Reputation Assessment in Collaborative Environments

Università degli Studi dell'Insubria

Dipartimento di Scienze
Teoriche ed Applicate – DiSTA

PhD Thesis of:
Lorenzo Bossi

Supervisor:
Prof. Alberto Trombetta

January 7, 2014

Acknowledgements

I thank my family who supported me day by day, even in the worst ones when I can be very grumpy and intractable. I would not have achieved this goal without them.

A special thank to my advisor, prof. Alberto Trombetta, who led me in this difficult path being always very patient and supportive.

I want to acknowledge also 7pixel society, with a special mention to Nicola Lamberti, because they believe in research and financially supported my PhD.

I am grateful to prof. Tan Kian-Lee who not only gave me a great opportunity during the last year of my PhD, but also a new hope to finish it; and eventually kindly accepted to review this thesis.

In addition, my gratitude goes also to dr. Stefano Braghin, who helped me understanding the PhD lifestyle, made really interesting coffee break debates about coding style and philosophy and pleased me with his friendship.

Last but not least I want to express my gratitude to all the professors, labmates, coworkers and friends who, sharing moments of their life with me, made these years memorable. It would be impossible to list everyone is worth a mention here, so I prefer to close with a simple

Thank you all

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The popularity of open collaboration platforms is strongly related to the popularity of Internet: the growing of the latter (in technology and users) is a spring to the former. With the advent of Web 2.0, not only the Internet users became from passive receiver of published content to active producer of content, but also active reviewers and editors of content.

With the increase of popularity of these platforms, some new interesting problems arise related on how to choose the best one, how to choose the collaborators and how evaluate the quality of the final work.

This evolution has brought much benefit to the Internet community, especially related to the availability of free content, but also gave rise to the problem of how much this content may be trusted. In the pre-digital era, professional reporters were designated to gather information and newspaper redactors to validate, filter and only eventually publish it. This mechanism guarantees – or makes its best to guarantee – the quality of the news reported. But nowadays a lot of news is published on websites that do not have this kind of filter. Overall, this can be seen as a good point since it lets information flow outside places where the "traditional" system is not allowed, such as countries in war or under totalitarianism. The problem is that without this filter at the source of information, the final user that has to make his own filters to discern real information from the manipulated one, deciding which source is trustworthy and which one is not.

Similar problems arise when we have to deal with an online collaborative environment. People cooperate because they are aware that together they can reach better result than if they work one by one. In the pre-Internet era, people looked for collaborators in their friend network, so that they already knew the candidate collaborators. This let them make – usually – good decisions, because they already *trust* the others and know that they can work peacefully and well all together. With the new era of Web 2.0, a lot of websites are born to help people find coworkers, usually each one specialized

in some very narrow field. Starting from LinkedIn[1], which is a social network that let you browse for professionals through the friendship connections; a lot of other websites let you not only look for people but also directly and actively work with them: for example writing code (e.g. GitHub[2], SourceForge[3]); composing music (e.g. Kompoz[4], Digital Musician[5]); doing handmade works (e.g. instructiontables[6], kollabora[7]) and many others. All these websites make the task to find a coworker so much easy that it becomes tricky again: if looking for a particular skill in the friend network can result in a very small group of persons, or even no one, looking for the same skill in a world wide community can result in a huge set of persons, too many to contact one by one to find the best coworker required. A filter is needed in order to reduce this set and this filter must be good in order to present to us only the *best* – often for a very personal definition of "best" – people.

These problems can be addressed using reputation systems, which let us rank people according to their public reputation and so infer from this a personal level of trust we want grant to them.

The purpose of this thesis is to present different reputation systems suitable for collaborative environments; to show that we must use very different techniques to obtain the best from the data we are dealing with and, eventually, to compare reputations systems and recommender systems and show that, under some strict circumstances, they become similar enough and we can just make minor adjustment to one to obtain the other.

1.1 Problem definition

It is quite easy to see that the terms *trust* and *reputation* denote concepts involving several subjective features related to a wide range of human activities and behaviors. As such, framing them into a – more or less – formal definition is far from trivial. Many different approaches have been proposed [30].

Starting from the definitions given by *The Oxford Dictionary of English*, we can read that reputation is:

> *A widespread belief that someone or something has a particular characteristic.*

On the other hand, trust is defined as:

[1] `http://www.linkedin.com/`
[2] `https://github.com/`
[3] `http://sourceforge.net/`
[4] `http://www.kompoz.com/`
[5] `http://www.digitalmusician.net/`
[6] `http://www.instructables.com/`
[7] `http://www.kollabora.com`

> *The firm belief in the reliability, truth, or ability of someone or something.*

The main difference between these definitions is that reputation quantifies a public opinion, while trust is totally a subjective value.

For the purposes of this thesis, we need some stricter definitions in order to describe better the reputation systems. So, for trust we stick with the informal definition given by Jøsang [37], namely:

> *Trust is the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even through negative consequences are possible.*

By using this trust definition, we define reputation as an aggregation of trust from different people.

Another key concept is that both reputation and trust are relative on a particular characteristic of the subject. Therefore, in a teamwork environment, the trust is not related to one person, but to a particular competence of a person. Therefore also reputation – which we have just defined as an aggregation of trust – must be related to the pair of person and skill.

In real world scenarios it is common for a person to have different reputations, which may be totally unrelated. For example consider someone who has a very high reputation as singer but a very low reputation as cook. Differently, if one is a very good singer maybe she/he is also a fairly good musician. This happen because skills are not totally unrelated with each other. Thus, the knowledge of the reputation of someone with respect to a particular skill can be used to infer the reputation about some other (related) skill.

Reputation depends also by the reference group of people. For example, a musician can have a very high reputation between people who share the same taste in music, let's say they like pop music, but a low reputation between people from a different interest, let's say rock.

We must bear in mind all this concepts when we discuss about reputation systems, because if the target system is very narrow and deals with very focused user skills we can omit these distinction and deal just about a single value for a user reputation; but if the system is quite wide and deals with different user competences, we must deal with a set of reputation for each user, otherwise we risk to misevaluate people and give wrong suggestions.

Among all the different kinds of mass collaboration systems [23], this thesis will follow describing two different types of them, classified considering the interactions that users have in the system itself: explicit collaborative environments and implicit collaborative environments.

In the explicit collaborative environments, users look for coworkers in order to achieve a common goal; they have the possibility to choose people to work with and to create – usually – small teams. In this scenario it is

easy evaluate the quality finally achieved by soliciting a direct feedback from the team members, because the quality of the teamwork is subjective and depends on personal experience. We will discuss this collaborative environment in Chapter 3 where we will present two different scenarios: a team formation system and a service composition system.

In the team formation system (Section 3.1.1) we will show how to create a graph of trust to attach to an existent social network. In this graph each user is identified by a set of competences and two users are connected if the first one trusts the second one with respect to a given skill. We will show that it is possible to create an algorithm that computes the *quality* of a team with respect to the personal opinions of each member and we will show that is possible to extend this algorithm to propose teams given a set of skills needed to fulfill a job.

In the service composition system (Section 3.2) we will deal with Internet of Things and cloud computing services by showing how we can adapt the previous network to create a service aggregation system. In this scenario the input of the algorithm is a set of services required and the output is *the best* selection of platform. We use the reputation of the services – which is an aggregation of reliability and mutual compatibility – to choose the best platform.

The perspective will totally change analyzing the implicit collaborative environments in the Chapter 4. In this scenario the users are part of a single big community that shares a common goal which is not well defined. In this case it is impossible to ask directly opinions about other users because (i) although different users can have different visions about how to reach the goal, everyone can contribute somehow to it; (ii) the community is too big and (iii) the teams are not well defined, in the sense that new users frequently join a sub-community, contribute for a while and eventually leave it, without really knowing or choosing other contributors.

We will discuss two case of studies, both related to Wiki systems, and we will see how constraint imposed on the data recorded by the wiki change – and simplify – the analysis and therefore the reputation system.

The first case of study deals with a wiki for semi-structured data. This kind of wiki is not popular yet but – since can be used to solve different open problems (see Section 4.1.1) – it has found a good interest in the research community. We use as reference the model specified in the Section 4.1 and discuss how to attach an attack resistant reputation system in the Section 4.1.13.

The second case of study is focused on a classical wiki that stores plain text pages. In this scenario we use Wikipedia[8] as model and show in Section 4.2 how a statistical model can be used to find the best user among the most active contributors. We show and validate its result using the Ital-

[8] `http://www.wikipedia.org/`

ian Wikipedia, but since our model is language independent and semantic unaware, it works with every textual Wiki.

Chapter 2

State of the art and related work

Reputation evaluation is a relevant task because it can help cope with a lot of issues that people face when dealing with big online communities. Unfortunately, the lack of formal definitions and different scenarios where reputation systems can be applied, results in models and results that are very difficult to compare.

In a widely cited survey [36], Jøsang et al. noticed that, even though reputation systems are already being used in successful commercial online applications and there is a rapidly growing literature around this area, the research activity is not very coherent because authors often propose new systems from scratch, without trying to extend and enhance previous proposals. In this work, the authors recall that trust can be classified in different ways according to its purpose [30]: i.e. identity trust describes the belief that an agent identity is exactly what it claims, and it can be achieved using some authentication schema; delegation trust describes trust in an agent that acts and makes decision on behalf of the relying party and cannot be achieved by any sort of authentication, no matter how complex it is. They classify trust also according to its semantic in a two dimensional space: the first dimension describes that trust can be subjective, when an agent provides a rating based on a subjective judgment, or objective when it is computed by formal criteria; in the second dimension trust can be specific when it measures only an aspect, such as the ability to deliver on time; or – as the opposite – it can be general when it averages different aspects.

