focus on the following aspects:

1. User behavior monitoring (reputation and trust systems)
2. User content management and navigation (tagging, bookmarks, catalog systems)
3. Communication between users (media, policies)
4. Activities and encouraging/moderating strategies (participation fostering systems)
5. Community state/achievement monitoring (lurker identification)
6. Recommendation systems (social recommendations, ad-hoc contents and services proposal)

Development roles

The development of a community-based application involves the same professionals described in the WebML process. In addition to this, we outline the necessity to open the community design to the involvement of two more professionals: *communication expert* and *social networks expert*.

Communication expert

This expert plays a relevant role in order to design a successful community; her work is to understand the business goals and environment constraints for identifying the

right *social processes* aimed to foster user participation and to achieve community commitment towards business goals. In addition to this, she has to individuate the proper front-end patterns to be used to turn social processes into hypertext patterns.

Identifying social processes means to understand which user activities are critical for the community sustainment and then, to identify the strategies according to which to foster/monitor such activities with opportune reward and reputation systems.

Furthermore, the communication expert has to *plan the sociability* in order to identify policies to manage and measure members reputation, codes of conduct, moderating solutions and member contribution quality.

Social network expert

A social network expert should design the community recommendation systems, matching business requirements and non-functional requirements into algorithms for mining the community behavior.

This expert studies the community under a network analytics perspective. Her scope is to define network models representing the members relationships within the community and indexes for monitoring them.

Usually, the network expert studies the business requirements in order to understand if there is the necessity to propose contents or services based on the implicit analysis of the member behavior and taste.

Social networks are very powerful for the engagement of community members, because they can be exploited to propose very personalized and attractive contents and services. Furthermore they allow community administrator to be in condition to trace the community dynamics through data mining techniques.

Requirements specification

During this phase the analyst has to collect requirements in order to provide a full overview of the main actors, scenarios and constraints of the application context. Just like in the traditional development process, requirements can be elicited through

software engineering tools provided by UML standards and no additional notation is needed.

What changes from the classical development process is the perspective according to which it is necessary to analyze and recognize information related to the application context. More precisely, in addition to the traditional business and environment requirements, the analyst needs to collect information in order to understand how to define the *community governance*, *social presence* and *communication modalities*.

According to this, we hereby review how to collect and represent application requirements.

Identification of users

What differs from the classical user hierarchical definition is the scope according to which the hierarchy of users has to be compiled. In order to define it correctly, we suggest to define them at three different levels: *content management*, *governance* and *social relationships*.

- *Content management level*. At this level is necessary to distinguish between internal and external users of the community, identifying application stakeholders, administrative roles, editors, etc.

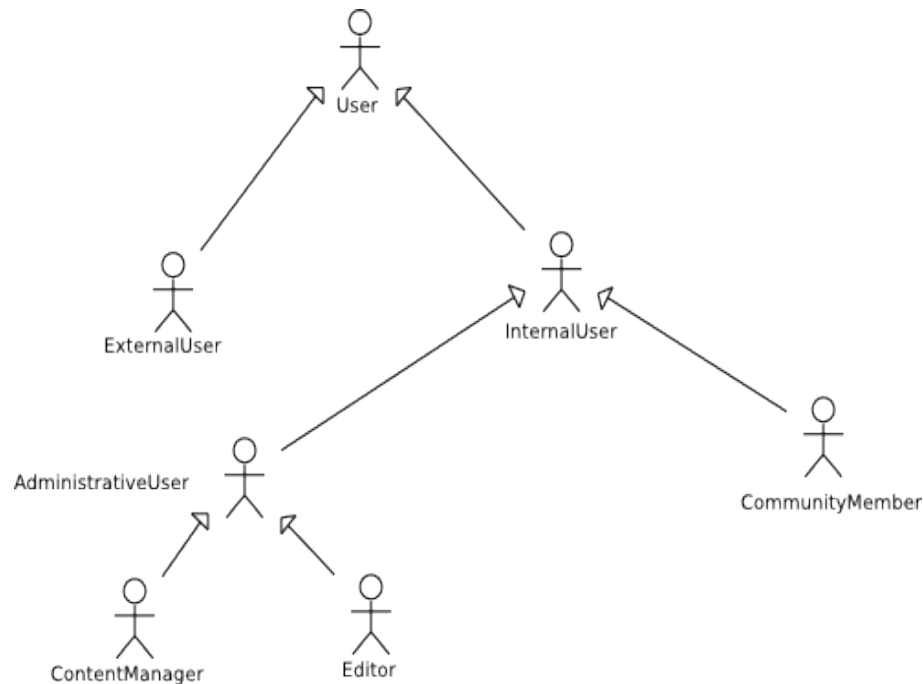


Figure 26: User hierarchy at content management level

- *Governance level.* At this level the analyst has to interpret how to map the social hierarchy according to the way in which the community needs to be structured, in order to find social roles within the community.

The question that we suggest to address at this level is: “who is controlled by who?”. There are many ways to define a governance model: one of the most widespread is currently a *democratic moderation* approach. It basically relies on a shared, accepted *code of conduct*: every member within the community is entitled to monitor others and draw attention on violations of the shared norms. In such system the governance hierarchy is simple; it consists only of two categories of users: moderators and contributors.

For more complex contexts, different governance model can be adopted: for instance a community built for employees of a certain company could be configured according to a governance model that reflects the company internal coordination.

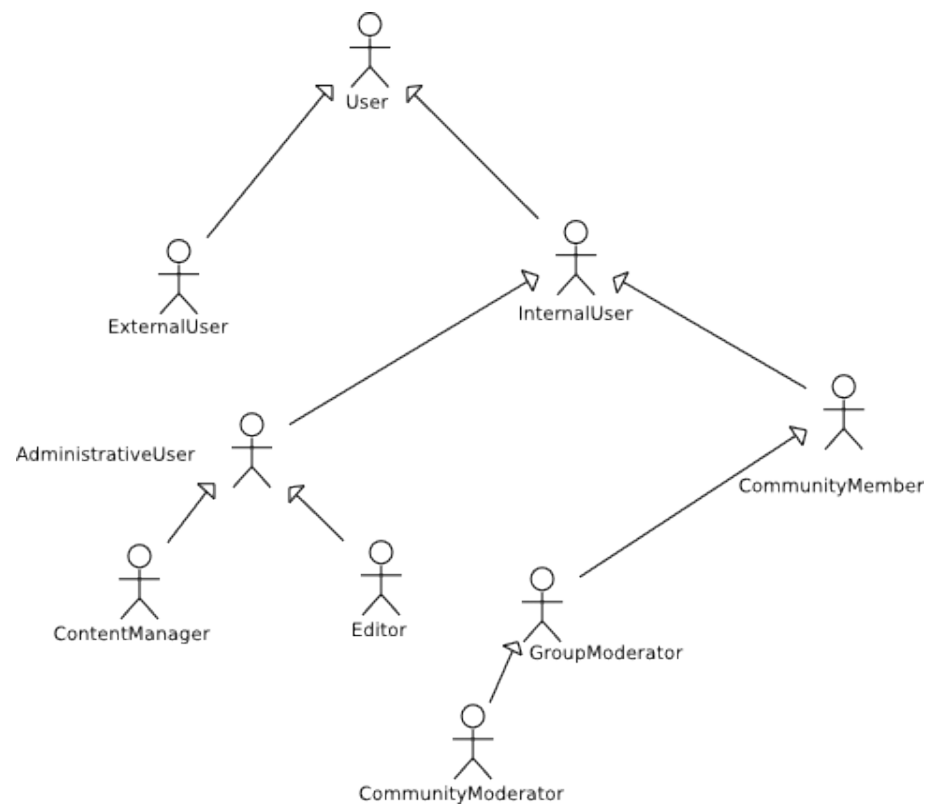


Figure 27: User hierarchy at governance level

- *Social relationship level.* Within real-life communities people establish relationships in response to particular needs, like the need to solve specific tasks, discuss a topic or simply “stay in touch”. When identifying the hierarchy of users of a virtual community, the analyst needs to take into consideration the nature of relationships that members will be allowed to establish, as the borders of collaborative interactions among them.

Once roles of users are identified at each level, these information can be represented through a hierarchical view of user types, that could easily associated to the definitions of activities that they are entitled to perform, according to the WebML User-Group-Module pattern.

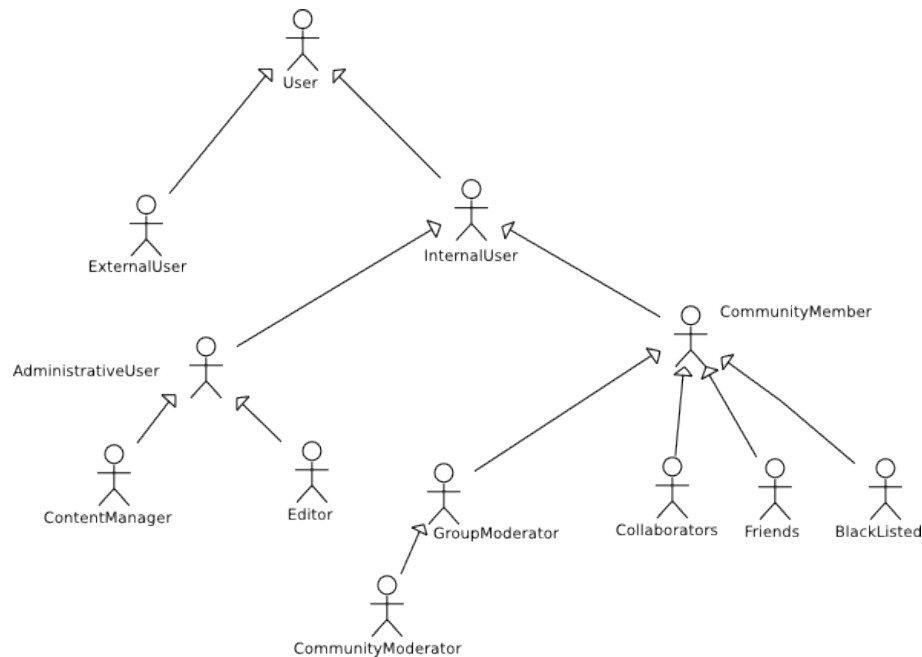


Figure 28: User hierarchy at social relationship level

When this hierarchy is outlined, the user typology description is still not complete, because some aspects related to the real behavior of users within community aren't considered. Users, in facts, are not only distinguished by the role they are assigned to, but also from their participation within the community; it's possible, then, to outline behaviors such as *pioneers*, *killers*, *lurkers* to help stakeholders to monitor community and plan further requirements.

Relying on these definitions, the analyst should discuss and eventually propose indexes and criteria for the run-time classification of users under their behavioral profile.

Functional requirements

Like for the standard WebML process specification, functional requirements collection is aimed to define processes and sets of activities that the application needs to support.

The pure identification of such processes and activities is not enough for a community-centered application design: in addition to these, a deeper analysis is needed to infer back-end processes in order to provide those services that are needed to *monitor*, *drive*

and *foster* the community growth, participation and achievement.

The key question should be: “Which processes and activities do we need to offer members for the community to become successful? How do we think to foster users' participation?”. Furthermore policies need to be set for fostering users participation, criteria for measuring members degree of achievement and trust, and also moderation strategies.

Usually these important requirements aren't explicitly defined by stakeholders and could be obscure, especially during the first phases of development. Anyway they require an ad-hoc analysis by experts with good knowledge in the field of communication.

During our research we identified the following aspects regarding typical functional requirements:

- Member reputation management
- Contribution rewarding strategy
- Evaluation of contributions and relevance criteria
- Moderation of contributions
- Contents/services proposition strategy

Guidelines for functional requirements

In order to effectively gather functional requirements, we suggest to proceed as follows. First of all, collect all processes and activities that need to be implemented within the virtual community. Then, for each process/activity, define accurate uses cases, trying to outline, for each use case, the following characteristics:

- *Performer*: who is in charge to perform the activity and is responsible for its result.
- *Moderator*: who is in charge to monitor the activity and is entitled to compensate abuses and/or mistakes occurred while performing it.
- *Involved entities*: which business objects are involved in the process.
- *Reinforcement*: does the activity have an incidence on the achievement of business/community goals?
- *Fostering strategy*: which strategy will be adopted to encourage people to perform the activity; it can refers to reward mechanism or to other fostering strategies.

- *Evaluation methodology*: the methodology according to which the community will evaluate the quality of the activity or the result of that activity. It also includes the definition of a rating scale.
- *Interoperability needs*: define whether the activity needs to be provided as a service for other platforms and/or if it has to exploit existing services deployed on different platforms (we suggest to plan interoperability for core activities within the virtual community).
- *Patterns of use*. if it is possible try to associate the activity to an existing front-end pattern.

Personalization requirements

The necessity to implement a virtual community imposes to spend a consistent effort towards the description of members' profile, social visualization and for designing personalized contents and services.

Before the advent of Web 2.0, personalization techniques based on user profiling characteristics were already been introduced for achieving one-to-one marketing strategies through mining of activity data within B2C portals. Now, within virtual communities, member profiling and personalization assume an even more important relevance for the Web application.

Member profiles assume a very important relevance in order to describe the social presence of people within the virtual community. Just like it happens for real communities, people need to express themselves through an identity and need to be aware of roles and social positions of other members. So the question is: “What distinguishes a member from another one?”.

All these aspects need to be taken into consideration according to the specific context in which the community is going to be put; for example, in a technical community, members may distinguish according to their technological skills.

Anyway, independently from the context, a typical profile property that describes members is the *reputation*.

Non-functional requirements

In addition to classical non-functional requirements (usability, performance, availability, scalability, security and maintainability) we introduce the *interoperability* requirement.

For interoperability, we intend the capability of a community platform to communicate and exchange information with other external applications. It is based on the service oriented architecture (SOA²⁹) which includes a pervasive use of Web services.

Without going in-depth with description of the SOA paradigm, we observed that online communities interpret interoperability mainly in two ways: providing new specialized services and concentrating existing ones. A clear example of the first case communities is provided by *Del.icio.us*, an online tagging systems whose services can be embedded by third parties without any particular technological effort. A clear example of the latter case is represented by *iGoogle*, which allows Google members to configure a personal mash-up by including functionalities provided by third party applications.

Considering this, the application architect should design the application adhering to the SOA paradigm and standards, in order to provide the infrastructure needed to integrate with existing services.

Community design

One of the main critical issues related to the community development is the definition of system according to which it is possible to control the community behavior in order to guarantee its sustainment.

Member reputation and contribution relevance are two of the main simple aspects that determine the community sustainment.

Like in the real community, usually people want to be appreciated by the others, the same happens for the virtual community where members are judged by their behavior and quality of their contribution.

Such phenomena can be used in order to moderate and foster member participations by giving them a reputation weighted on the relevance of their contributions or activities performed into the community environment: for instance, a member good action will be rewarded by an increase of the performer reputation, contrary her

29 Service Oriented Architecture (SOA)

reputation will be decreased if such action goes against the community sustainment.

Reputation and relevance are two of the main aspects which can be exploited in order to control the community behavior, many others can be considered for peculiar contexts: for instance, in a virtual community for the development of open source software, members would like to distinguish themselves by exposing the number of bugs they discovered. In this case such aspect could be taken as a relevant variable to be considered in order to design an effective participation fostering system.

In order to design a moderating-fostering participation system such elements are to be defined:

- Which member activities impact to the community
- How to reward members for their good contributions
- How to measure the contribution goodness

The steps to accomplish this task are:

1. To build the model which represent the member reputation, contribution relevance or other significant quantity aspects which measure the member participation whit appropriate *state variables*.
2. To design the possible workflows according to which it is possible to reward the members and adjust the involved state variables

While the design of state variables, allows to define the quantitative indexes according to which it is possible to represent aspects which infer the community behavior, workflows can be used to define the combination of activities have to be performed in order to update such indexes.

Community state variable design

Community state variables

A community can be viewed as a complex dynamic system whose state depends both on the activities performed by the members and the environment in which it exists. Such representation of the community allows application designers to represent elements that affect the community behavior turning them into state variables.

We call a *community state variable* a peculiar property of an entity whose value changes every time a specific event occurs.

Contribution relevance and user reputation are two common examples of state variables: every time a member perform an activity on a contribution she causes a change of them, and this change affects the whole community.

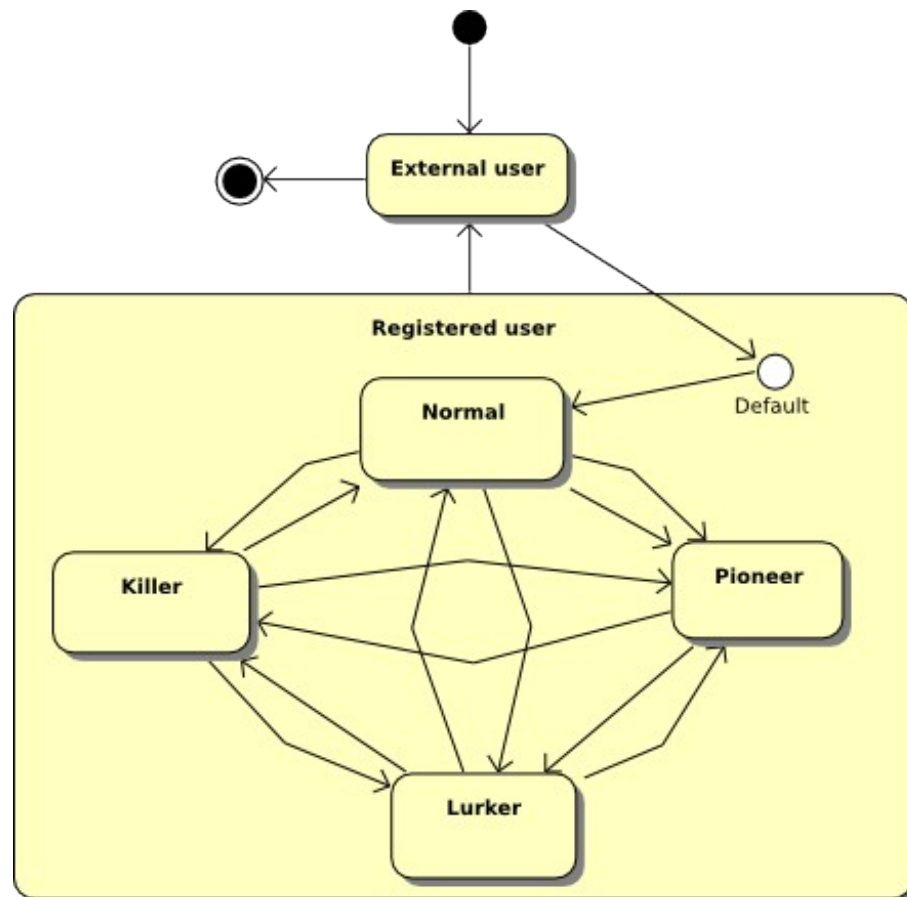


Figure 29: An example of state variable describing the participation of a member within the community

Influences

In some contexts, the changing of state of a particular variable influences the state of other variables. Let's consider, for instance, the reputation of a member: it is intuitive to assert that its value is influenced by the overall evaluations of her contributions performed by other community members.

In this case we can represent the member reputation as the state variable "*reputation*" and we can define the state variable "*contribution relevance*" in order to describe the average of the community member evaluations onto such contribution. In such

example it is intuitive to understand that the “*reputation*” state variable of a member is indirectly influenced by the actual value of the “*contribution relevance*” state variable of every contribution performed by such member.

In order to represent all the state variable influences, a simple oriented graph can be used: each node of the graph represents a state variable, while each arrow represents an influence.

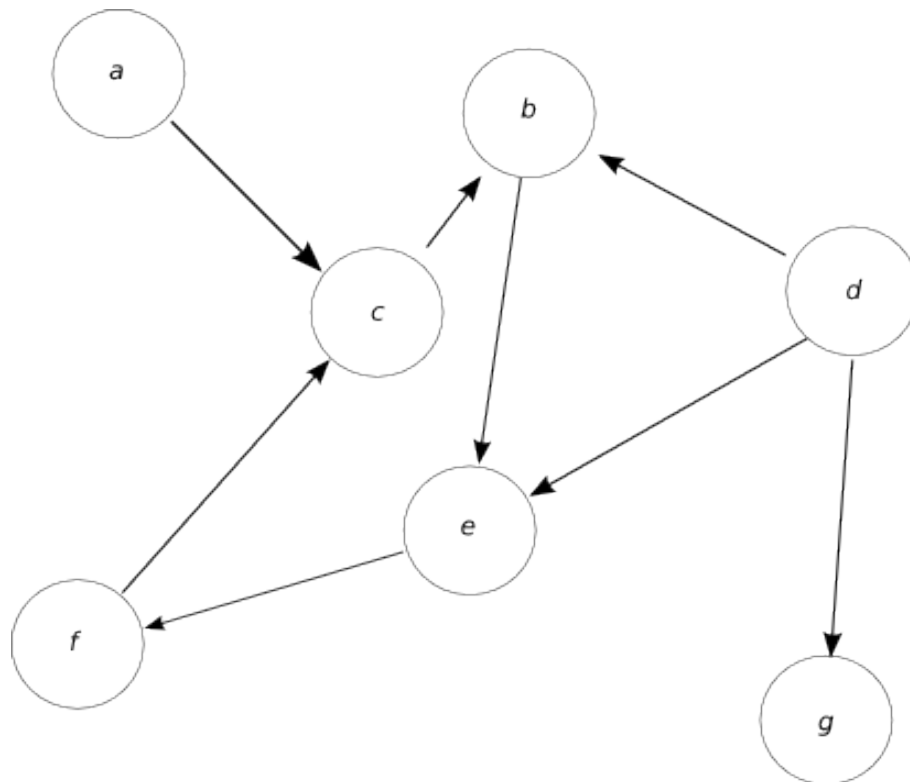


Figure 30: Representation of influences between state variables using a graph

The use of a graph can help to find non-evident relationships between state variables, anyway the choice of the level of approximation according to which kit is possible to represent such relationships sensibly impacts on the overall system complexity. It's important not to exceed with details, but rather focus the attention only on relevant aspects to be represented.

Time dependency

Another aspect that can affect state variables is *time*. In order to explain this aspect we can take into consideration a simple case, for instance the relevance of a piece of news published. As mentioned before relevance of a piece of news could be estimated on the basis of the evaluations given by the readers according to a proper rating scale.

Anyway, such definition doesn't care about the fact that a piece of news loses its relevance as it becomes aged.

For such variables it is necessary to define a temporal range according to which the state variable value is adjusted, depending on its age. The temporal range and the function that represent the state variable evolution have to be defined in order to respect the nature of such variable: for instance the time range according to which the relevance of a capital market news is estimated is completely different from that of a post in a online cooking blog.

Summary

In order to summarize the main characteristics of a community state variable we list the main properties that the analyst needs to take into consideration:

- dimension and scale definition;
- groups of users involved;
- events that cause the variable updating;
- the function by which the variable update is calculated;
- possible state transitions;
- influences on other state variables.

Community state variable taxonomy

In order to help the designer to identify easily the characteristics of state variables we propose the following taxonomy:

- *Interdependency*. We call independent those state variables whose state transition isn't influenced by any other state variable transition. Those that don't match such definition are classified as dependent variables.
- *Time variance*. Time variant state variables are those variables whose effective value is directly influenced by time; the relevance of a piece of news is a typical example of this kind of variable. Otherwise, state variables are time invariant: for instance, the counter of reviews of a specif article doesn't change its intrinsic meaning along with time.

- *Automatic vs. manual control.* For automatic state variables, the state transition automatically occurs every time the solicitation event happens. A typical example of such class of variables is the relevance of a contribution: every time someone generates an “evaluation” event on the contribution, the relevance of such item can be automatically adjusted according to a specific function. Otherwise, if the variable state transition is ruled by a complex decisional process in which human agents are involved, such state variables are classified as manually controlled variables.

Mapping state variable transitions using active rules

In order to map the state variable behavior, we propose to exploit the concept of *active rule* introduced by the *Event-Condition-Action (ECA)* paradigm^{xlii}.

Active rules are defined using the following notation:

```
rule [ rule-name ]  
      when [ events ]  
      if [ condition ]  
      then [ action ]
```

The *events* part contains a list of events according to which the rule is activated, such events are restricted to temporal events, insertions, deletions, qualified updates (on specific attributes).

The *condition* part contains a boolean expression according to which it is possible to define particular predicates in order to startup/block the rule activation when the triggering events occur.

The *action part*, contains a sequence of commands that have to be performed when the rule is activated.

Like active rules, a community state variable transition is solicited, under specific *conditions*, by predefined *events* and it is performed by a specific *action*. This conceptual similarity is the joining point between the abstract view of the community systems and the technological realization of it.

In facts, active rules can be implement as *triggers* in ADBMS^{xliii}, and they are used for process enactment in WFMS^{xliv}. The use of them introduces many advantages in the application design because it permits to exploit already affirmed concepts without compel the application designer to devise new technological solutions. At this level of the development process, it is not necessary to focus on how to implement physically the active rules, but it is important to map each state variable according to the ECA notation. Such description, will be used in order to define the active rules implementation during the next phases of the development process, in facts it will be used in order to define the needed meta-data to be added on the data design phase and for the correct definition of the hypertext patterns.

In order to adapt the ECA notation for our purpose we propose to extend it which the following notation:

rule [*rule-name*] **controllers** [*moderators*]
when [*events*]
if [*condition*]
then [*action*]

The *moderators* part represents the list of users which can control the state variable transition, this field was introduced in order to represent such state variables whose state update is not automatic, but it's rather defined by a decisional process which requires the human intervention. Following the taxonomy definitions given in the previous section, all the *manually controlled* state variables, have to be represented with the list of *moderators* that concourse to the variable update process.

The *events* and *condition* parts don't change their meaning; the first represents the list of events which trigger the rule activation, while the latter represents the boolean expression that needs to be verified in order to enact the rule.

The *action* part needs to be described a little more in-depth; at this level of abstraction an action can assume two levels of granularity: it could be a *simple action* or a *complex action*.

A *simple action* is an action according to which the value of the state variable is updated automatically by a simple computation when the rule is enacted. Such actions are described by a specific *function* which is invoked when the simple action is performed; according to this, the value of the state variable can be updated by a specific calculation algorithm.