Mezzetti proposes a socially inspired reputation model [49] for web services, concluding with a proposal of a trust aware naming service.

Reputation can be used in wireless sensor networks too as a measure of the quality of a node, resulting in a better throughput at the price of a little more energy consumption [50].

A problem often faced when dealing with reputation systems is how to

aggregate feedback from different sources. In [27] the authors show that a simple arithmetic mean is not always the best choice because it is a good indicator only when ratings are unbiased and normally distributed. Unfortunately, this is not a normal scenario in real world data. They show that with real world data the weighted mean is in general more informative, while the median is more robust.

The importance of reputation in community is highlighted in [45], where the authors propose also an abstract model for it. In their simulations the authors show that there is a need for collaborative systems that allow large group of professionals to make decisions better than single individuals.

A good overview of different reputation systems can be found in [26], where the authors discuss the importance – and the difficulty – of making a manipulation resistant reputation system. The idea is that if malicious users can alter the results provided by a reputation system, the system itself becomes totally useless since no one will trust its results anymore.

Because of this big variety of models and proposals, in this thesis we will focus mainly on the related work closer to the topic we will discuss in the next chapters: reputation in team formation and in wiki environments.

2.1 Reputation in team formation

As it is relatively easy to track the behavior of users of an online social network and – more in general – structured communities, such information can help in forming teams and performing/solving a given set of tasks.

A framework similar to what we will discuss in Section 3.1.1 is presented in [42], where the authors model a system to manage users, tasks and skills. The compatibilities among users are inferred from an underlying social network. The social network is represented as an undirected, edge-weighted graph, where the weights denote how well the corresponding users may collaborate. Thus, the problem is to find, given a task t, a subgraph of the social network such that every skill needed for accomplishing t is possessed by at least one node in the subgraph. It is shown that this problem is NP-Hard and accordingly, approximation algorithms that perform well in real-world datasets are given. However, similarities among enumerated skills are ignored in that approach. Likewise, past interactions among users in previously formed teams are overlooked.

In [14], a similar framework (tasks, skills, etc.) is taken into account but here the problem is how to optimally allocate the users' skills to cope with multiple, concurrent tasks. As in the previous work, the allocation of skills takes into account the social interactions among users, as recorded by a social network. The model presented in this work is utility-based. That is, given a set of concurrent tasks that require certain skills, the aim is to find a skill allocation that maximizes the probability of successfully completing

the highest number of tasks. Thus, the problem is orthogonal to what we study here. Furthermore, again the effect of interactions among users being involved in past teamwork is not taken into account.

The problem of how to define and compute trust in an online community is a well-known and thoroughly researched problem. It has been analyzed and modeled with very different approaches. One of them consists of computing transitive closure of trust in a single dimensional model using subjective logic [35]. Other approaches are probabilistic, like in [74] where authors propose a model to compute trust of generic provider agents; while in [60] authors develop a probabilistic multidimensional trust model focused on how trust can be propagated in a real-world scenario of a supply chain market. In [61] authors compute trust in a multidimensional network where agents can estimate the utility of a contract by computing the probability that each dimension of a contract can be fulfilled. They also propose a protocol to propagate rumors and compute trust in a totally distributed network.

A similar scenario, where a multidimensional network is used to automatically make recommendations to users, is described in [40]. In this work the authors use a multidimensional social network on a Flickr dataset to suggest potentially interesting users. They create a multidimensional graph where every node is a user and edges on a single dimension represent some kind of relation between users (i.e. both commented the same photo). Finally they aggregate information from all the dimensions to create a list of *similar users* to suggest. User's feedback about suggestions is used to correct the weight used to aggregate the different dimensions.

A method to propagate social trust through friendship relations is presented in [52]. The purpose of this work is to compute the popularity of the users in the system. The authors show that their approach can be used to model the *friendship effects* in a social network (i.e. a person followed by a popular person is likely to have more followers).

Pizzato et al. describe, in various works [56, 57, 58], the people-to-people class of recommender systems using the case of online dating. They focus on the fact that this kind of recommender systems differ from the traditional items-to-people recommenders as they must satisfy both parties and they call this type of recommenders *reciprocal*. They also notice that, in such kind of recommenders, the cost of a poor recommendation can be quite high because users are more afraid of continuous rejects from other people than wrong suggestions on purchasable items.

In [47] Masthoff discusses how a system can aggregate recommender information from individual users to make a recommendation for a group of users (i.e. suggest which movie to watch among certain friends). For multidisciplinary team recommendation, coverage of the necessary skills as well as collective compatibility among the members of the team are essential, and this work looks at these issues that are typically studied in isolation in

a holistic manner.

2.2 Reputation in service composition

Internet of Things (IoT) is a novel paradigm that is receiving an increasing interest from the research community [3, 64, 69]: the increasing number of *things* that are able to communicate wireless (RFID and NFC sensors, mobile phones, wearable computers, etc.) leads to the idea that these things – communicating together – could reach some common goal.

It is already possible to see some preview of this technology: for example smart phones are already capable of discovering open networks and it is possible to find NFC tags used as smart label for contact less payment. But the base idea of IoT is that interactions must be easier and wider. In order to reach this goal a lot of effort is made in developing standard protocols to discover services and provide communications among different devices [32, 67].

Standard service discovery protocols are already widely used: for example Simple Service Discovery Protocol (SSDP) – a part of Universal Plug and Play (UPnP) protocol[1] – is designed mainly for residential networks to let PCs, printers and routers seamlessly discover each other and configure themselves for data sharing.

Research in service composition is focused on developing richer languages to describe services: for example in [7] authors analyze composing semantic web services using a constructive description logic; in [10] authors proposed a language to define security constraints to check during the phase of service discovery and filter the services required in the service composition phase. A similar work is proposed in [20] where an ontology-based annotation is used to define security constraints using the DAML-S protocol.

To the best of our knowledge, one of the first works that proposed a reputation model for service composition is presented in [43], where the authors described a model that uses an ontology to enforce similarity checks between services, but computes the reputation – in a quite naive way – as the average of users' vote in a given context (i.e. the application domain).

In [73] a model is presented to evaluate QoS in service selection. Each service is ranked by different non-functional properties – in their example: (i) execution price, (ii) execution duration, (iii) reputation, (iv) reliability and (v) availability – in order to select the overall best service. Also in this work, reputation is defined as the average ranking given to the service by end users.

[1] `http://www.upnp.org/`

2.3 Reputation in Wiki environments

The concept of wiki – intended as an online platform for open collaboration that allows people to add, modify or delete information, in the form of text – is a relatively old idea. The first wiki was developed in 1995 by Ward Cunningham [29], but the popularity of this approach grew in 2001 with the launch of the Wikipedia project that, nowadays, ranks in the top-ten most-visited websites worldwide.

The wiki model, that lets everyone edit its contents, naturally poses a lot of questions about how reliable such contents are. Wikis work because volunteers collaborate adding information, and one of the fundamental principles on Wikipedia is the assumption that editors' edits and comments are made in good faith. Nevertheless wiki information can be wrong in many different ways: someone can write wrong content but in good faith, someone else writes to support his/her own point of view against all the others, eventually there are also just vandals who like to simply destroy content. Moreover, wiki information is written in plain text with all the issues that it implies: it is possible to find good information so poorly written to be difficult to read and understand, or very bad information written so well that seems plausible and all the different shading between these two extremes. In general it is very difficult to discern the textual presentation from the underlying information.

Several studies had addressed these problems; one of the most cited was published in 2005 in Nature [28], where the authors assert that scientific articles from Wikipedia came close to the level of accuracy in Encyclopædia Britannica and have a similar rate of "serious errors".

Since this work, a lot of research was made to evaluate both Wikipedia content or users – often called Wikipedians. As we will discuss further in Section 5.1, the difference between evaluating content and evaluating users who produce it is very scarce.

We can group the approaches used to evaluate Wikipedia roughly in three different big categories:

1. text survival, where the underlying idea is that textual information that remains unchanged for long time is better than text that is edited very fast;

2. information visualization, where information like text similarity or edit history is organized in a graph to identify recurring patterns that can discriminate good versus bad content;

3. machine learning techniques, where Wikipedia categories like Featured

articles[2], Stubs[3] or Disputed[4] are used to create a ground truth of good and bad pages that is used to train a classifier with features extracted from the pages themselves.

In [2] Alder and Alfaro present a reputation system based on text survival. They evaluated their system using Italian and French Wikipedias showing that their system shows a good predictive value because changes made by low-reputation authors have a larger probability of being undone compared to high-reputation authors' changes.

In [59] the authors introduce the impact of an edit as a measure of how many views it has. The idea is that edits on most viewed pages have a higher impact than edits on less viewed pages. Their proposed model uses a text survival approach but, instead of measuring just the time that a text survives, they measure the text life as number of its views. They concluded the work proposing a classification of damages that users can make.

WikiTrust [19] is an extension for Firefox browser which colors text of Wikipedia based on its reputation. Text gains reputation if it is revised by high-reputation authors, and authors gain reputation if their text survives for subsequent edits.

In general, one common problem of the text survival approach is that it is difficult to track when a text is moved from one page to another, because this information does not emerge from any log and there are too many pages to search for common text among them. Unfortunately is quite common in Wikipedia to split a page, when it becomes too big, in smaller ones or to move text from pages to other more appropriate ones.