A *complex action* is an action according to which the value of the state variable is

determined by the result of a specific evaluation process, to which one or more moderators take part. In such scenario, when the active rule is enacted, the *complex action* starts the execution of a specific workflow.

Workflows can be described through many different notations; we propose to adopt the BPML, since a WebML extension for such notation is already available^{xlv}.

Example 1

By this example we want to show how to define an active rule for an automatic controlled state variable transition.

The case

In a virtual community for music lovers, people can publish and rate songs. The community stakeholders would like to incentive people in publishing good music and in inviting others to join the community.

In order to do that, the proposal of the marketing and communication experts is to define a system according to which the *reputation* of members is based on the quality of their productions and on their capability to introduce new people to the community. Furthermore, in order to weight the quality of a production, a relevance index can be used. The *relevance* of a production has to be estimated calculating the average of the members evaluations, weighted on the reputation of each evaluators. The evaluation of a contribution can range from one to five stars.

In order to foster users to invite eager and smart people, the marketing expert would like to recompense members that invite such users. To do that, the idea is to weight the reputation of the members whit the reputation of the members who they have invited. They call this property *sociability* of a member. Sociability is measured by points, members whose points are greater then 500 points are considered leaders.

Community stakeholders want to reward people with good reputation, if the reputation of a user exceed the 2000 points, then the user automatically becomes a power user and is allowed to download a full album for free per month.

Describing state variable

This simple case is an example of how to design a fostering-moderating participation strategy; unlike the simplicity of the strategy, the design of the system hides many

complex aspects, that need to be considered.

According to this case, state variables to be considered are the *member reputation*, *member sociability* and *contribution relevance*. These three variables are mutually dependent as follows:

- The member reputation changes as the relevance of her contribution is updated.
- At the same time the evaluation given to a contribution is dependent to the reputation of the users who performed the evaluation, so, indirectly, the member's reputation impacts on the relevance of the contribution.
- The member reputation depends also on her sociability, which depends on the reputations of people she invited to join.

An overview of such dependencies is shown in the following figure.

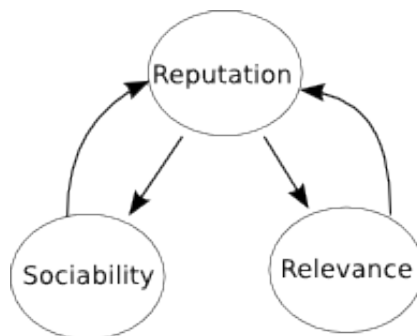


Figure 31: State variable dependencies

Once dependencies have been established, it is necessary to describe each state variable.

Member reputation

- *Description*. It describes the reputation of the member, based on her sociability and on the overall relevance of her contributions.
- *Measuring scale*. Points (≤ 2000 : normal users, > 2000 : power users).
- *Groups involved*. Member.
- *Triggering events*. The relevance of a contribution is updated. The reputation of a member who was invited by the member is updated.
- *Action*. Recalculate member reputation.

- *State transitions.*

Reputation	<i>normal user</i>	<i>power user</i>
<i>normal user</i>	-	>2000 points
<i>power user</i>	<=2000 points	-

- *Influences.* It can affect on the member *sociability* of the other members.

Contribution relevance

- *Description.* It describes the relevance of a contribution based on the members evaluations.
- *Measuring scale.* Stars (0: very low; 1: low; 2: mid-low; 3: middle; 4: high; 5: very high)
- *Groups involved.* Member.
- *Triggering events.* A member evaluates a contribution.
- *Action.* Recalculate contribution relevance.
- *State transitions.*

Relevance (stars)	<i>very low</i>	<i>low</i>	<i>mid-low</i>	<i>middle</i>	<i>high</i>	<i>very high</i>
<i>very low (0)</i>	-	+1	+2	+3	+4	+5
<i>low (1)</i>	-1	-	+1	+2	+3	+4
<i>mid-low (2)</i>	-2	-1	-	+1	+2	+3
<i>middle (3)</i>	-3	-2	-1	-	+1	+2
<i>high (4)</i>	-4	-3	-2	-1	-	+1
<i>very-high (5)</i>	-5	-4	-3	-2	-1	-

- *Influences.* It can affect on the member *sociability* of the other members.

Member sociability

- *Description.* It describes the capability of a member to invite high quality

members.

- *Measuring scale*. Points (<500:= passive user; >=500:=leader)
- *Groups involved*. Member.
- *Triggering events*. The reputation of an invited member is updated.
- *Action*. Update the sociability of the member.
- *State transitions*.

Sociability	<i>passive user</i>	<i>leader</i>
<i>passive user</i>	-	>=500 points
<i>leader</i>	<500 points	-

- *Influences*. It can affect on the member *sociability* of the other members.

Converting state variables to active rules

In order to map the member reputation transition we define the *member reputation-update* rule:

```
rule [ member reputation-update ] controllers [ none ]  
    when [ member.contribution is updated; member.sociability is updated ]  
    if [ always true ]  
    then [ updateReputation (member) ]
```

In order to map the contribution relevance transition we define the *contribution relevance-update* rule:

```
rule [ contribution relevance-update ] controllers [ none ]  
    when [ member evaluate contribution ]  
    if [ member != contribution.owner ]  
    then [ updateRelevance(contribution, member.reputation) ]
```

In order to map the member sociability transition we define the *member sociability-update* rule:

```
rule [ member sociability-update ] controllers [ none ]  
    when [ member.reputation is updated ]  
    if [ always true ]
```

then [*updateSociability(member)*]

Design the action

All the three rules have a *simple action*, so it is necessary to define algorithms according to which the state variable values are updated. The three functions are respectively: *updateReputation*, *updateRelevance* and *updateSociability*.

In order to establish the algorithms according to which these actions are performed, it is necessary to give a pseudo-mathematical definition of reputation, relevance and sociability.

The reputation of a member i at the instant t is given by:

$$Reputation_i(t) = f(Sociability_i(t), Feedback_i(t))$$

The function f represents a generic function according to which the *reputation* of the i member is adjusted, taking in account the value of the *sociability* of such member and the the *feedback* of the community on all the contributions of such member at the instant t .

Taking $Z_i(t-1)$ as the set of all the z members that have been invited by the member i before the instant t the *sociability* of member i is computed as the average of the reputations of members in $Z_i(t-1)$:

$$Sociability_i(t) = Avg(Reputation_z(t-1) \forall z \in Z_i)$$

Taking $W_i(t)$ the set of the all o contributions which owner is member i at the instant t and $\Gamma_o(t-1)$ the set of all the evaluations given to the contribution o before the instant t the feedback of member i at instant t is given by the generic function g :

$$Feedback_i(t) = g(Relevance_o(t-1) \forall o \in W_i, \|\Gamma_o(t-1)\|)$$

Where the relevance of a contribution o at the instant t is given by the following definition:

$$Relevance_o(t) = h(Relevance_o(t-1), Evaluation_o(t))$$

The function h is a generic function that adjusts the relevance of the contribution o taking into account the relevance of the contribution o before the instant t and the average of the all evaluations performed on o at such instant t . It is represented by the following definition:

$$Evaluation_o(t) = Avg(e_{o,j}(t) \forall j \in \Gamma_o(t))$$

Where $e_{o,j}(t)$ is the evaluation that the member j gives for the contribution o at the instant t , weighted on the her reputation until the instant t :

$$e_{o,j}(t) = Reputation_j(t-1) \cdot Intensity_{o,j}(t)$$

Intensity represents the effective value of the evaluation that the member j performed on contribution o at the instant t .

Example 2

By this easy example we want to show how to define an active rule of a manually controlled state variable transition.

The case

In a virtual community for sharing pictures, it is necessary to implement a system according with users can signal to the community moderator the presence of offensive pictures.

Every picture has a flag, according to which is possible to signal if it is offensive. When an a picture is marked as offensive a inquiry process is enacted: the moderator has to check the actual offensiveness of such picture. If the picture is actually recognized offensive, then the picture owner's state is updated from *free* to *observed*. If the picture owner was already in the *observed* state, she is banned from the community and all her pictures are to be hidden to the community.

Describing state variable

Member status

- *Description*. It describes the status of the users: free, observed, banned
- *Measuring scale*. Number of signaling; (0: free, 1: observed, 2: banned)
- *Groups involved*. member (picture owner), moderator
- *Triggering events*. A member flags a picture as offensive.
- *Action*. The moderator checks the signaling notification and adjusts the status of the picture owner.
- *State transitions*.

Status	<i>free</i>	<i>observed</i>	<i>banned</i>
<i>free</i>	-	Yes	No
<i>observed</i>	Yes	-	Yes
<i>banned</i>	No	Yes	-

- *Influences*. It can affect the visibility status of the banned user's pictures.

Converting state variable into an active rule

rule [*status-update*] **controllers** [*moderator*]

when [*picture flagged*]

if [*picture.flag == true*]

then [*new Instance(InquiryProcess, picture, picture.owner)*]

Design the action

In this case the action correspond to a “*complex action*”, so it is necessary to design the workflow of the *inquiry process* that has to be enacted.

A proposed BPML schema for the action is depicted below.

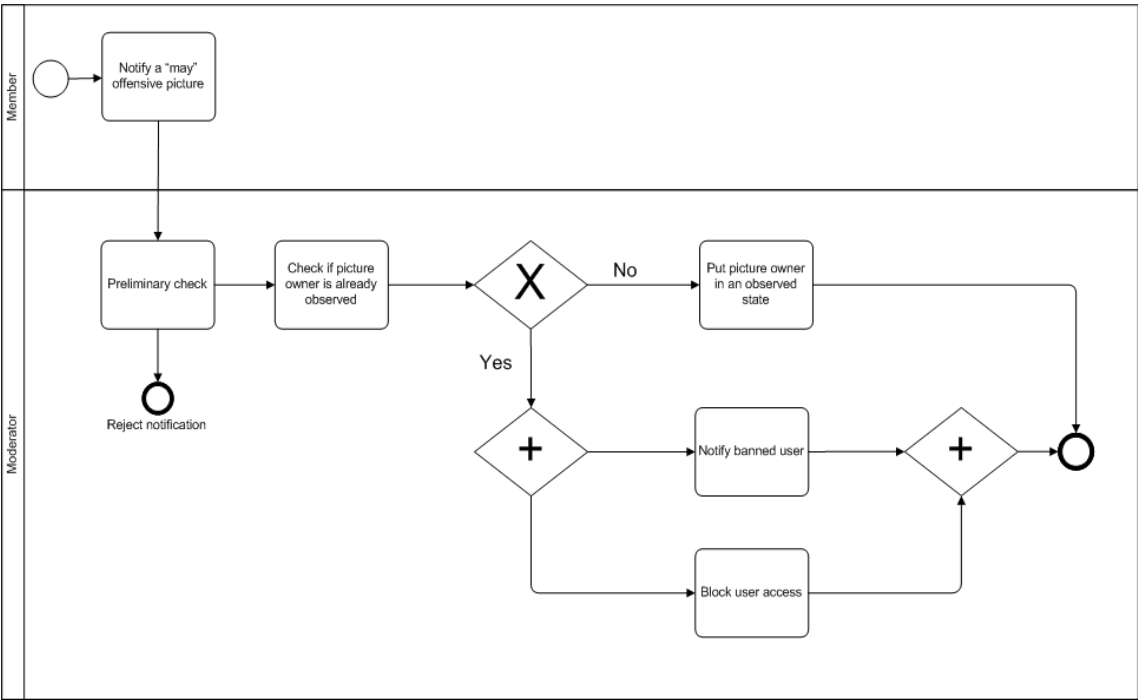


Figure 32: BPML diagram of the moderation workflow

Implementation

In this chapter we propose an implementation in WebML for results we outline in “Patterns” chapter. We decided to focus our efforts on back-end patterns only, because they enclose the real “community logic” and system reactions to members activities.

Data model extension

Data modeling for community-based applications follows the general guidelines for conceptual database design, possibly refined with procedures for data-intensive web modeling. To cope with the community representation the model is extended with entities and relationships. Also the user metadata is extended, with the addition of new attributes.

The default WebML basic data model allows to represent the User-Group-Module pattern we already discussed in the previous chapters. We extended the basic data model as hereby shown. We will briefly discuss this model in order to let the reader understand the model peculiarities. In order to avoid to stick onto details we will not discuss in detail every single entity and relationship.

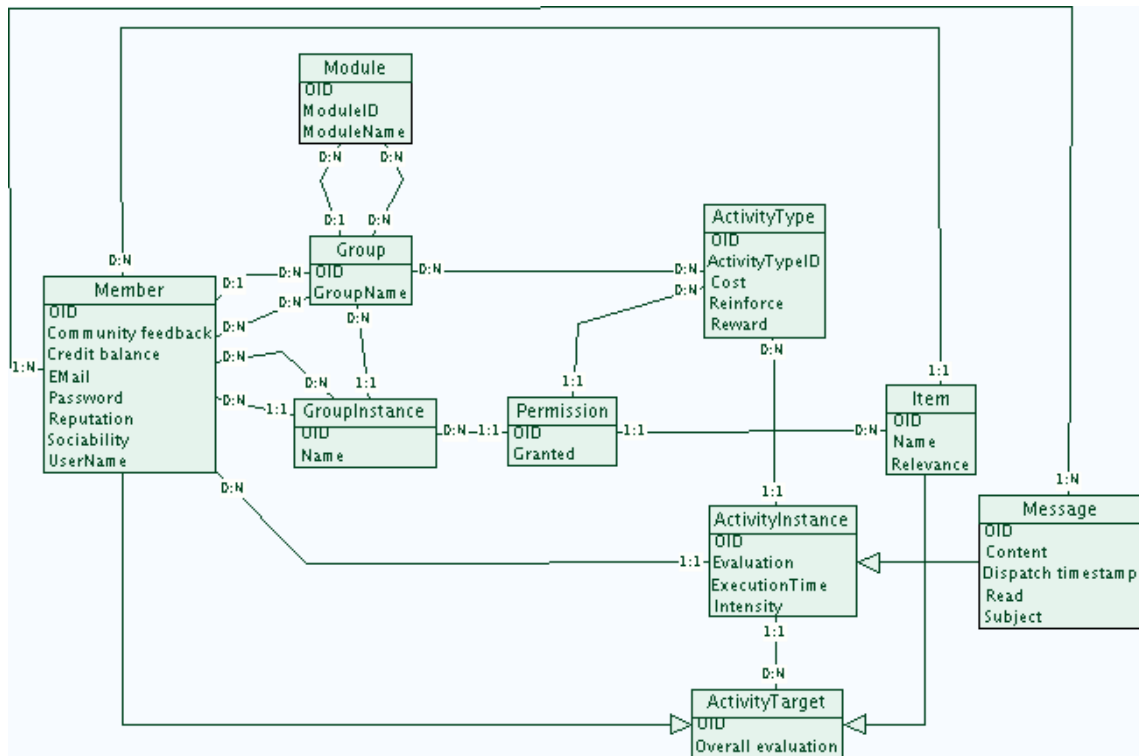


Figure 33: The data model extension

Member

Member is the extension of the standard User entity in WebML. Although it wasn't strictly necessary, we decided to change the entity name from *User* to *Member* for underlining the difference of perspectives when approaching the design of a community-centered application.

This entity is the extension of user, modified with the addition of new attributes: *community feedback*, *reputation* and *sociability*. Regardless the scale used for quantifications, *community feedback* indicates the overall opinion that the community has for the quality of the member's contributions, *sociability* the extent to which other community members are likely to establish social relationships with the member and *reputation* is a combination of the previous two according to community requirements³⁰.

Furthermore we introduce a *credit balance*, which is related the capability of the

³⁰ These attributes, according to what has been explained in the “Process” chapter, are *community state variables* because they represent the “community opinion” for the member in object. As we already explained, community state variables are manipulated by active rules and workflows, so quantitative aspects related to them will not be object of our discussion.

member to perform controlled activities within the community.

Item

The *Item* entity represents a generic element of interest within the community. It is related to the *Member* entity via the *Item-to-Owner* relationship (N:1), which links an item to the member who owns it.

Group

The *Group* entity represents groups of users within the application. This is the same as in standard WebML data modeling; groups definitions correspond to the roles defined within the application and directly impact on visibility of contents to users.

GroupInstance

The *GroupInstance* entity represents a collection of members. Differently from the *Group* entity, which, in a way, represents a group definition, *GroupInstance* represents an aggregation of individuals. *GroupInstance* is linked to *Member* by two relationships *GroupInstance-to-Founder* (N:1) and *GroupInstance-to-Participants* (N:N).

ActivityType

The *ActivityType* entity represents definitions of activities available within the community. It is characterized by the following attributes.

- *ActivityTypeID*. It is a value used application-wide for identifying the type of activity.
- *Cost*. It represents the cost of performing an activity; the performance of any activity produce a payment from the performer's credit corresponding to the cost of the activity type.
- *Reinforce*. It is a boolean value which indicates whether the activity type is reinforcing for the object (1) or not (0). This is necessary for distinguishing between positive activities and negative ones when *evaluating* them.
- *Reward*. It indicates the reward that the performer will get against each performance of the activity type.

The *ActivityType* entity is related to the *Group* entity in order to determine which groups of users are entitled to perform any activity.

Permission

The *Permission* entity is used for representing custom access policies for performing activities on items, which are usually defined from the item owner.

This allows each member to set, for each item she owns, grant/deny policies for each *GroupInstance* she founded to perform a particular *ActivityType*.

Module

The *Module* entity represents static modules of which compose the application. This is the same as in standard WebML data modeling.

ActivityTarget

The *ActivityTarget* entity represent an abstraction of what can be object of an activity performance. We already discussed how some activities are clearly social (social activities), and put in relation members between them, while others are oriented to items (contribution activities).

Regardless the type of activity, any contributions impacts on its object determining a variation of *relevance* and *evaluation*. These two attributes are community state variables and, how we already discussed, they can be modified by active rules and workflows.

ActivityInstance

The *ActivityInstance* entity represents the performance of an activity through an instantiation of *ActivityType*; this explains the *ActivityInstance-to-ActivityType* relationship, characterized by N:1 cardinality.

An *ActivityInstance* has always a *Member* who performs it (*ActivityInstance-to-Performer* relationships) and an object/target for the activity (*ActivityInstance-to-ActivityTarget*).

Furthermore, an *ActivityInstance* is described by the following attributes:

- *Evaluation*. We already discussed how each activity performance can be seen as implicit evaluation for the target of such activity. This attribute quantifies the implicit evaluation which the activity instance entails.
- *ExecutionTime*. It indicates the moment of the execution, through a timestamp.

- *Intensity*. This attribute indicates the intensity of the activity performance. Some activities have different degrees of intensity; in the case of a rating, for example, this attribute can report the rating that a user is giving to an item.

Message

The *Message* entity is an extension of *ActivityInstance* which represents a message exchanged between members. In this example we modeled a simple point-to-point message. This kind of entities have in general quite specific definitions which can reasonably be delegated to the application modeler. Nonetheless that of message is quite a common feature: this is why we decided to include it directly in the data model.

A message is described by the following attributes:

- *Content*. The text content of the message.
- *Dispatch timestamp*. The timestamp of the moment in which the message has been dispatched by the system.
- *Subject*. A text string containing the subject of the communication.
- *Read*. A boolean flag which indicates whether the receiver has already read the message or not.

The message is related to the *Member* entity by the *Message-to-Recipient* relationship. Conversely, the link from a message to the member who sent it is inherited from the *ActivityInstance* entity.

Back-end patterns

Overview

Back-end patterns are aimed to manage activities that members perform within the virtual community. In the most general case, according to the PPR chain we introduced in the “Patterns” section, each activity is made by the following steps:

1. *Permission check*. The system evaluates if the member has the right credentials to perform the activity.
2. *Payment*. If the activity is controlled, then it means that the performer is required to spend part of her balance for covering the activity cost. This step is not necessary if the activity is not controlled.
3. *Performance*. The actual performance of the activity; this step is completely up to the application to be developed.
4. *Evaluation*. The performance is evaluated and the overall evaluation of the object of the activity is updated.
5. *Reward*. If the activity is reinforcing, then a reward needs to be given back to the performer.
6. *Reputation adjustment*. If the object of the activity is an item, then the reputation of its owner needs to be adjusted according to the new overall evaluation of the item. If the object of the activity is a member then it's directly her reputation to be updated depending on the evaluation.

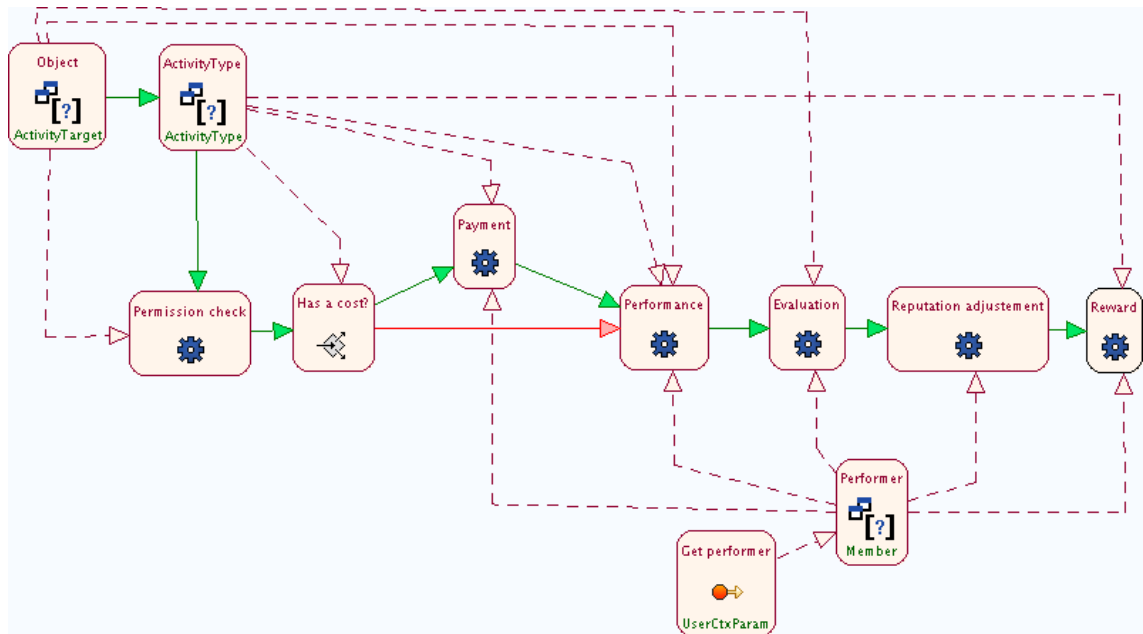


Figure 34: WebML diagram of the PPR chain

Evaluation

This pattern is activated on every activity that a member performs in order to evaluate the impact of such activity within the community.

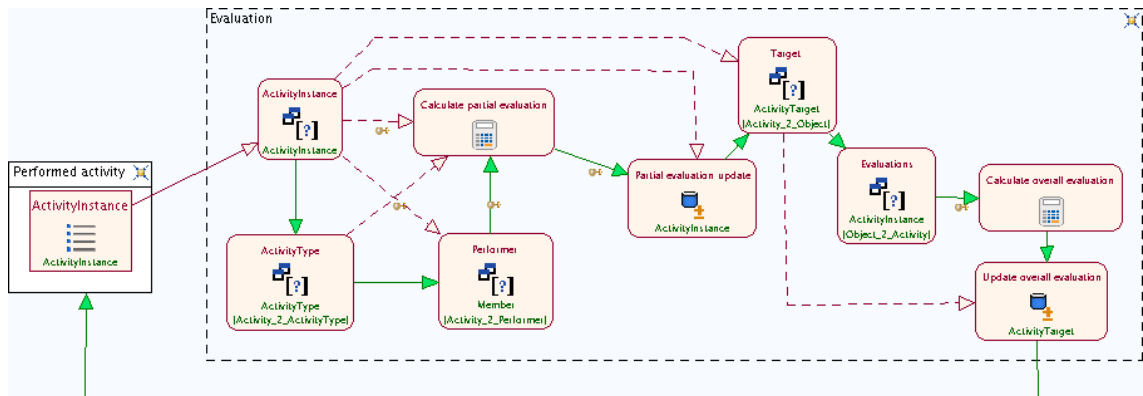


Figure 35: WebML diagram of the Evaluation pattern

First of all partial evaluation is calculated on the *ActivityInstance*. Then all the evaluations available for the *ActivityTarget* involved are gathered and used to calculate the overall evaluation of the *ActivityTarget*.

Notification

This pattern is activated every time the system needs to check a member's message box.

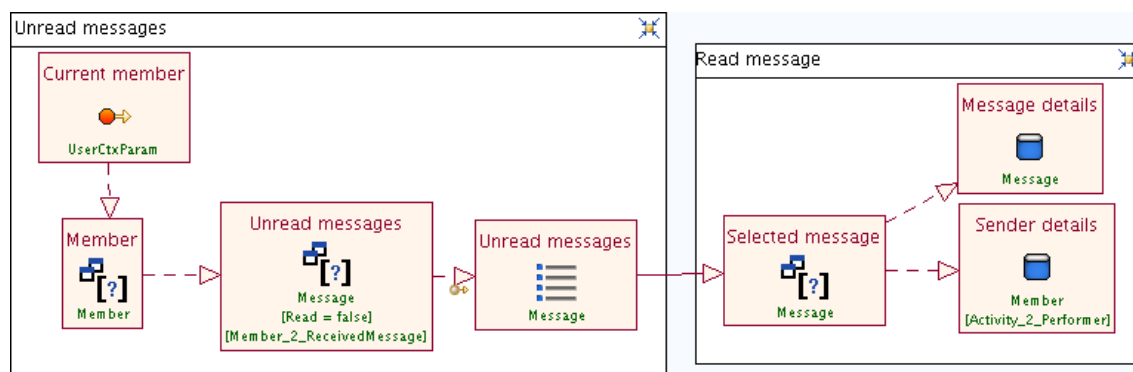


Figure 36: WebML diagram of the Notification pattern

As shown in the above diagram, it consists only on the selection of messages that haven't been read yet. Our assumption is that messages are forwarded to a third-party delivery service at the moment of dispatch, so that notification is directly handed by the specific protocol used for message forwarding (instant messaging, e-mail, etc.), as is depicted by the sending pattern reported below.