In [62], Sabel develops a tool which displays revisions of a Wikipedia page in a tree based on the similarity of the text. By analyzing the ramification of the tree it is possible to recognize some visual patterns typical of edit wars and vandalism acts. Sabel shows that good articles usually have a linear growth.

In [72] authors propose a dynamic Bayesian network to compute article trust using as features the author trust and the amount of insertion and deletion for each article revision.

In [65], Swarts discuss about the collaborative construction of information – that is called "facts" in the paper – on Wikipedia. Starting from the premise that knowledge is not fixed, he shows how people collaborate in Wikipedia to produce articles. Swarts does not focus his work on the quality of the facts, but in the process that makes them emerge, grow and stabilize in accepted information by the community.

[2] http://en.wikipedia.org/w/index.php?title=Wikipedia:Featured_articles&oldid=576658176

[3] http://en.wikipedia.org/w/index.php?title=Wikipedia:WikiProject_Stub_sorting/Stub_types&oldid=575430035

[4] http://en.wikipedia.org/w/index.php?title=Template:Disputed&oldid=546562025

Another analysis about team collaboration in Wikipedia is made in [70], where the authors created a social network connecting Wikipedia coauthors and showed that also distrust can be beneficial to the articles' quality, when distrust is accompanied by a high activity on the talk pages.

A deep analysis of conflicts in Wikipedia is made in [71], where the authors analyze how edit wars are made and, eventually, conclude that usually they are fought by few editors only.

A sort of reputation systems focused on Wikipedia administrators is shown in [17], where the authors develop a way to quantify the suspicious activity of an administrator measuring his focus on controversial topics. The idea is that Wikipedia administrators have great power because they can block the edit on controversial pages or even block authors, so an administrator that act maliciously can modify a hot topic and mold public opinion.

2.3.1 Semi-structured Wiki

One of the main purposes of a reputation system in a Wiki is to stimulate the community to contribute more and better in increasing the shared knowledge preserved by the Wiki.

In this scenario the reputation system becomes a sort of game with a purpose where, through a process of gamification, the reputation score is seen by the users as an index that they need to increase to show their engagement in the community [33, 38].

On the other hand, the purpose to collect better data in the wiki system can also be achieved by offering better tools to store, edit and validate the data. Traditional wiki systems work very well with plain text but have a lack of support to structured data. Unfortunately, as is it well known, most information is better represented by structured – or, possibly, semi-structured – data rather than plain text or wiki markup language.

Classical wiki systems handle structured data through infoboxes[5]. An infobox is a template page which is designed to be included in other pages, pages can pass variables to the template, so infoboxes are used to display similar information sharing the same layout in different pages. The problem with infoboxes is that they do not support any kind of constraints to validate the corresponding data, nor a system to evolve automatically data with respect to a new version of the template. All this work is made manually by the community, but this leads to some inconsistency in the presentation of the data, especially in the less visited pages. As such, different projects have been proposed in order to cope with this issue.

To the best of our knowledge, the most mature work closest to what is discussed in Section 4.1 is present in [9] in which the authors describe a pro-

[5] `http://en.wikipedia.org/w/index.php?title=Help:Infobox&oldid=455091951`

totype for a wiki for structured data. The main difference with our proposal is that they manage only one – usually fairly big – XML document. It turns out that the considered schemas are simpler than ours: for example, they do not allow specification of the datatype of the tree structure. Rather, the authors focus on query language issues. They are developing a powerful query language which enables selecting only a fragment of the XML queries based on various constraints which can involve also annotation on nodes. Finally they support only two types of schema updates: (i) insertion that happens automatically when inserting a data which require a schema extension and (ii) deletion which delete also the corresponding data subtree.

A very early prototype is Data Wiki[6]. Developed at Google Labs and now moved to Appspot, it lets users to add, modify and delete tuples from a table.

Other works deal only with the schema evolution. We think that the most important contribution for our context is presented in [41] in which the authors propose a conceptual model for XML Schema evolution. They use a graphical environment to define schemas and schema update. Then some normalizations are performed on updates to minimize the updates themselves. After schema update, they recheck document validity and perform a document update. But there's no way to update node values on documents. Another interesting work is [31], where the authors define a set of update primitives for XML Schema. They study which evolution primitives are known not to compromise documents validity. Then, they use a labeling process to keep track of the document portions whose validity might have been compromised so they can revalidate only subtree to speedup the process. Their approach to documents' evolution consists on the detection of the minimal modifications required to make the documents valid for the evolved schema. But only document structure can evolve, not document data.

Since semi-structured wikis are still in a stage of research and – with some minor exception – not used by the public, it is difficult to find works that describe a reputation system for this environment. To the best of our knowledge, the only one is described in [25] where their primary scope was to develop a recommender system for any kind of data. To achieve this, they modify a wiki to manage structured data (i.e. movies and restaurant). The problem is that they need to develop a new module for every schema.

We will discuss reputation systems for semi-structured wikis in Section 4.1.13.

[6] `http://datawiki.appspot.com/`

Chapter 3

Reputation in explicit collaborative environments

It is a well known fact that cooperation is the way to achieve the most ambitious goals. Cooperation is the key to put different competences and qualities together to achieve a common purpose, something that cannot be done by only one person or using only one service.

The problem is that with the advent of Web 2.0, both the resources – people and services – are so overabundant that it becomes difficult to find the best ones to cooperate. The risk is that when there are too many alternatives, eventually it is like have none of them; because analyzing all of them becomes too difficult and makes it almost impossible to have a wise choice.

Usually people ask for some recommendation to friends in order to have some suggestions and orientate within so many possibilities; the problem is that sometimes we do not have friends with the specific competence or knowledge to give us the recommendation we request. In this case a recommender system can solve the problem: with an automatic system it is easier to aggregate large amounts of data and use this widespread knowledge to make a personalized recommendation. In teamwork and explicit collaborative scenarios it is important to have tailored recommendation, since everyone is looking for something different: the same service can have different reviews according to the expectations of the user, also a coworker can be seen in a very different way according to temperament affinity.

We recall the definition of *trust* and *reputation* from Section 1.1, where we defined trust as a personal belief that someone or something will act as expected, while reputation is taken as an aggregation of trust since reputation represents a public opinion. In this chapter we will also define reputation in a slightly more precise way as the transitive closure of trust. We will use also the word *recommendation* as the relation that is symmetric of the trust relation (see Image 3.1).

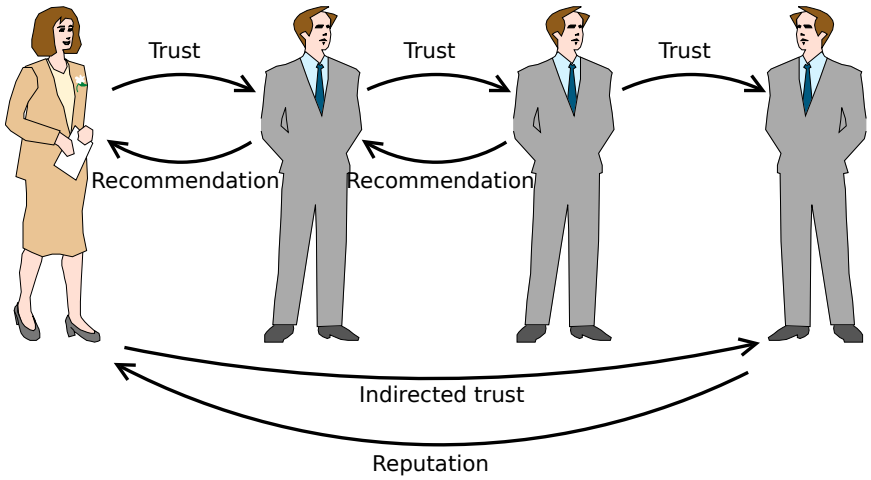


Figure 3.1: Recommendation is the relation symmetric of trust, the transitive closure of trust is considered a reputation

In our approach, reputation is computed locally, so – in principle – every user may compute a different reputation based on his/her previous personal experiences with other users (asymmetry). Informally, a user is encouraged to report truthful information about his/her previous interactions because every user privately computes the reputation of other users. Hence, misreporting information has consequences only on his/her own reputation computation. In order to compute the reputation of a user from the point of view of another, in the absence of direct trust information, the system infers it by searching for a trust path between the two users (trust transitivity).

The core idea of this work is to define algorithms that maximize the overall internal reputation for teams or services. It is important to understand that this process can not be simplified as selecting the entities with highest reputation. Consider the example given in Figure 3.2): someone needs to perform a certain task, s/he worked successfully in the past with both Cat and Dog and they are proficient in the skills required to accomplish the said task. Nevertheless, a team with both Cat and Dog as members is not optimal – thus, the team does not reach its fully potential nor maximizes the team reputation value – because such users do not work well together. It is better if the team master choose Canary instead of Cat even though its reputation is lower, because the overall team reputation will be higher.

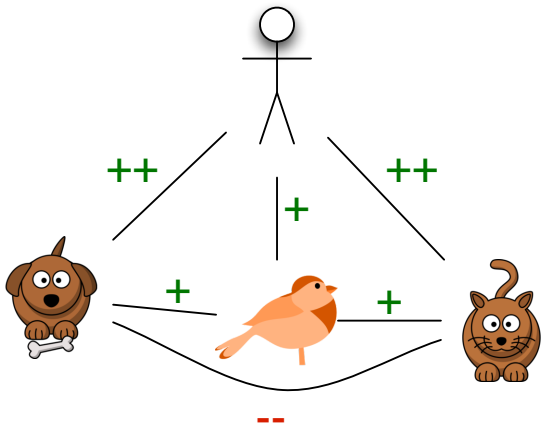


Figure 3.2: The reputation of a team is not just the sum of its members' reputation

3.1 Case of study: team formation system

Often one needs to form teams in order to perform a complex collaborative task. Therefore, it is interesting and useful to assess how well constituents of a team have performed in the past, and leverage this knowledge to guide future team formation.

With the advent of Web 2.0, social networking and crowd-sourcing technologies, there is an increasing trend of collaborative teamwork in online environments. This is naturally receiving an ever-growing academic attention as well [14, 60, 42].

Previous work (as seen in Section 2.3.1) focuses on how to find an optimal solution for the allocation of users for accomplishing the given tasks, taking into consideration how well the users are mutually compatible. Compatibility among users may be extracted from external sources, like interaction graphs in online social networks. User skills are often determined by the system by *fiat* and in a static, immutable manner. More precisely, a set of users – each possessing some skills – and a set of tasks, that need certain skills to be accomplished are assumed a priori. Then, upon requiring that a certain set of tasks has to be performed (from now on, for simplicity we consider the case in which there is a single task to be performed), the "best" set of users having the required skills is provided, thus forming the best suited team for the task.

In this section, we present a framework for team formation in which the assessment on how good a user is in a given skill is determined dynamically, depending on, among other things, the judgments or feedback ratings that the user had obtained when performing tasks requiring similar skills. Naturally, taking into account users' ratings lends itself to a system dealing with the reputations of the users themselves. That is, users – by giving (and receiving) ratings about the work other users have done (and about the work they have done) – form a *web of trust* such that (i) two users are directly connected if one rated the work done by the other; (ii) the overall reputation of a user (with respect to a given skill) is computed as an aggregate value of the ratings (relative to such skill) given by other users; (iii) the perception of the reputation of user u from the point of view of another user v depends on the reputations and ratings of the users connecting (possibly by multiple hops over the web of trust) users u and v.

An important aspect of the presented work is that we take into account how skills are related among them. In other words, we consider the skills to be similar (in varying degrees) and this contributes in determining who are the best team members that can cope with the task to be performed. The computation of reputation takes into account skills' similarities thanks to the "multi-dimensionality" of the web of trust, that takes into account users' ratings for every skill, along with information about skills' similarities. We next outline and discuss some relevant points concerning how reputation

may be computed and used in assembling the best possible team for a given task.

In this work we propose a model for assessing the reputation of participants in collaborative teams. The model takes into account several features such as the different skills that a participant has and the feedback of team participants on her/his previous works. We validate our model based on synthetic datasets extrapolated from real-life scenarios.

3.1.1 Reputation in a team formation setting

In real-world settings it is fairly common to have different reputations, depending on the different skills that are being evaluated. Furthermore, such reputations may be correlated, given that the corresponding skills are similar and can be different according to the reference group (see the example in Section 3.1.2). We try to capture this natural phenomenon in our model. Once a team has been formed, team members may rate each other about the work done. We assume that every team member sees everything that has been done by every other team member. While performing such ratings, they add to the reputation system new information regarding the trust they have about particular skills of other users. Features that should be rated may vary from how well a team member collaborates with others or his/her effective skill on a given topic. In this fashion, the reputation of a user is determined not only on the basis as assessed by the team recommendation system (an *ex-ante* process which derives information by graph mining and information retrieval mechanisms [14, 8, 42, 39]), but also on the ex-post judgment of one's peers. Doing this operation they add in the reputation system new information about the trust they have about particular skills of others.

The *ex-ante* rating process of a user may include the judgment about him/herself or the judgment of another high-reputation user. Note that such judgment is given before the team recommendation process has occurred. The last point is particularly relevant in assessing the *ex-ante* reputation of unregistered users, since they are allowed by the team recommendation system. Thus, the overall reputation is taken into account in future team recommendations.

3.1.2 Motivating example

In order to show how such concepts may be related to finding the right people for performing a given task, we consider the following scenario. Alice is a singer who wants to find people capable of composing songs (that is, their music and lyrics) and play them as well. Since Alice is not good in composing, she looks for help from Carl, a composer she knows. In doing so, she is depending on him and implicitly taking the associated risks of failure

because she trusts him.

Alice also looks for a guitarist, but she does not know anyone with such skill. Thus, she asks some of her friends for a recommendation, with the goal to find some good – that is, with a high reputation – guitarist. From the collected suggestions, Alice has to choose only one person. In doing this, Alice takes into account how much she trusts the friend that made the suggestion and how much such friend trusts as guitarist the suggested person. It is not sufficient for Alice to trust her friend on a general ground, rather it is essential that Alice trusts him/her as an expert in rating musicians. It is thus very important that both reputation and trust are relative about a particular skill (as suggested in [53]). In real-world scenarios it is common for a person to have different reputations, which may show a degree of correlation among themselves.

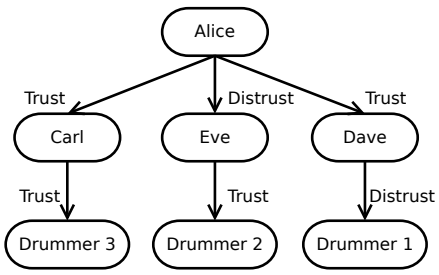
That is, Alice does not trust any recommendation of her musician friends, but she trusts the recommendation of another friend of her, who is very knowledgeable in music trends. This may succeed because skills are not totally unrelated among each other. Thus, the knowledge of the reputation of someone with respect to a particular skill (rating music) can be used to infer the reputation about some other related skill (rating a guitarist).

Asking for a recommendation is thus based on a sort of transitive closure of trust. But, generally speaking, trust cannot be assumed to be a transitive relation [15] unless consider further constraints [37] are specified. In our approach, we explicitly pose such constraints by introducing a similarity function over a given set of skills. Skills are not considered as independent from one another, but we take into account the fact that if one possesses a given skill then it is highly probable that she/has some proficiency in another, related one. We use such skill similarity in order to build transitive trust relationships among users that do not have direct connections. As a consequence (as we will explain formally in Section 3.1.3) one of the more relevant features of our approach is that a user may have different reputations about a given skill, depending on which user is asking for it. In literature, such feature has been referred to as *asymmetry* of reputation and plays a fundamental role in guaranteeing the trustworthiness of users' ratings, as explained in [54]. The fact that the users' reputations are computed in an asymmetric way takes into account that one's reputation in a given skill s depends on – of course – on the expertise she/he has in skill s, but also on the expertise that other raters have in skills related to s.

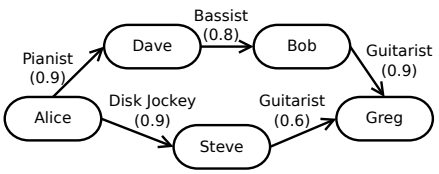
Furthermore, consider that Alice needs a drummer, too. None of her friends knows such a musician, and she looks for recommendations from friends of her friends. This way of proceeding can be iterated and if everyone in the sequence of recommendations is trustworthy, Alice accepts these recommendations by means of a *chain of trust*, that is a directed graph in which the node relative to person A is connected with a directed edge to the node relative to person B if A trusts B.

We present in an informal way how reputation is managed in our approach by explaining a possible way for Alice to decide over three candidates, as shown in Figure 3.3a. The first one is considered untrustworthy by a trustworthy person (Dave) and Alice decides against this candidate; the second one is recommended by Eve and – since Alice does not consider Eve trustworthy – she decides against this candidate as well; Alice finally accepts the recommendation from Carl because refers to a trustworthy person recommended only by trustworthy persons. Furthermore, an important issue is how to deal with different – possibly discording – recommendations. This happens fairly often in real-world situations as well: people may choose to trust more the ratings coming from persons "closer" to them (that is, chains of trust involving the minimum number of persons); or they may opt for ratings from the persons with highest reputation (disregarding the "distance" from such persons). As we will explain in the model (Section 3.1.3), we aim at finding a chain of trust that minimizes its length while maximizing the reputations of the recommenders included in it. Hence, in the example in Figure 3.3b, if Alice asks for the reputation of Greg in playing guitar, she has two different paths. We remark that in the computation of the reputation, the similarity of the skills are taken into account by adding a weight on an edge connecting two persons (the higher the weight, the less similar the relative skills)

The shortest one (from Steve) has the first edge labeled with a skill very different from *playing guitar*, so its weight is very high. For this reason, the longer path weighs less than the previous one and it is used to compute the reputation.



(a) Alice prefers the last drummer because is a trustworthy person recommended by a person she trust (Carl)



(b) To compute the reputation of the guitarist Greg as seen by Alice, we use the longer path because edges are labeled by skills more similar to guitarist

Figure 3.3: Examples of transitive closure of trust

3.1.3 The model

We now set up the model in which we define in a precise way the concepts sketched so far. We assume a finite set $U = \{u_1, \ldots, u_n\}$ of users, that will

be grouped in teams. We also assume a finite set $S = \{s_1, s_2, \ldots, s_k\}$ of skills which users are apt to perform (i.e. playing an acoustic guitar, bass or violin). Skills may be related, and we introduce a *similarity* function

$$l : S \times S \to [0, 1]$$

to take into account such relationships. We write $l(s, s')$ to denote the probability that given the presence of skill s it is likely that s' is present as well. For example, over a set of music-related skills, one may assume that if a user is skilled in playing the acoustic guitar, she/he should be more likely able play the electric guitar than the bass guitar. On the other hand, a user proficient into playing acoustic guitar may not be able to play trumpet, in general.