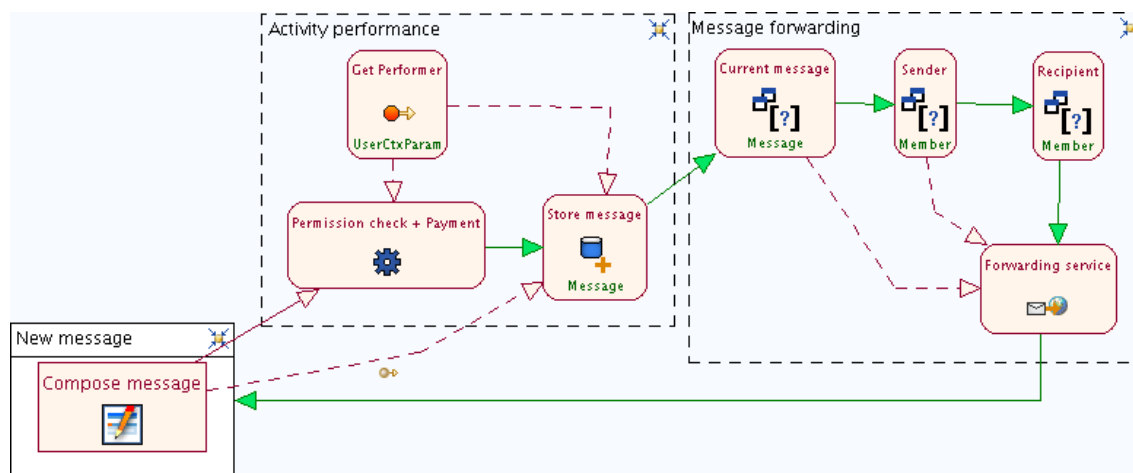


Figure 37: WebML diagram of the message forwarding operation

As shown in the above example, the sending of a message is treated as a non-rewarded activity, so it is subject to permission check and (eventually) to a payment. Then the message is stored and then forwarded to a third-party forwarding Web Service without evaluation nor reward.

Payment

This pattern is activated prior to every controlled activity that a member may perform. This pattern evaluates whether the performer can afford the activity or not before proceeding to the actual performance of the activity.

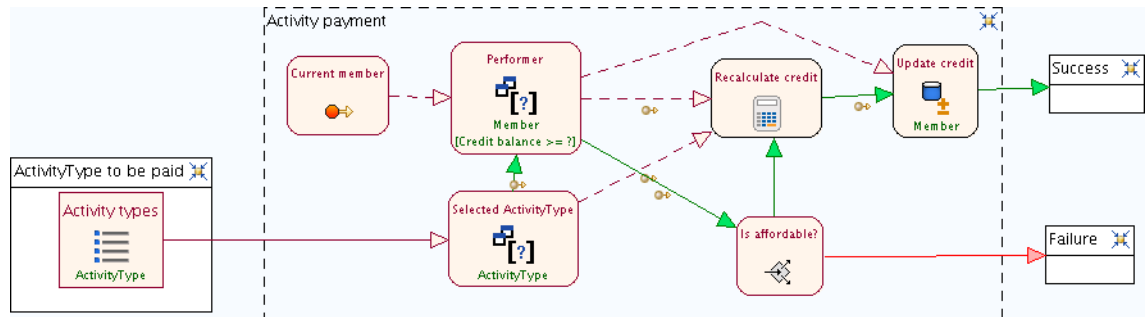


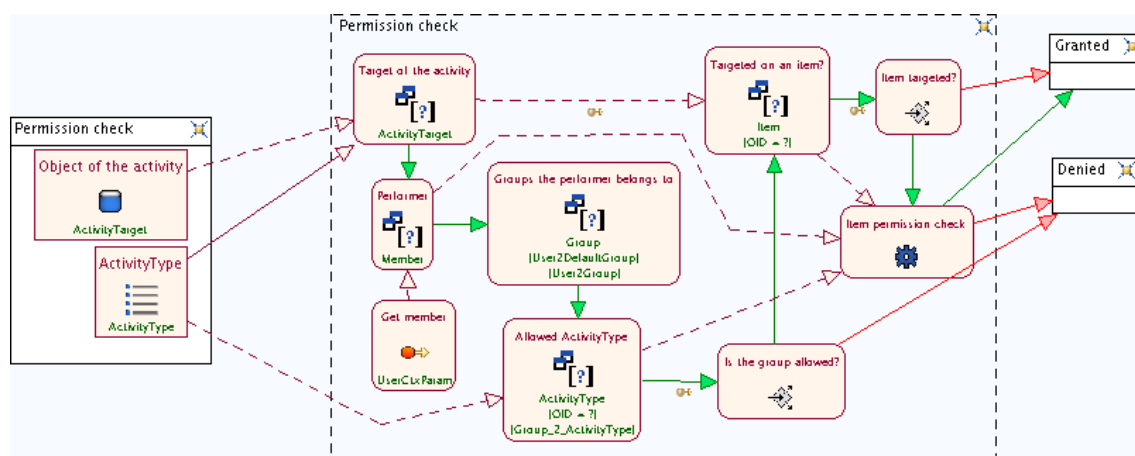
Figure 38: WebML diagram of the Payment pattern

From the definition of the activity that the member is going to perform, the pattern evaluates if the performer has enough credit balance to afford the activity; if she can the such balance, together with the activity type cost, is used to calculate the new balance before storing it into the performer's profile. If not the pattern returns a failure.

Permission check

This pattern is the first to be activated, as soon as a member of the community tries to perform an activity. This pattern check if the performer belongs to a group which is in charge to perform such activity. Then, in case of success and if the target of the activity is an item, the patter checks whether the owner of the item has set any specific restriction for that activity for *GroupInstances* that the member might be part of.

This pattern is a little complex; for the sake of clarity, we split the pattern in two WebML diagrams: the first depicting the control of groups, the second showing the permission check performed on the item, according to the owner's preferences.



The *item permission check* unit, appearing in the above figure is only a simplification used in order to keep the diagram clean; the actual content of the unit is depicted below.

Figure 40: WebML diagram of the Permission check pattern (on items)

This pattern is activated after the evaluation, each time that the activity performed is a contribution and, because of this, as an item as object. In these cases, after the evaluation, the relevance of the item needs to be updated.

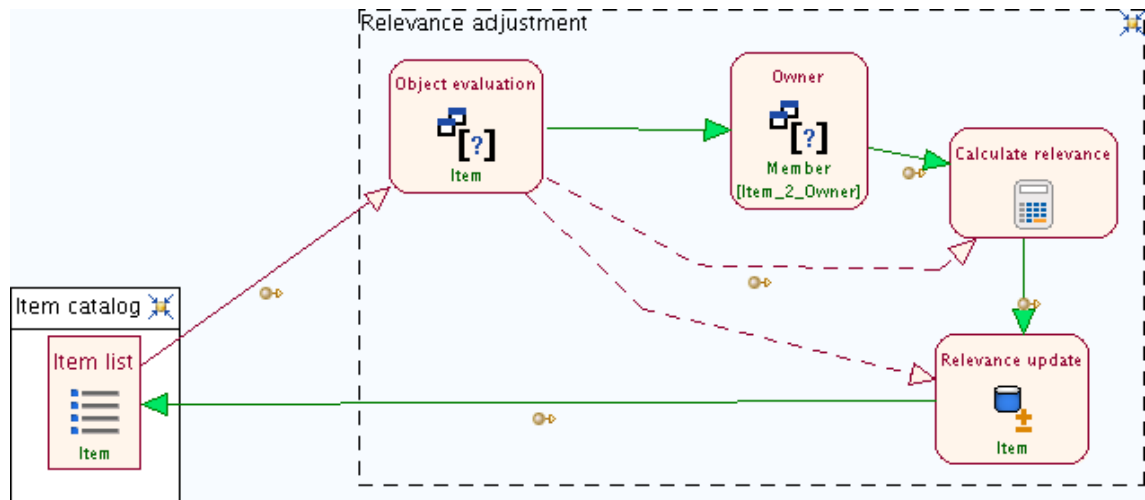


Figure 41: WebML diagram of the Relevance adjustment pattern

As it's clearly depicted in the figure above, the relevance of an item generally depends from the overall evaluation that the community has given for the item, and on the reputation of its owner.

Reputation adjustment

This pattern is activated against any contribution activity, which that means every time that an activity has an item as object. We already explained how each contribution activity can be seen as an implicit evaluation: because such activities are evaluated, they in general determines a variation of the owner reputation.

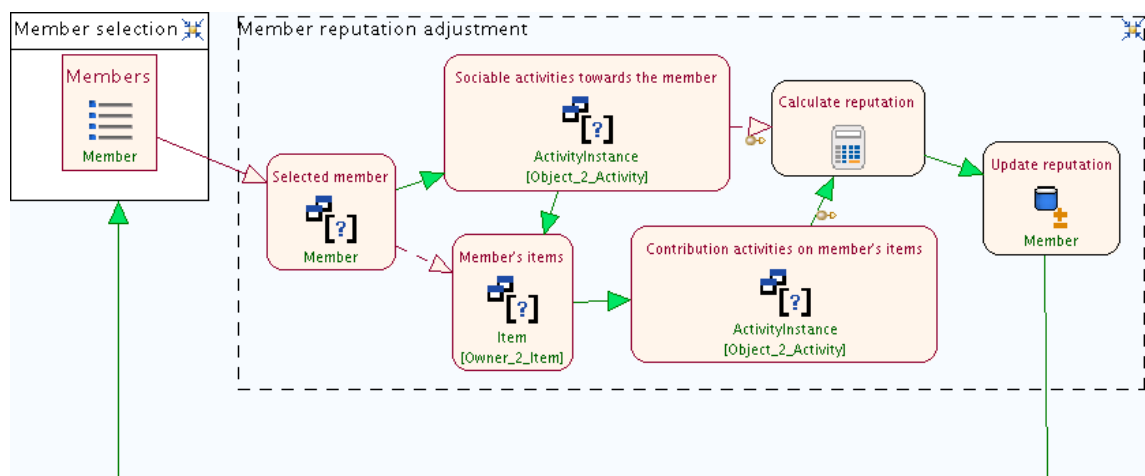


Figure 42: WebML diagram of the Reputation adjustment pattern

As shown above, the reputation of a member is evaluated according to two different measures: the evaluation of all the contribution activities performed on her items (*community feedback*) and the evaluation of all the sociable activities performed on her (*sociability*); these two values take part in the member reputation calculation.

Reward

This pattern is activated against the performance of an activity instance performed by a member.

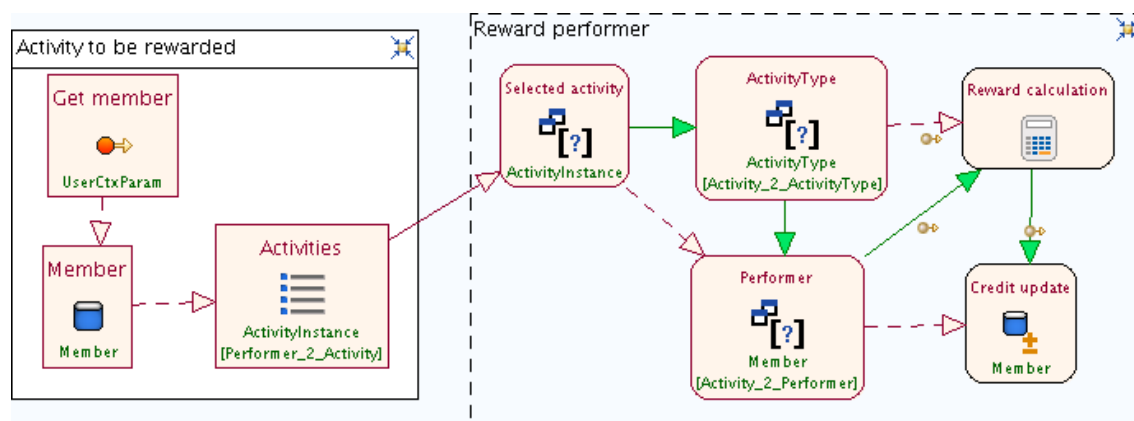


Figure 43: WebML diagram of the Reward pattern

From an activity instance the activity type is taken for extracting the reward; then the reward is used for a calculation that adds to the performer's credit the reward provided by the activity type. The result of such calculation is then stored into the member's profile.

Syndication

This pattern is activated each time an item or a container of items changes. It simply needs to update a formal description for the item.

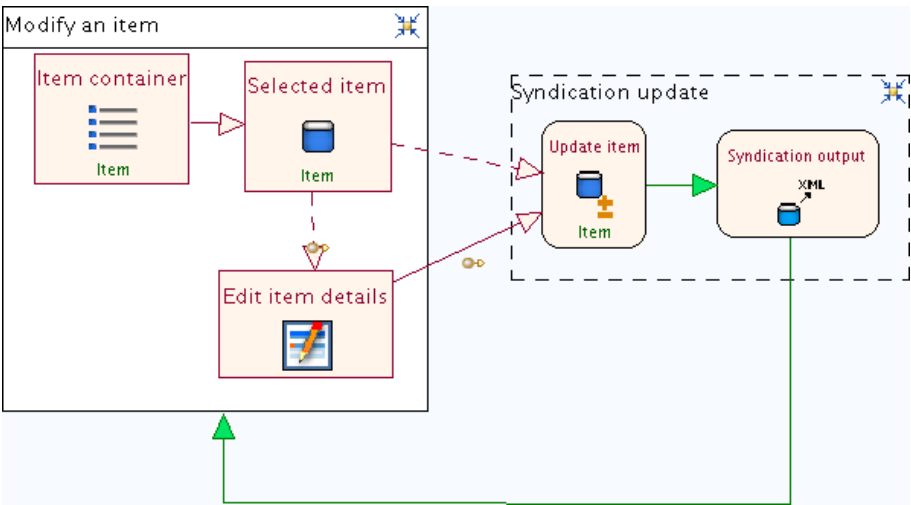


Figure 44: WebML diagram of the Syndication pattern

Case study

In this chapter we present a case study for describing the development of the Mie 2.0 application, the community based extension of the Made In etaly (MIe³¹) digital store.