Furthermore, we assume that the similarity function l satisfies the following (rather intuitive) properties:

1. $\forall s \in S$, $l(s, s) = 1$, which means that every skill is semantically similar to itself, and that
2. $\forall s, s' \in S$, $l(s, s') = l(s', s)$, which means that a skill s is similar to another skill s' as much as s' is similar to s.

We do not assume that l holds a transitivity-like or triangular-like properties such that $\forall s, s', s'' \in S$, if $l(s, s') = x$ and $l(s', s'') = y$ then $l(s, s'') \simeq x$ (or y) or $\leq x + y$. In fact, given the similarity values between s and s' and the one between s' and s'', nothing can be inferred about the existing relationship between s and s'' (as shown in Figure 3.4). The exact definition of the similarity function depends of the problem instance.

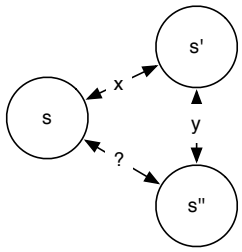


Figure 3.4: Similarity relationships between skills

In our model, both users and tasks are characterized by a set of skills. The user's skill set identifies her/his competences, while the skill set of a task denotes the competences required to accomplish it. Note that a user may participate in solving different, concurrent tasks – contributing with at least one of her/his skills – and a task may be accomplished by different teams of users having the required skills.

As introduced in Section 3.1.1, the goal of the presented work is to find the best team to successfully accomplish a given task t characterized

by a set of skills $S_t = \{s_1, \ldots, s_m\}$. To do this, we take advantage of ratings assigned to each user from her/his past team co-members. Thus, the proposed model does not only leverage on skills' competence as declared by a user, but on such competences as defined by other users according to their past experiences interacting with her/him. In such a way, ratings may help in building teams that aim to maximize productivity and collaborative efforts. The ratings associated with a single user u_f coming from user u_i are the values of a function r defined below, which depend on a given skill s, as well.

The reputation function r is defined as

$$r : U \times U \times S \to [-1, 1]$$

where $r(u_i, u_f, s)$ denotes the reputation that user u_f has on the skill s as assessed by the user u_i. More precisely, the reputation value is 0 if u_f has a neutral competence in the skill s – according to u_i –, 1 if u_f is considered very competent in s and -1 if u_f is considered not competent with respect to s.

The reputation function r induces a weighted, labeled, directed multigraph $\mathcal{G}$ over the set U of users: two users u_i and u_f are connected by a directed edge labeled with s and weighted with the value x if (and only if) the reputation function is such that $r(u_i, u_f, s) = x$.

In general, when dealing with large online communities, we may assume that the graph $\mathcal{G}$ is not completely connected, which means that it is unlikely that all users directly know each other. Hence, we do not have all the possible reputations for all the possible users over all the existing skills. Moreover, we may assume that directly connected users have common interests, that is, they may possess similar skill sets [63].

Based on such assumptions, we propose an algorithm for computing the reputation on a skill s, between two users not necessarily directly connected in $\mathcal{G}$ by an edge labeled with skill s.

This is achieved by searching the shortest path $SP(u_i, u_f)$ in $\mathcal{G}$ between u_i and u_f such that its edges satisfy the following condition: given a skill $s \in S$ and an edge $e \in E$, the weight of the edge e with respect to s, denoted by $w_s(e)$ is computed as:

$$w_s(e) = \frac{1}{l(s, s(e)) \cdot (2 + r(e))} \tag{3.1}$$

Informally, Formula 3.1 assigns higher weights to edges with skills not similar to the requested one (when they are totally different the similarity function is null, so the weight is infinite) and with low reputation. In this way, the algorithm yields a shortest path having edges with highest reputation labeled with skills more similar to the requested one. Furthermore, we require that $\forall e \in SP_s(u_i, u_f) r(e) \geq 0$, unless u_f is the head of the edge e.

We introduce this constraint because in a chain of trust – recall its definition in Section 3.1.2 – one does not want rankings from untrusted sources. On the other hand, the last edge may be negative because, in such a case, we trust a negative opinion expressed by a trusted source. We refer to such a path between two nodes as the *admissible path*.

Having defined what is an admissible path between two nodes in $\mathcal{G}$, we now proceed to describe how to compute the reputation between them with respect to a skill $s \in S$. We define an *extended* reputation function

$$r_s^* : U \times U \to [-1, 1]$$

that takes as input a pair of users u_i, u_f and measures the reputation that u_i has of u_f about skill s. The computation of the extended reputation function is based on the reputation values defined, by the different users, on the edges composing an admissible path between u_i and u_f. It is described in Algorithm 1 and is the minimum value of the reputation multiplied the similarity between the skills found on the admissible path.

Data: The skill s to search for the reputation
Input: The admissible path $SP_s(u_i, u_f)$ between two users
Output: The extended reputation function $r_s^*(u_i, u_f)$
begin
  $rep = null$;
  for $e \in SP_s(u_i, u_f)$ **do**
    $rep = min(rep, l(s_e, s) \cdot r_e)$;
  end
  if $rep == null$ **then**
    return 0;
  else
    return rep;
  end
end

Algorithm 1: Computation of the extended reputation function

We have now the tools required to compute suggested teams for a given task t. We propose two possible scenarios. At first we consider that the user asking for the task t wants to be a member of the team which will accomplish t. Later on, we generalize the approach assuming a scenario in which who commissioned the task is not going to be involved in it.

Reminding that a task t is defined over a set of skills $s_t = \{s_1, \ldots, s_n\}$, our objective is to find a set (or, team) of users $T \subseteq U$ such that:

1. all the skills required to accomplish the task are provided by the team members, which means that $s_t \subseteq \left\{\bigcup_{u \in T} s_u\right\}$;

2. each team member contribute to the fulfillment of the task with at least a skill, which means that $\forall u \in T,\ s_u \cap s_t \neq \varnothing$;

3. the size of the team is the smallest possible, in the sense that there are no redundant members;

4. the reputation of the team is maximized.

Based on the function r^* previously described, we define the reputation of a team $T = \{u_1, u_2, \ldots, u_n\}$ as the average of the reputation of all its members. More formally

$$r(T) = \frac{\sum_{u \in T} \sum_{v \in T \setminus \{u\}} r^*_{s_v}(u, v)}{|T|(|T| - 1)} \tag{3.2}$$

In the first scenario, in which we remind the user u who asked for the resolution of the task is also a member of the team, the algorithm first selects the "best" skill for the existing user. This is done choosing the skill with the highest reputation score according to the edges having u as end node. After this, the algorithm searches for the best user for each remaining skill. This is achieved exploring the shortest paths, rooted in u, searching for the best team member for each skill (see Algorithm 2).

The generalized version of the proposed algorithm, which is suitable for the second scenario where a customer searches for a team, is shown in Algorithm 3. In this version, the algorithm searches for the best team in order to complete the required task according to the available reputation values.

Following the previous example, suppose that someone wishes to compose a simple jingle. Suppose further that this task can be accomplished by two persons, say Alice and Bob or Alice and Carl. Alice is a singer while Bob and Carl are guitarists. Alice never worked before with Bob nor with Carl, so she has not an opinion about them ($r(Alice, Bob, guitar) = r(Alice, Carl, guitar) = 0$). But she worked with Dave who worked both with Bob and Carl. Alice has a strong reputation on Dave, who plays bass, so she trusts him and, since bass and guitar are quite similar, she trusts his opinion more than Eve's one who is saxophonist. In this case, to compute $r^*_{guitar}(Alice, Bob)$ we can choose two different path. We prefer Dave's opinion for Formula 3.1, because Dave's skill is more similar to what we need to estimate than Eve's one. While the definition of *admissible path* is used to enforce the constraint that we need a chain of trust, not of distrust: suppose that there is also Ted, who worked with everyone. He is a guitarist, so his opinion should be the strongest one to evaluate Bob and Carl. But Alice has a very bad reputation of Ted, so she does not trust him and does not want his opinion infers her choice (recall Figure 3.3).

Data: A map *userskill* between skills and a list of users who possess that skill
Input: A task $t = \{s_1, \ldots, s_n\}$, an initial user $u \in U$, a searching depth limit d
Output: A team T or $\perp$ if such team can not be found within depth d

```
begin
    su = select_best_skill(s_u ∩ s_t, u);
    userskill = ∅;
    for each user u' distant at most d from u do
        for each skill s' of u' do
            if s' ∈ s_t then
                userskill[s'].add(u');
            end
        end
    end
    if userskill does not contain at least a user for each skill in
    s_t \ su then
        Failed to find a suitable candidate for some task skill;
        return ⊥;
    end
    for every team T in userskill do
        compute r(T ∪ u) and return the team with maximum
        reputation
    end
end
```

Algorithm 2: Team selection given a user (egocentric team)

3.1.4 Complexity

The core of the algorithms consists in finding the admissible path between users. Since the admissible path is the shortest path with respect of the weight Function 3.1, we can use Dijkstra's shortest path algorithm to find it. Dijkstra's algorithm finds the shortest path between two users in time $O(|U|^2)$ (where U is the set of users). Computing the extended reputation requires a fixed number of mathematical operations for each edge of the admissible path (see Algorithm 1), so we can consider the complexity of the extended reputation function to be the same as Dijkstra's shortest path.

The reputation of a team consists of the average of the reputation between each pair of members (see Function 3.2). Hence, for a team T, we compute $|T| \cdot (|T| - 1)$ reputations.

In the worst case to compute the reputation of one team requires $O(|T|) \cdot O(|U|^2)$ steps. Given the facts that the size of the team is significantly

Data:
Input: A task $t = \{s_1, \ldots, s_n\}$, a searching depth limit d
Output: A team T or $\perp$ if such team can not be found within depth d

```
begin
    T = ∅;
    for u ∈ usersdistantatmostdfromthecustomer do
        T' = select_egocentric_team(t, u, d);
        if T' ≠ ⊥ then
            if r(T) < r(T') then
                T = T';
            end
        end
    end
    if T ≠ ∅ then
        return T;
    else
        return ⊥;
    end
end
```

Algorithm 3: Generalized team selection

smaller than the total number of users and that it is upper bounded by the number of skills, we can consider $O(|T|)$ limited by a constant. Therefore the final time complexity of the algorithm is $O(|U|^2)$.

3.1.5 Experimental results

In what follows we present the experimental results of the proposed approach. To the best of our knowledge there are no public datasets on team collaboration. Therefore, in order to evaluate the proposed approach, we needed to generate test data.

The prototype

In order to create the dataset and test our algorithms we developed a Java prototype which uses the JUNG[1] library to store and manage the user graph. We used JUNG's implementation of Dijkstra's shortest path algorithm to find the path between to users used to compute their reputation. Because this is the most time consuming operation, it's very important to implement a cache to store this results (the pair of users, the skill and the computed reputation) in order to speed up our proposed team formation algorithms.

[1] `http://jung.sourceforge.net/`

Hence, we implemented a caching mechanism to improve the overall performances.

Dataset generation

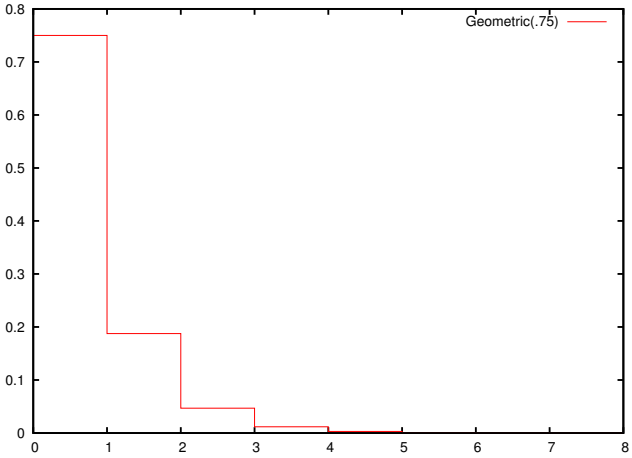
The dataset used in the evaluation of the proposed algorithms have been generated in such a way that they respect the properties of real world social network graphs. Thus, to be comparable with real data. In particular, many studies show that complex networks' vertex connectivity follows a power law distribution, because new vertices connect preferentially with already well connected ones [1, 4]. Thus, to make a dataset suitable for our purposes, we create a graph of users where the edges – representing the past interactions between users – follow a power law distribution. To obtain such a graph, we take advantage of a well known social graph generation algorithm from Eppstein [24] (see [11] for a more comprehensive discussion on graph generation algorithms). We tweak the algorithm so that the generated graphs resemble the Extended Epinions dataset [46] because, to the best of our knowledge, it is the real-world dataset which is the most similar to what we need for evaluating our algorithms.

We defined 10 skills connected in a binary tree, and a similarity function based on their distance in it: if it is greater than the half of the longest path we consider the skills unrelated. In this way we model a non linear and quite complex relation of similarity which can be suitable for a real scenario where, let's say, there is a big competence area divided in subareas each of them is composed by other subareas. In this case it is logical to consider two subareas of the same quite related each other, and two areas too distant in the graph unrelated. We consider 10 skills to be sufficient to describe the competences of a significantly large community, as stated – for example – by U.S. National Research Council [55] in the case of Computer Science.

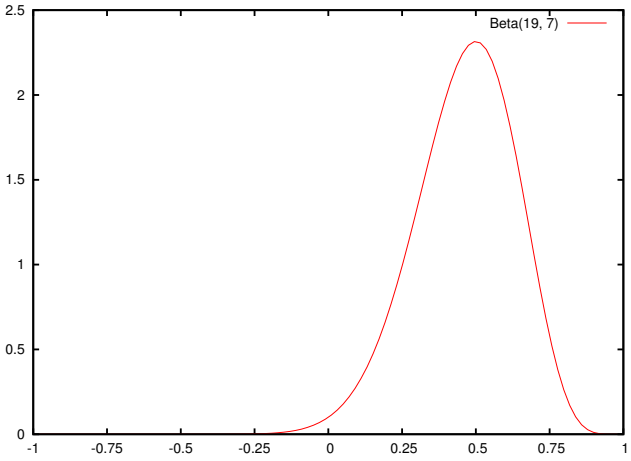
The assignment of the skills to users is made using different algorithms in order to simulate different degrees of cohesion in the community. The number of skills of a given user is a random number drawn with geometric distribution (Figure 3.5a), since in this way there are few users with many skills, while a large number of users have only a few of them. Starting with a subset of users, we uniformly select a skill for each user in such subset. For each of such skills, we then randomly generate a set of related skills and we assign them to the neighbors of the users in the initial subset.

After that, we use a beta distribution (Figure 3.5b), modeled according to real rating from Epinions dataset, to compute at random his reputation as seen from other users. We chose such a distribution type, because it better fits the Epinions data.

For our experiments, we generated a set of social graphs of various sizes. Namely consisting of 50 to 200 users. The complete statistics of such graphs are presented in Table 3.1.



(a) User's skill number is defined by a geometric distribution



(b) Users' rating are generated from a beta distribution scaled in $[-1; 1]$

Figure 3.5: Distribution used to generate the team formation dataset

# Users	# Reputations	Users similarity (avg)	Competences similarity (avg)
50	65	0.542773	0.718668
100	283	0.560407	0.659207
150	757	0.547320	0.611460
200	1286	0.493560	0.580858

Table 3.1: The statistics of the generated datasets

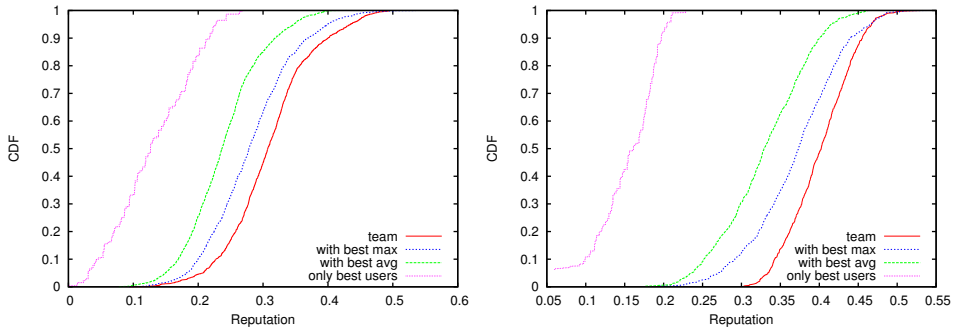
Evaluation

To evaluate the algorithms, we performed several series of tests with the datasets generated. For each dataset we identified the top user for each skill, which means the user with the highest reputation for a given skill according to its incoming edges.

We simulated only the scenario in which a user wants to perform a task being one of the member of team (Algorithm 2) because the scenario in which a user wants to commission a task to some other users is a generalized version of the first one, where the first algorithm is called multiple time for each possible team leader (Algorithm 3). That is, the team formation algorithm is required to find the remaining members of a team able to perform a given task. We limited the search of suitable users to different distances $d \in \{4, 5, 6\}$ noticing that, while increasing the search distance from 4 to 5 there is sensible growth of the average reputation, using searching distance greater than 6 does not change the quality of the results.

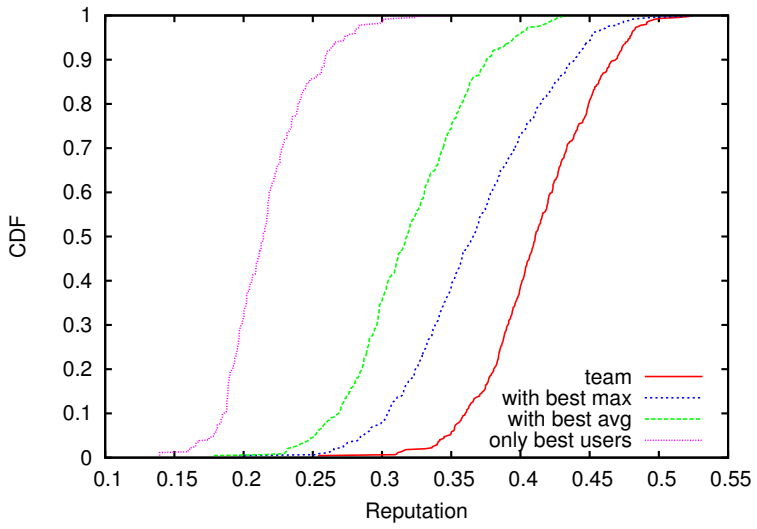
We compare the reputation of the generated team with those of the team composed by the top users and the teams created substituting each user of the original team with the corresponding top user (excluding the user who requested the generation of the team). Figure 3.6 shows the team reputations using a cumulative distribution function.