In order not to move the reader's attention from the WebML Community development process to the real complexity of the MIe business context, we decided to simplify the real case.

³¹ Made In eTaly digital store (MIe) <http://www.madeinitaly.it>

Context

Made In etaly (MIe) is a joint venture of music companies which through a consortium operates several labels for the development and the promotion of Italian music worldwide. The MIe web portal is basically a digital store with a catalog of more than 15.000 tracks, which aggregates publications from about 15 independent music labels. Its core business is to sell products and services to third party digital stores (iTunes, DeeJay Store, MSN Music, Y! Music, etc.) and digital mobile telephone companies (Tim, Vodafone, H3G, etc.).

The MIe consortium now decides to open the platform to the B2C market in order to propose its digital catalog to the public. In order to achieve this, the MIe consortium is willing to build a community platform in order to attract new consumers, more independent labels and to scout for emerging artists.

The idea is to give each community member a “personal space” where they can act both as music consumer, by buying the MIe digital products, and as music independent producer by uploading their digital creations. Furthermore, in order to foster participation, MIe wants to reward best emerging artists with a promotion within the community.

Requirement analysis

Mie in a nutshell

The current Mie web portal is an institutional website, which only provides content management services to its associated members in order to let them manage the digital catalog. The current platform has been designed for the following actors: *Mie administrators*, *Mie associates*, *Mie editor* and *external users*.

The main goals of the current platform were:

- to propose institutional information (news, consortium vision and philosophy, organizational structure, members);
- to propose an overview of the main products and services;
- to give associated labels the possibility to configure their own digital catalog (adding sub-labels, artists, multimedia files) and publish their own news within the portal;
- to show the entire Mie digital catalog to the public.

The main business object of such environment were:

- *digital product*: songs, videos, ringtones and Java games for mobile phones;
- *digital catalog*: the catalog according to which digital products are aggregated by virtual digital supports (CD, DVD) and grouped by genres;
- *news*: published both by *administrators* and *associates*;
- *label*: every *associated* runs a set of labels under which digital products are published;
- *artist*: each artist can be associated to many labels.

Community requirements elicitation

According to the new MIE business strategy the new business goals are:

- to sell products through the web platform to retail market;
- to acquire new independent labels and sell them the platform services in order to publish their artists and catalog;
- to scout for new emerging artists.

In order to reach such goals, we propose to design a community platform which will achieve the following objectives:

- To give consumers, labels and independent artists a “personal space” through which they can publish, collect and share their own contents and digital products.
- To give registered members the capability to invite people to their personal space and to form groups of interest.
- To create a system according to which each member of the community can rate and flag others' digital products.
- To create a system according to which the reputation of a member depends both on the relevance of creations she published and on the extent to which she invited artists that effectively succeeded and became popular within the community.

User hierarchy extension

In order to represent the community members taxonomy, it is necessary to extend the current user and group definitions. According to the methodology we introduced in the “Process” chapter, new members roles are reported below.

By the content management point of view:

- *MIE member*: it is the representation of a generic member of the community, according to the WebML definition of internal user. It has a proper personal space with functionalities that allow to edit a profile, add, publish and share contents and to invite people by forming groups and social relations as friendships and collaborations.

By the community governance point of view:

- *group moderator*: each member is the moderator of the groups hosted in her personal space;
- *artist moderator*: it is the community member role, whose responsibility is to evaluate artists' digital creations, to find both artists flagged as “offensive artists” and breaking-through artists (artists whose reputation is growing fast). In the first case the flagging operation is aimed to draw the MIE moderator's attention on the flagged artist;
- *MIE moderator*: it has the complete control on community members. Its main tasks are to ban offensive members and to promote to the state of artist members who produces most popular contents.

By social relationships:

- *artistic collaborator*: it is a MIE member which collaborates to the publishing of a particular digital creation;
- *friend*: it is a MIE member that shares its personal space in order to discuss to a particular topic or share contents;
- *fan*: it is a MIE member that declares to like a particular artist; fans form fan-clubs of an artist, areas where they can share contents and speak about their stars. Artists can publish events within their fan-clubs such as new album releases and live dates.

By behavior:

- *normal user*: such MIE members like to buy and download contents they enjoy, but don't contribute much to upload digital creations, or discussing within groups of interests;
- *artist*: this MIE member publishes a lot of digital creations;
- *talent scout*: this MIE member likes to rate and comment others digital creations; their observations are appreciated by the community;
- *PR*: this MIE member likes to invite new people, to form groups and to tailor many social relationships with others; it also actively promotes artists and events.
- *passive*: this MIE member is registered but mostly idle, it is not influent for the community development;

- *offensive*: members flagged as offensive by others;

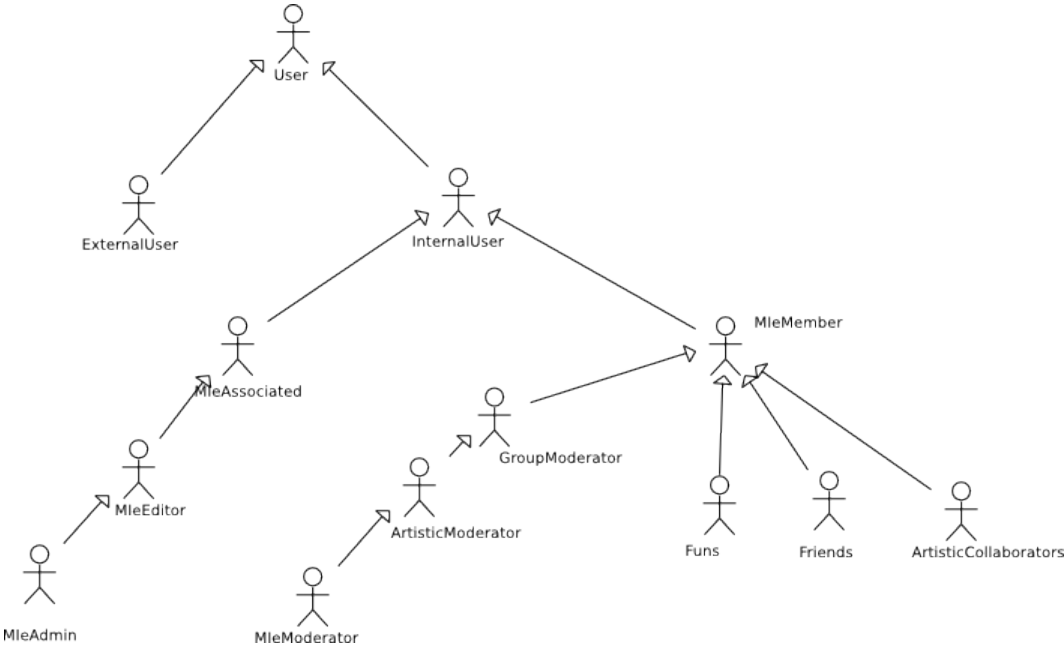


Figure 45: The user hierarchy for the MIE platform

Design community services by patterns

In this paragraph we'll refer to patterns we distilled in the chapter “Patterns”, in order to speed up the functional requirements elicitation for the application. In the table reported below, we summarize the main services and the related pattern which the platform needs to cover in order to satisfy the primary needs of the MIE community.

Pattern	Motivations	Service
Browse by connections	To help members to find friends which similar profile	Friends recommendation service
Browse by tag	To help members to traverse the digital catalog	Digital catalog browsing by tag service
Exportation	To give members the possibility to publish news and their play lists on their personal blog	News RSS feed service Playlist exportation widget service
Organization	To give members the possibility to own an online personal space where they can organize their downloaded/uploaded contents	Personal member space services Playlist manager service

Methodology and patterns for developing community-based web applications with a model-driven approach

Pattern	Motivations	Service
Publication	To give the possibility of members to publish contents for the community	Multimedia content upload service
Rating	To give the possibility of members to rate and flag contents	Rate service Flag service
Recommendation	To give the possibility of a member to suggest a particular product to her friends	Product recommendation service
Group creation	To give the possibility to members to form groups of interest	Group content management services
Invitation	To give the possibility to members to invite people who don't already join the community	Invitation service through e mail service delivery
Permissions setting	To give members the possibility to set-up the access policy to their personal space and contents.	Access policy management services
Relationship setting	To give member the possibility to establish friendships and art collaboration relationships.	Friends meeting services Art collaboration services
Social visualization	To give members the possibility to easy understand the role of each others members into the community	Member state management services Automatic role detection services Member state visualization services
Talk	To give the possibility to exchange information between users	Message exchanging services

Community design

The moderating-fostering strategy

In order to show an example about how to design a moderating-fostering system by state variables, we propose a strategy according to which each member is evaluated by three indexes:

- *Reputation*. It's an index expressed in points that represents the community perception of a member as an artist. Reputation is calculated by weighting the overall evaluations given to the published contents of such member.
- *Sociability*. It's an index expressed in points that represents the capability of a member to involve other members in the community “life”. It is calculated by the computation of the average reputation of members that are connected to the evaluated person.
- *Consensus*. It's an index expressed in points which represents the community perception of a member's reliability in criticizing contents. Consensus is calculated by weighting the overall evaluations given to the published critic reviews of the observed member.

The overall “value” of a published content (track, video, ring tone) is represented by the relevance of such content and it is calculated on the average evaluation scores weighted on the reputation of the evaluators. It is measured in points.

By reputation and relevance, it is possible at the same time to stimulate members to publish good quality contents (in order to acquire importance and celebrity in the community) and to prevent fake contributions or bad publications (because community evaluations impact on the publishers' reputations).

Member sociability is studied in order to foster people to contact other members, it is

introduced to foster members to tailor relationships and invite non-members to join the community. This aspect was introduced to involve people who would like to distinguish themselves as PR.

Member consensus is studied to involve non-artist members in the community life, by giving them the possibility to play as critics. This strategy is introduced in order not to cut-off people who aren't interested in publishing contents, but rather in their usage.

Both members and contents can be flagged as offensive every time they are evaluated so by the community; when someone is flagged as an offensive member, an inspection process is enacted. In such process the moderators evaluate the real offensiveness and decide to adopt a specific action against the flagged member, like calling back to order or banning from the community.

Community state variables

In order to map the community state, we need to model the state behavior of members and published contents.

While member's behavior can be mapped into behavioral groups elicited above (normal user, artists, talent scout, PR, passive, offensive), the published contents behavior need a specific design. For the sake of simplicity we'll use a five stars evaluation scale: *very low*, *low*, *mid-low*, *middle*, *high*, *very high* popularity.

Both members and published contents can move to the special state *offensive* when a member *flags* another member's items as offensive.

To map the state variable of a member behavior we propose to split it into the following sub-state variables:

- reputation
- sociability
- consensus
- offensiveness

To map the state variable of a published content behavior we propose to split it into the following sub-state variables:

- relevance
- offensiveness

Such sub-variable aren't mutually independent; their relationships of influence are as follows:

1. *Reputation-Relevance*. The reputation of a member is weighted on the overall relevance of her published digital contents.
2. *Sociability-Reputation*. The sociability of a member doesn't depend only on the number of connections but also on the reputation of the connected members.
3. *Relevance-Reputation*. The relevance of any evaluation is weighted on the reputation of the of evaluator.
4. *Consensus-Relevance*. The consensus of a member represents the community acceptance of critic reviews that such member attaches to digital productions
5. *Member-Published content offensiveness*. Every time a published content is defined offensive the owner offensiveness is set to true.