We can observe that, on average, the teams generated by the proposed algorithms have a better reputation score than the ones consisting only of top users and than the teams in which we substituted a member with a top user, especially when the user graph became bigger, because it statistically became more difficult to have a top user near the selected team. We remind that the computed reputation takes into account indirect trust, therefore the computed value is greatly affected by the subjective trust that each user has in the other members of the team. Hence, according to the presented experimental results, we can say that the proposed algorithm is a suitable mean to identify cohesive teams which should, reasonably, fulfill the appointed tasks.



(a) Graph with 100 users

(b) Graph with 150 users



(c) Graph with 200 users

Figure 3.6: The results for commissioned team in different graphs

3.2 Generalization: service composition system

In this section we discuss how to extend the previous model to work in a totally different context with a slightly different definition of reputation.

Internet of Things (IoT) and Web of Thing (WoT) are two new paradigms that are built on the idea that nowadays we are surrounded by a variety of *things* which are able to connect each other to interact and reach a common goal. The idea of both of these paradigms is that, if the things communicate using a standard protocol, with the proper discover protocols they can interact in an easy way simplifying – with a potentially high impact – the every day life and behavior of potential users.

The difference is that in the former paradigm objects communicate using protocols built directly on the top of TCP, in the latter they communicate using web protocols built on the top of HTTP. The strength of the former consists in more flexibility and capability – i.e. bidirectional communication sockets – while, on the other hand, the latter makes service composition and test easier also for not programmers – let's think that it is possible to interact with REST interfaces using a normal browser.

For example, imagine a domotic hotel room: instead of using a remote for the television, another one for the air conditioner and searching the desk for the restaurant menu, a smartphone could initiate a discovery protocol to find all the nearby smart objects, discover the television, the air conditioner, the restaurant website and – using some well known standards – automatically create an interface to interact with them. The big advantage of this approach is that the interface will be uniform and localized in the user's language, freeing him/her from the need to understand and learn how to use the new remote each time he/she moves to another hotel.

In this scenario, standardization is essential to let all the devices communicate with each other. Service discovery protocols are important too to let users make queries like *find all the nearby devices* or *find an online rock radio station*. Service composition tools must be so easy to use to let everyone compose a service like *play the web radio station to the closest stereo and pause it when my phone is making a call*; last but not least, it is important to have reputation tools to rank and filter services.

When the services are potentially unlimited, and deal with sensible data, it is essential to filter and use only the trusted ones: if my phone can talk with the bus stop sign to retrieve the bus timetable and pay the ticket, I want to be sure that I'm buying a real ticket and not a faked one; if I ask my television to show a webcam of Prague, I want to choose between a list of webcams ranked by decreasing quality, for some – maybe complex – definition of quality, i.e. best sight, possibility to remote control it and so on.

3.2.1 Motivating example

In this section we discuss about how to attach a reputation system to a service composition system. We will not deal about how to discover services or how to connect them, our assumption is that already an infrastructure exists for it or – if it does not exist or is not completely automatized – it is supported by manual work.

In this scenario imagine someone who wants to set up a new e-commerce website. In order to work, the website needs a set of services: a web hosting, a database, a payment service, a mail tracking service, a datasheet provider to automatically fill the technical specification of the item sold.

Nowadays usually – especially for small projects – people do not bother too much to find these services because they rely on some well known one. For example database and web hosting are typically bought from the same provider, well known payment services are hard coded in the website and technical specifications of items are written by hand or totally missing, leaving the duty to find them to the user.

In an IoT scenario, all these services are exposed and described in some standard language that makes them discoverable and understandable by search engines. Hence it is possible to query the search engine and obtain a list for each service needed. At this point the problem becomes choosing the services to use among all the possibilities. In this scenario – as in the one described in Section 3.1.1 – reputation cannot be computed globally; imagine for example a very popular and high quality database service, it is better to discard it, despite its high reputation, if it exposes the API only for Ruby and the website is written in PHP. On the other hand, if we have a website written to use MySQL but we find a better database service that uses PostgreSQL, we can consider migrating to it because the little effort to change the website is worth the better service.

In our model a service can recommend another one when they are connected by a trust relation. In this case trust measures an aggregation of different factors: mainly API compatibility or known problems in service composition. Also users can recommend services, the idea is that their recommendation is based on a trust computed by slightly different aggregation of factors: for example a user can evaluate better the quality of the assistance service or the respectfulness of the service level agreements.

The aggregation of these trusts, computed locally to the requester, gives the reputation of the service. Thus the reputation is used to rank and select the “best” service aggregation according to the user’s request.

3.2.2 The model

The model presented in this section is an extension of the one presented in Section 3.1.3; to summarize we can say that the main difference consists in

adding the support for the users who provide reputations but no services. In order to support this change we had to modify the algorithm to compute reputation and alter all the definitions to fit the new scenario.

We assume a finite set $SP = \{sp_1, \ldots, sp_n\}$ of service providers, that will be grouped in working unit WU to perform some complex task. A finite set of reputation provider $RP = \{rp_1, \ldots, rp_m\}$ contains whatever in the system can provide reputations but not services (i.e. users or service comparison aggregators).

We also assume a finite set $S = \{s_1, s_2, \ldots, s_k\}$ of service types that are provided by the service providers (i.e. cloud storage, database server, web hosting). Services can be interchangeable, in the sense that it is always possible to exchange two service providers when they offer the same service and is likely to exchange two service providers when the services are similar enough to be exchanged with minimum modification in the system. Therefore we introduce a measure of the service interchangeability $l : S \times S \to [0, 1]$ informally defined as follows:

- $l(s, s') = 1$ iff s' can be used in every situation where s can be used (e.g. they offer the same database model and version);
- $l(s, s') \in [0.5, 1)$ iff the services can be exchanged with minor effort (e.g. different databases but with the same language specification);
- $l(s, s') \in (0, 0.5)$ iff the services can be exchanged with big effort (e.g. different web hosting with different language support);
- $l(s, s') = 0$ iff s' is a totally different service compared to s (e.g. a random number generator and a web hosting).

Furthermore, we assume that the interchangeability function l satisfies the reflexive property – namely $\forall s \in S$, $l(s, s) = 1$, which means that every service can be exchanged with itself.

We do not assume that l holds a transitivity-like or triangular-like property, because knowing the difficulty to migrate from a service s to a service s' and from s' to s'' does not let us infer any relation between s and s'' that could be the same service or a total different one.

We do not even require a symmetric property, because even though it seems reasonable – i.e. the effort required to migrate from MySQL to PostgreSQL is reasonably similar to the one required to change back – it is completely wrong when it is possible to simulate service with another one that offer a larger set of API – i.e. it is easy to simulate a key-value pair storage using an SQL server, but the opposite is far from trivial.

In our model, we define a working plan – WP – as a subset of service types. It defines the sketch of the project, itemizing the services needed. The algorithm we propose takes as input a working plan and provides as output the working unit with the highest overall reputation able to fulfill

the working plan requested. We highlight the fact that the service types exposed by the working unit may be different by the service set requested by the working plan, because we think that is better to make a little effort and change the initial plan to use a highest reputation service – more reliable for example – instead of changing nothing and using a very low quality service provider. Besides, the service interchangeability function assures that the changing effort is always minimized and worth the reputation increase.

In this model, each service provider is identified by its set of services exposed, as example note that almost every web hosting service provides a database service too. Nevertheless, the reputation of the same service provider can be different regarding which service type we are considering. Our algorithm considers this issue by looking for the best matching of the service type, and not service provider. Therefore it can happen that a service provider has an unused – but required – service if the algorithm is able to select a higher reputation service provider for it.

To find the best working unit to successfully accomplish a given working plan $WP_t = \{s_1, \ldots, s_w\}$, we take advantage of ratings assigned by each reputation provider and service provider. Thus, the proposed model does not only leverage on service provided as declared by a service provider, but on such quality as defined by other – both service and reputation – providers according to their past experiences interacting with it.

We recall that a service provider can express a trust value upon another one according to internal policy (like API compatibility or commercial agreement). But if we only rely on them the risk is to find a very low connected and highly clustered graph centered on the biggest providers. We also miss an entry point for the user who has a working plan to implement. Whereby we introduced in the model the *reputation providers* as entities that provide *only* reputation. A user is a reputation provider that can trust services as well as other reputation providers – coworker for example – and so the entry point in the graph is the reputation provider represented by the user who is projecting a working plan, so the algorithm uses his/her connections to discover the services requested by the working plan.

In such a way, ratings may help in building working units that collect service providers that are known working well together, maximizing productivity and minimizing connecting effort.

For the sake of readability, we introduce a new set $P = SP \cup RP$ which aggregate every entity that can provide reputation – both service and reputation providers – and we will call it just *provider*.

The reputation function r collects all the trust values in the systems. Formally is defined as

$$r : P \times P \times S \to [-1, 1]$$

where $r(p_i, p_f, s)$ denotes the trust that the entity p_f has gained on the service type s as assessed by p_i, from -1 (totally distrusted) to 1 (totally

trusted). The precise semantic is a bit tricky and depends on what is evaluating what (service or reputation provider). An evaluation between two reputation providers can assert the competence regarding the usage of the service or the credibility of the evaluated. An evaluation between two service providers can assert the API compatibility or special commercial agreement like discount if using both of them. An evaluation from one service provider to one reputation provider can be related to some sort of *certified user*, like a company that certifies its consultants or known aggregators. Eventually, an evaluation from one reputation provider to one service provider is – usually – the one that contains direct usage experience as seen by some user, team or organization.