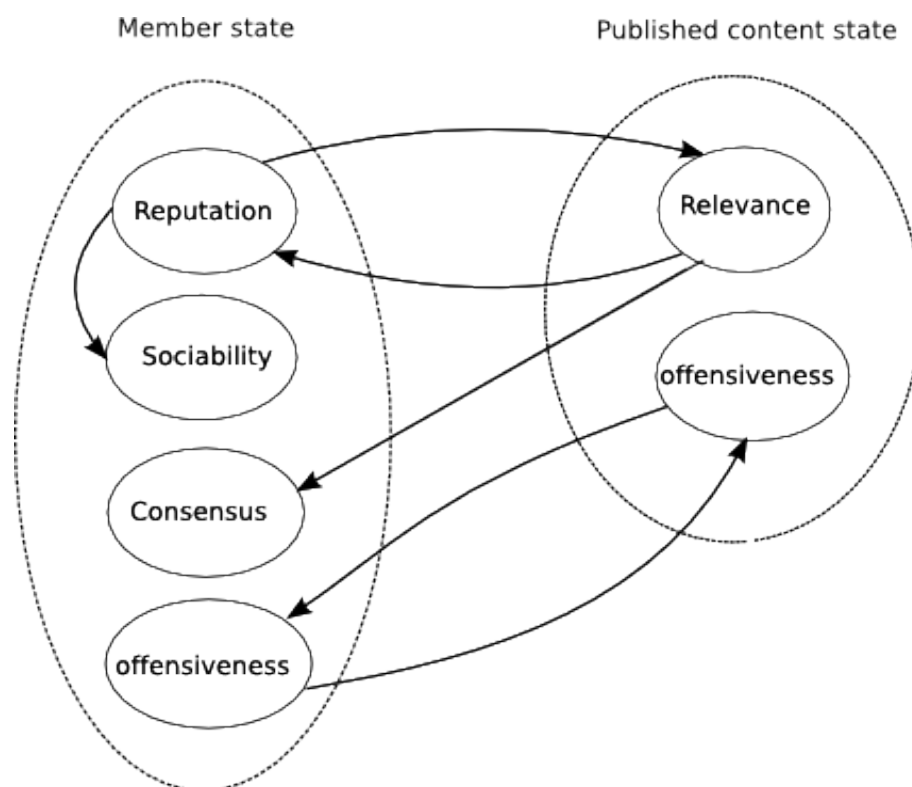


Figure 46: Dependences between community state variables

Established the dependences graph now it is necessary to specify all the aspects related to the each state variable:

Member state

- *Description*: describes the state of a member
- *Measuring scale*: points
- *Users involved*: members
- *Triggering events*: the reputation and/or sociability and/or consensus of the member change.
- *Action*: to update member state.
- *State transitions*. In order not to complicate the state transitions table we report a radar graph according to which it's possible to see state changing qualitative thresholds.

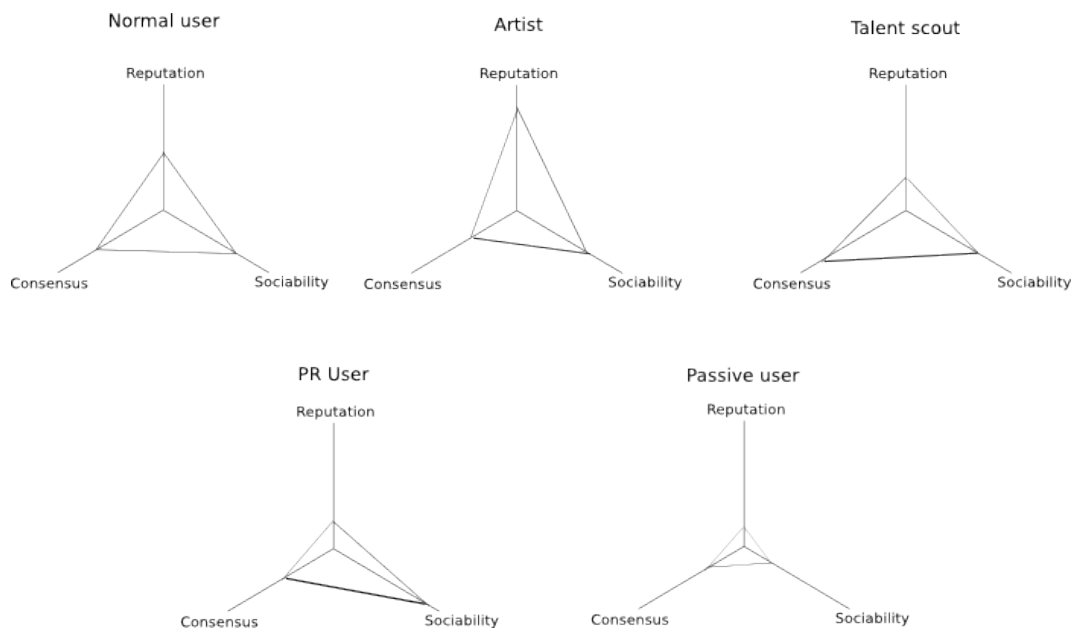


Figure 47: Member states and thresholds

- *Influences on other state variables*: none.

Reputation

- *Description*: describes the reputation of the member based on the *relevance* of her published contents (tracks, video, etc.)
- *Measuring scale*: points
- *Users involved*: members

- *Triggering events*: the relevance of a published content by the member is updated.
- *Action*: to update member reputation
- *State transitions*:

Reputation (pts)	passive user	normal user	artist
passive	-	50<pts<2000	
normal user	pts<=50	-	pts>=2000
artist	pts<=50	50<pts<2000	-

- *Influence on other state variables*: it can affect *sociability* and *consensus* of the other members.

Sociability

- *Description*: describes the capability of a member to involves high quality members.
- *Measuring scale*: points
- *Users involved*: members
- *Triggering events*: a reputation of a “connected” member is updated
- *Action*: update the sociability of the member
- *State transitions*:

Reputation (pts)	passive user	normal user	PR user
passive user	-	50<pts<500	pts>=500
normal user	pts<=50	-	pts>=500
PR user	pts<=50	50<pts<500	-

- *Influence on other state variables*: none

Consensus

- *Description*: describes the art critics ability of the member based on the *relevance* of her published critics reviews.
- *Measuring scale*: points
- *Users involved*: member

- *Triggering events*: the relevance of a published content by the member is updated.
- *Action*: to update member consensus
- *State transitions*:

Reputation (pts)	passive user	normal user	talent scout
passive user	-	50<pts<2000	
normal user	pts<=50	-	pts>=2000
talent scout	pts<=50	50<pts<2000	-

- *Influence on other state variables*: none

Contributor relevance, critic review relevance

- *Description*: describes the relevance of a published content or art critics review based on the members evaluations.
- *Measuring scale*: stars (0: very low; 1: low; 2: mid-low; 3: middle; 4: high; 5: very high);
- *Users involved*: members
- *Triggering events*: a member evaluate a published content, critics review
- *Action*: update published content relevance, update critics review relevance
- *State transitions*:

Relevance (stars)	very low	low	mid-low	middle	high	very high
very low (0)	-	+1	+2	+3	+4	+5
low (1)	-1	-	+1	+2	+3	+4
mid-low (2)	-2	-1	-	+1	+2	+3
middle (3)	-3	-2	-1	-	+1	+2
high (4)	-4	-3	-2	-1	-	+1
very-high (5)	-5	-4	-3	-2	-1	-

- *Influence on other state variables*: member reputation and consensus.

Mapping state variables transitions

For each state variable (*member state* and *published content state*) and the relative sub-state variables we define the transitions with the ECA-like notation proposed in the “Process” chapter.

In order to map the member state transition we define the *member state-update* rule:

```
rule [ member state-update ] controllers [ none ]  
    when [ member.Reputation is updated;  
            member.Sociability is updated;  
            member.Consensus is updated]  
    if [ member.offensiveness==false]  
    then [ updateState (member)]
```

In order to map the published content state transition we define the *published content state-update* rule:

```
rule [ published content state-update ] controllers [ none ]  
    when [ publishedContent is updated]  
    if [ publishedContent.offensiveness==false]  
    then [ updateState (publishedContent)]
```

In order to map the member reputation transition we define the *member reputation-update* rule:

```
rule [ member reputation-update ] controllers [ none ]  
    when [ member.publishedContent is updated;  
    if [ member.offensiveness=false  
        AND member.publishedContent.type == digitalCreation  
    then [ updateReputation (member)]
```

In order to map the member sociability transition we define the *member sociability-update* rule:

```
rule [ member sociability-update ] controllers [ none ]  
    when [ member.connectedMember.reputation is updated]  
    if [ member.offensiveness==false ]  
    then [ updateSociability(member)]
```

In order to map the member consensus transition we define the member consensus-update rule:

```
rule [ member consensus-update ] controllers [ none ]  
    when [ member.publishedContent is updated]  
    if [ member.offensiveness==false  
        AND member.publishedContent.type==criticismReview ]  
    then [ updateConsensus(member)]
```

In order to map the content published content relevance transition we define the content published relevance-update rule:

```
rule [ content published relevance-update ] controllers [ none ]  
    when [ member.evaluateContentPublished]  
    if [ member != contentPublished.owner  
        AND member.offensiveness==false]  
    then [ updateRelevance(contentPublished, member)]
```

Two further rules need to be implemented: one to represent the ban transition of a member signaled as offensive the latter is the block transition for offensive contents.

The *member ban-rule* is the following:

```
rule [ member ban-rule ] controllers [ Me moderators ]  
    when [ member.offensiveness.isUpdated]  
    if [ member.offensiveness==true]  
    then [ callTheBanningProcess(member)]
```

The *published content block -rule* is the following:

```
rule [ published content-rule ] controllers [ Group modetator, Artistic moderator]
```

```
when [ publishedContent.offensiveness.isUpdated]  
if [ publishedContent..offensiveness==true]  
then [ callTheContentBlockProcess(publishedContent.member)]
```

Define the update functions

According to the methodology proposed in the chapter “Process”, it is possible to express in a pseudo-mathematical language the functions representing the state variable value update. Such language, during the implementation phase can be easily mapped through ad-hoc algorithms written in the programming language of the application platform.

In this case the following functions have to be implemented:

- `updateState(member);`
- `updateState(publishedContribution)`
- `updateReputation(member)`
- `updateSociability(member)`
- `updateConsensus(member)`
- `updateRelevance(member)`

In order to model the update of member state we can define the **updateState** function as follows.

Given a member i at the instant t her state is given by the function:

$$State_i(t) = f(Reputation_i(t), Sociability_i(t), Consensus_i(t))$$

Where f is a custom³² function that switches the state of the member i by weighting the values returned by the functions that calculate reputation, sociability and consensus of such member at the instant t .

Now, the reputation of a member i at the instant t is given by the function **updateReputation** defined as follows.

Taking $W_i(t)$ as the set of the all O contents published at the instant t whose

³² For custom we intend a function which computational nature can be defined by the community designer as her peculiar needs. We adopt such notation to do not add more complexity to the definition of state variable update function

owner is the member i and $\Gamma_o(t-1)$ the set of all the evaluations given to o before the instant t , the reputation of i at instant t is given by the custom function g :

$$Reputation_i(t) = g(Relevance_o(t-1) \forall o \in W_i, \|\Gamma_o(t-1)\|)$$

Where the relevance of a published content o at the instant t is given by the **updateRelevance** function defined as follows:

$$Relevance_o(t) = h(Relevance_o(t-1), Evaluation_o(t))$$

h is a custom function that adjusts the relevance of a published content o taking into account the relevance of o before the instant t and the average of all the evaluations performed on o at until the instant t . It is represented by the following definition:

$$Evaluation_o(t) = Avg(e_{o,j}(t) \forall j \in \Gamma_o(t))$$

Where $e_{o,j}(t)$ is the evaluation of the member j on o at the instant t weighted on the j -member's reputation before the instant t , defined as follows:

$$e_{o,j}(t) = Reputation_j(t-1) \cdot Intensity_{o,j}(t)$$

Intensity represents the effective value of the evaluation that the j -member performed on the o -contribution at the instant t , corresponding to the number of stars given.

Now in order to calculate the sociability of a member, a further definition needs to be given to explain the **updateSociability** function.

Taking $Z_i(t-1)$ as the set of all the z members connected to the member i before the instant t , the sociability of i is computed as the average of all the reputations of members in $Z_i(t-1)$:

$$Sociability_i(t) = Avg(Reputation_z(t-1) \forall z \in Z_i)$$

In order to accomplish the model representation of the state variation of a member, it is necessary to define the consensus of a member. The **updateConsensus** function is defined as follows.

Taking $C_i(t)$ as the set of the all c critic reviews at the instant t whose owner is the member i , and being $\Gamma_c(t-1)$ the set of all the evaluations given to c before the instant t , the consensus of i at instant t is given by the custom function p :

$$Consensus_i(t) = g(Relevance_c(t-1) \forall c \in C_i, \|\Gamma_c(t-1)\|)$$

Where the relevance of c is calculated exactly as the relevance of published items.

The definition of published contents state transition is easier then the member state transition model; the `updateState` function can be expressed as follows.

Given a published content p its state at the instant t is given by the function:

$$State_p(t) = f(Relevance_p(t))$$

Where f is a custom function which switches the state of p by evaluating the result of its relevance.

Define the ban process

We propose a BPML representation of the “*ban process*” enacted when the *member ban-rule* is launched.

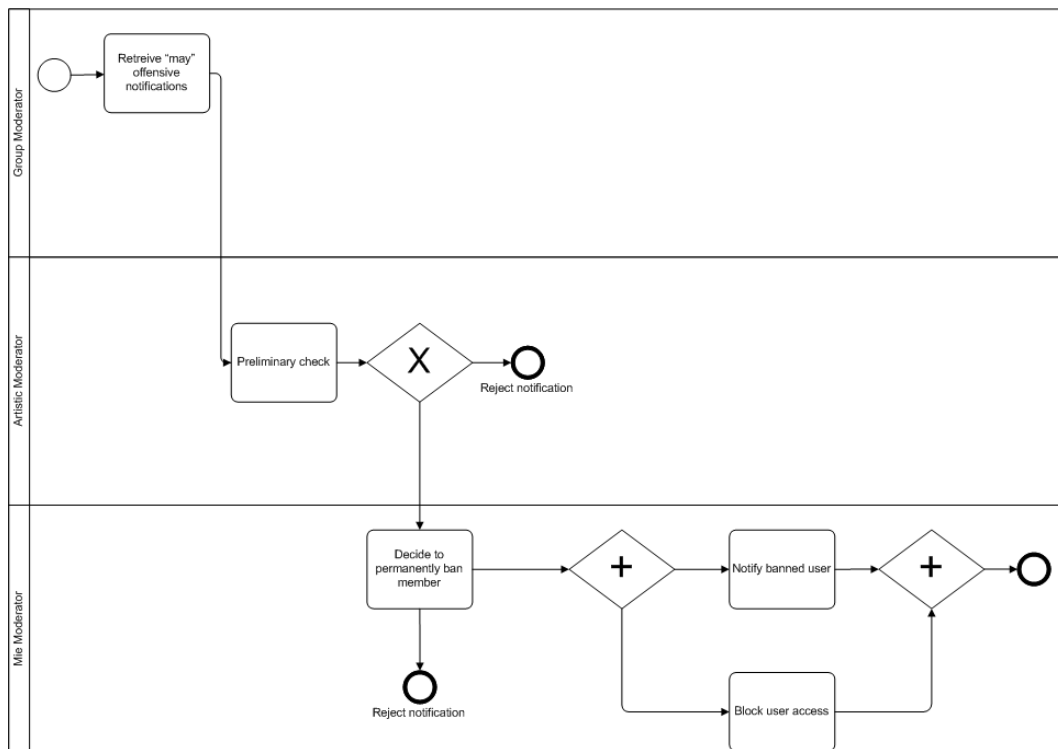


Figure 48: BPML diagram for the banning of a member workflow

Conclusions

In this chapter we draw conclusions for the work that we have done, suggesting further developments for which our work opened the way.

Results

Starting with the on-the-field practice, made during our stage at Web Models s.r.l., to the final delivery, this work has altogether taken us about 8 months in order to be accomplished. We realized from scratch the B2B portal of the *Made In etaly* consortium³³ and we set the basis for bringing the WebML model-driven development to the issue of designing community-based web applications. The WebML process extension we proposed can actually be essayed for designing community-based platforms. Patterns can be translated into practice extending WebML primitives; furthermore specific units can be designed and implemented in order to bring the community-based development support to the WebRatio tool.

³³ <http://www.madeinitaly.it/>

Further works

We ran mostly of our work as a research work. Literature we've explored used to address very specific issues of virtual communities, taking the matter under many different perspectives. Our efforts have been aimed to the achievement of a broader overview of the subject, trying to provide a methodical approach to the issue of designing community-based applications (an area counting much more success cases rather than consolidated results).

During our work, due to the nature of the research, we've often touched lightly interesting topics that might have deserved a greater attention. In many cases we had to stop in order not to get lost in unexplored territories; we hereby provide a list of such arguments, looking forward to suggest topics for further analysis and researches.

- *Scalability studies.* During our work we focused all the attention on the representativeness of the model, leaving concepts like computational complexity and scalability apart. The capability of such system to scale well for large community certainly deserves a more accurate analysis.
- *Integration of data mining systems.* The relevance of data mining techniques for community-based applications is crucial both for business purposes as for the community promotion. Data mining can be used to outline community topology, subgroups dynamics, trend-setting phenomena and so on. Our work has taken these aspects into consideration only for representing relationships between objects but, due to the specificity of the topic and to the non-suitability of WebML for representing off-line elaborations, the issue has been deferred to later works.
- *Communities as collaborative environments.* Our model of virtual community is capable of describing atomic actions performed by a single member. A further work could explore the possibility of modeling complex actions, such as collaborative works and/or workflows of activities for modeling non-elementary activities within the community.

- *Community state monitoring.* We defined community state variables using active rules, which corresponds to represent virtual communities as finite state machines; a further work could inspect the possibility to improve such modeling, inspecting in-depth theoretical concepts (like state reachability, optimizations, non-deterministic situations) and applying them to community monitoring also studying different community models using stochastic, non-deterministic representations.
- *Join between different social networks.* Patterns we outlined with our work provide a generalization of concepts that most popular social networks implement in practice. With special regard to relationships, a very interesting topic for further works could be the design of a decoupling service, capable of getting interfaced to most popular social networks, and to join data between different communities in order to perform cross-community queries (for instance: “find all the Flickr photos of my Linked-in 1st degree connections tagged with the word «holidays»).

Ringraziamenti

Un sentito ringraziamento a Nicoletta Di Blas e Caterina Poggi per il supporto offerto durante le ricerche preliminari ed a Piero Fraternali per l'aiuto datoci nei momenti di maggior bisogno. Vorremmo anche ringraziare tutti a Web Models, per averci dato la possibilità di misurarci con un contesto professionale così importante. Un ringraziamento anche a Michela Frigerio ed a Valentina Riva per i suggerimenti dati durante la stesura di questo documento.

Lorenzo

Alla mia famiglia, a Stefano, Sandro e mia madre in particolare, per il supporto e la fiducia dimostrata durante tutti i miei studi. A Sara che mi è sempre stata vicina in un modo così prezioso ed alla sua famiglia per l'affetto che non mi ha mai fatto mancare. A mio nonno Angelino, che mi diceva sempre: «Studia, mi raccomando, che diventerai qualcuno!», anche quando ero un bambino e finite le scuole andavo a trovarlo. A tutti i compagni di studi incontrati lungo la strada, in particolare Fabio, Flavio e Gabriele perché anche grazie al loro aiuto sono riuscito a superare i momenti più difficili. A Matteo, che ha lavorato con me con costanza e dedizione, sopportandomi e dimostrando grande comprensione ed umanità. Alla famiglia Silva per l'infinita gentilezza e per la premurosa ospitalità. A tutti gli amici, che con me, gioiranno per questo risultato.

A tutti loro vanno i miei più sentiti ringraziamenti.

Matteo

Ecco, finalmente è giunto il momento di ringraziare, con queste poche righe, quanti mi hanno accompagnato in questo percorso, fatto certamente di alti e bassi, ma

soprattutto di entusiasmi e soddisfazioni.

In primis i miei genitori che mi hanno sostenuto con passione e convinzione, permettendomi di studiare serenamente, sopportando con generosità “questo Matteo – spesso – con la testa tra le nuvole”. Li ringrazio per non aver mai smesso un secondo di credere in me.

Ringrazio mio fratello Marcello, con il quale ho sempre condiviso un grande amore per la conoscenza oltre che a numerose nottate a studiare, o forse più a scherzare insieme nelle pause per esorcizzare la stanchezza.

Grazie a te nonna, sei molto di più.

Grazie a te Alice, che con la tua dolcezza hai sempre ascoltato i miei discorsi, come dici te, “da ingegnere”, per tutte quelle volte che mi hai accolto con il tuo sorriso e mi hai dato la carica nei momenti duri.

E poi grazie a voi, Ros e Niki, veri amici.

E infine come non ringraziare te Lorenzo.

Bibliography

All references are attached at the end of this document.

-
- i Paul Graham (November 2005). Web 2.0. Retrieved on 2006-08-02. "I first heard the phrase 'Web 2.0' in the name of the Web 2.0 conference in 2004."
- ii McKinsey Global Survey. "How businesses are using Web 2.0" (2007)
- iii H. Rheingold. "The Virtual Community: Homesteading on the Electronic Frontier", London: MIT Press. (ISBN 0262681218) (2000)
- iv S. Tardini, L. Cantoni. "A semiotic approach to online communities: belonging, interest and identity in websites' and videogames communities" (2005)
- v J. Preece. "Online communities: design usability, supporting sociability". Introduction. Wiley & Sons (ISBN 0-471-8059998) (2001)
- vi M. Hauben, R. Hauben, T. Truscott. "On the History and Impact of Usenet and the Internet (Perspectives)". Wiley-IEEE Computer Society P. (ISBN 0-8186-7706-6) (1997)
- vii J. Preece. "Online communities: design usability, supporting sociability" Introduction. Wiley & Sons, (ISBN 0-471-8059998) (2001)
- viii C. Herbert. "Using Language". Part II, Pages: 92-95. Cambridge University Press . (ISBN 0521567459) (1996)
- ix J. Short, E. Williams, B. Christie. "The Social Psychology of Telecommunications". London: Wiley & Sons (1976)
- x L. Triacca, D. Bolchini, L. Botturi, A. Inversini. "MiLE: Systematic Usability Evaluation for E-learning Web Applications". ED Media 04, Lugano, Switzerland (2004)
- xi J. Nielsen, R. Mack, "Usability Inspection Methods", Wiley (1994)
- xii P. Kolloch, M. Smith, "The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace", London: Routledge (1999)
- xiii J. Bishop, "Increasing participation in online communities: A framework for human-computer interaction", Computers in Human Behavior 23 (2007), Pages: 1881-1893
- xiv A. 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
- xv 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
- xvi 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)
- xvii 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.
- xviii 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
- xix 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)
- xx 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)
- xxi A. Chin, M. Chignell, "A Social Hypertext Model For Finding Community In Blogs", HT'06, August 22-25, 2006, Odense, Denmark.
- xxii 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
- xxiii T. Yamada, D. Sakano, Y. Yasumara, Kuniaki, "Extraction of Reliable Reputation Information Using Contributor's Stance", Web Intelligence, Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web

- Intelligence table of contents,
Pages: 382-385, ISBN:
0-7695-2747-7, 2006
- xxiv M. Jamali, H. Abolhassani, "Different Aspects of Social Network Analysis", IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings) (WI'06) (2006)
- xxv J.I. Khan, S. Shaikh, "Relationship Algebra for Computing in Social Networks and Social Network Based Applications", Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on Volume, Issue, 18-22 Dec. 2006 Pages: 113 - 116
- xxvi T. Falkowski, J. Bartelheimer, M. Spiliopoulou, "Mining and Visualizing the Evolution of Subgroups in Social Networks", IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings) (WI'06) , pp.52-58, (2006)
- xxvii C.C. Yang, Torbun D. Ng, Jau-Hwang, Wang, C. Wei, H. Chen "Analyzing and Visualizing Gray Web Forum Structure", Intelligence and Security Informatics, Springer Berlin / Heidelberg, Pages: 21-33 (2007)
- xxviii S. Qiao, C. Tang, J. Peng, Hongjian Fan, Y. Xiang, "Mining Virtual Community Core Members Based on Gene Expression Programming", VCCM Mining, Intelligence and Security Informatics, Springer Berlin / Heidelberg, Pages 133-138 (2006)
- xxix S. Choy, A.K. Lui, "Web Collaborative Tagging Systems", IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings) (WI'06), 2006
- xxx E. Lim, B. Vuong, H. W. Lauw, Aixin Sun, "Measuring Qualities of Articles Contributed by Online Communities", Nanyang Technological University Nanyang Avenue, Singapore
- xxxi D. Bezerra, B.L.; de Assis T. Carvalho, F.; Macario Filho, V., "C^2: A Collaborative Recommendation System Based on Modal Symbolic User Profile", Web Intelligence. WI 2006. IEEE/WIC/ACM International Conference on Volume, Issue, 18-22 Dec. 2006 Pages: 673-679
- xxxii S. Lo; C. Lin, "WMR - A Graph-Based Algorithm for Friend Recommendation". Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on Volume, Issue, 18-22 Dec. 2006 Pages: 121-128
- xxxiii Y. Kawachi, S. Yoshii, M. Furukawa. "Labeled Link Analysis for Extracting User Characteristics in E-commerce Activity Network", Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on Volume, Issue, 18-22 Dec. 2006 Pages: 73-80
- xxxiv N. Matsumura, Y. Sasaki. "Understanding Leadership Behavior in Human Influence Network", Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on Volume, Issue, 18-22 Dec. 2006 Pages: 95-102
- xxxv J. Preece. "Online communities: design usability, supporting sociability" Introduction. Wiley & Sons (ISBN 0-471-8059998) (2001)
- xxxvi J. Bishop - "Increasing participation in online communities: a framework for human-computer interaction" (2007)
- xxxvii R. Cheng, J. Vassileva "User and community – adaptive reward mechanism for sustainable online community" (2006)
- xxxviii M. Van Welie, "Welie Interaction Pattern Library" – <http://www.welie.com/patterns>
- xxxix S. Ceri, P. Fraternali, A. Bongio, "Web modeling language (WebML): A modeling language for designing Web sites", Computer Networks 33, 1–6, 137–157. 2000.
- xl M. Brambilla, S. Ceri, P. Fraternali, I. Manolescu "Process Modeling in Web Applications" Politecnico di Milano, Futurs - LRI, PCRI.
- xli J. Preece, "Online Communities - Designing Usability, Supporting Sociability", Wiley, 2000
- xlii Barilis E., Ceri S., Paraboschi S., "Modularization techniques in active rule design", Tech. Report IDEA. WP. 003.01, Politecnico di Milano, Milano, Italy, Dicembre 1994
- xliii P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, "Basi di dati e architetture di evoluzione"
- xliv F. Casati, S. Ceri, B. Pernici, G. Pozzi, "Deriving active rules for workflow enactment", Politecnico di Milano, Milano, Italy, March, 1996
- xlv M. Brambilla, S. Ceri, P.

Fraternali, I. Manolescu, "Process Modeling in Web Applications", ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 4, October 2006, Pages: 360-409.