The reputation function r induces a weighted, labeled, directed multigraph $\mathcal{G}$ over the set P of providers: two providers p_i and p_f are connected by a directed edge labeled with s and weighted with the value x if (and only if) the reputation function is such that $r(p_i, p_f, s) = x$.

In general, when dealing with large online communities, we may assume that the graph $\mathcal{G}$ is not completely connected, which means that it is unlikely that all the providers have a direct reputation about all the others.

Based on such assumptions, we propose an algorithm for computing the reputation of a service provider p_f, according to the required service s, as seen by an entity $p_i \in P$ not necessarily directly connected in $\mathcal{G}$ by an edge labeled s to p_f.

This is achieved by searching the shortest path $\mathcal{SP}(p_i, p_f)$ in $\mathcal{G}$ between p_i and p_f such that its edges satisfy the following condition: given a service $s \in S$ and an edge $e \in E$, the weight of the edge e with respect to s, denoted by $w_s(e)$ is computed as:

$$w_s(e) = \frac{1}{l(s, s(e)) \cdot (2 + r(e))} \tag{3.3}$$

Informally, Formula 3.3 assigns higher weights to edges with services very different from the requested one (when they are totally different the interchangeability function is null, so the weight is infinite) and with low reputation. In this way, the algorithm yields a shortest path having edges with highest reputation labeled with services most interchangeable to the requested one. Furthermore, we require that $\forall e \in \mathcal{SP}_s(p_i, p_f), r(e) > 0$, unless p_f is the head of the edge e. We introduce such a constraint because in a chain of trust – recall its definition in Section 3.1.2 – one does not want rankings from untrusted sources. On the other hand, the last edge may be negative because, in such a case, we trust a negative opinion expressed by a trusted source. We refer to such a path between two nodes as to *admissible path*.

Having defined what is an admissible path between two nodes in $\mathcal{G}$, we now proceed to describe how to compute the reputation between them with

respect to a service $s \in S$. We define an *extended* reputation function

$$r_s^* : P \times SP \to [-1, 1]$$

that takes as input a provider p_i, a service provider p_f and measures the reputation that p_i has of p_f about the service s. The computation of the extended reputation function is based on the reputation values defined by the different providers, on the edges composing an admissible path between p_i and p_f. It is described in Algorithm 4 and is the minimum value of the reputation multiplied the interchangeability between the services found on the admissible path.

Data: The service s to search for the reputation
Input: The admissible path $\mathcal{SP}_s(p_i, p_f)$ between two providers
Output: The extended reputation function $r_s^*(p_i, p_f)$
begin
  $rep = null$;
  for $e \in \mathcal{SP}_s(p_i, p_f)$ **do**
    $rep = min(rep, l(s_e, s) \cdot r_e)$;
  end
  if $rep == null$ **then**
    return 0;
  else
    return rep;
  end
end

Algorithm 4: Computation of the extended reputation function

We have now the tools required to compute the reputation for a given working unit and, therefore, to select and rank the highest reputation working unit that can fulfill a given working plan.

Firstly we define the reputation of a working unit W as seen from the reputation provider $u \in RP$ – who is looking for the service aggregation – based on the function r^* previously described, as the average of the reputation of all its service providers and the contractor u. More formally

$$r_u(W) = \frac{\sum_{p \in W \cup \{u\}} \sum_{p' \in W \cup \{u\} \setminus \{p\}} r^*_{s_{p'}}(p, p')}{|W|(|W| + 1)} \tag{3.4}$$

Reminding that a working plan is defined over a set of service types, our goal is to find a set of service providers – the working unit – such that:

1. all the services required in the working plan – or some easily interchangeable service – are provided by some service provider;

2. each service provider contributes providing at least one service;

3. the size of the team is the smallest possible, in the sense that there is at most one service provider for each service required;

4. the reputation of the working unit is maximized.

The exact process is described in Algorithm 5, informally what we do is looking for all the service providers at a known distance from the provider that requested the composition. For each service requested we create a list of all the providers that can provide it or an easily exchangeable one – according to the defined service interchangeability function l – and compose from it all the possible working units. We compute the reputation for each working unit and return the highest one, or an error if no working unit is found.

Data: A map *providers* between services and a list of providers that can provide a compatible service
Input: A working plan $W_p = \{s_1, \ldots, s_n\}$, the node $u \in P$ that initiated the request, a searching depth limit d
Output: A working unit W or $\perp$ if such unit can not be found within depth d

```
begin
    services = ∅;
    for each service provider p distant at most d from u do
        for each service s of p do
            for each service s' of W_p do
                if l(s, s') ≥ 0.5 then
                    providers[s'].add(p);
                end
            end
        end
    end
    if providers does not contain at least a service provider for each
    service in W_p then
        Failed to find a suitable candidate for some task skill;
        return ⊥;
    end
    for every working unit W ∈ providers do
        compute r_u(W) and return the working unit with maximum
        reputation
    end
end
```

Algorithm 5: Service composition of a working unit

3.2.3 Complexity

As in the team formation algorithms described in Section 3.1, also the core of the Algorithm 5 consists in finding the admissible path between two entities. Therefore, all the considerations made in Section 3.1.4 are still valid and we can consider the complexity of this algorithm as the complexity required to find the shortest path between the two providers. Using Dijkstra's algorithm, the time complexity is $O(|P|^2)$, where P is the set of providers.

3.2.4 Experimental results

The algorithm presented in this section was tested with an enhanced version of the prototype described in Section 3.1.5. It selects the 10% of nodes that have the highest ratio between outgoing and incoming reputation as reputation providers, all the other nodes are labeled as service providers.

In this prototype we relaxed also the constraint that trust must be a symmetric relation: in the previous one it was a reasonable constraint because we considered trust as a relation modeled after a teamwork, so if two persons have worked together it is reasonable to think that both of them have a certain degree of trust – maybe different and related to different skills too – considering the other. In this case this constraint is not reasonable anymore because if someone trusts a service there is no reason to think that the service must have any knowledge of the person who is using it – also if it is not strictly forbidden.

The tests were made on an Intel® Core™ i7 3.40GHz PC with 8GB of RAM. Note that the prototype is not highly parallelized so its execution time does not fully benefit of the multi-core architecture. The average execution time for each service composition request is displayed in Figure 3.7, and it goes from 7 seconds for the smallest graph (50 nodes) to 42 minutes for the biggest graph (130 nodes). Also with these few points it is possible to spot that the execution time is exponential in the size of the graph.

Figure 3.8 displays the reputations for different service aggregation requested from a random reputation provider that require a random set of services. Using a cumulative distribution function, the graphs show the probability that a working set has reputation lower than the given x value. We can see that the working set selected by our algorithm has higher reputation compared to the one aggregating only the services with the highest reputation for the services type requested. Generally speaking, changing only one service with the one with the highest reputation is enough to drop the working unit overall reputation. There are only some exceptions in the graphs with 70, 110 and 130 nodes (Figures 3.8b, 3.8d and 3.8e) where changing only one service could lead to comparable – sometimes even slightly better – results. These cases can be easily explained seeing the graph structure, where some very high reputation provider is well connected with a lot

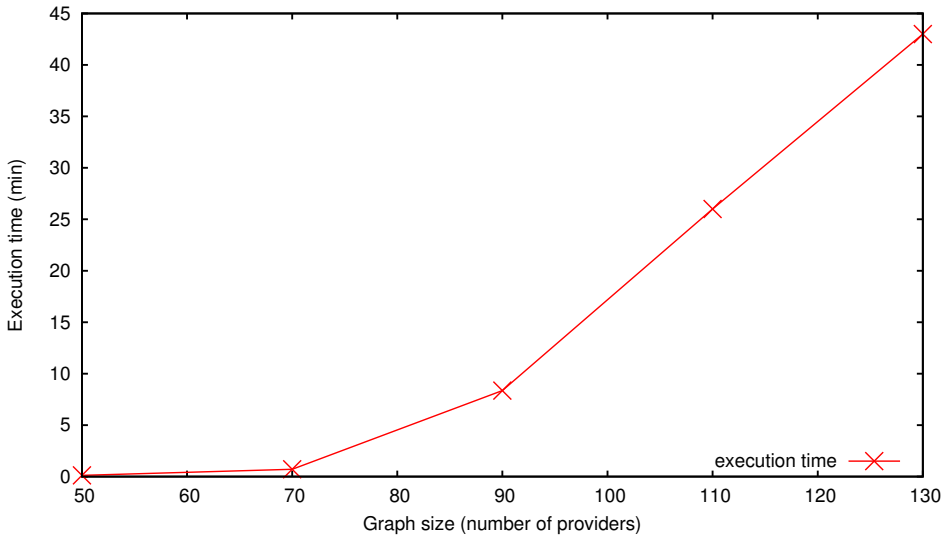
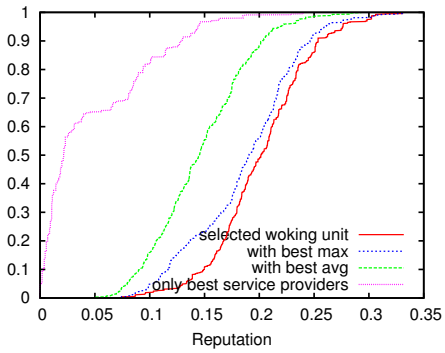
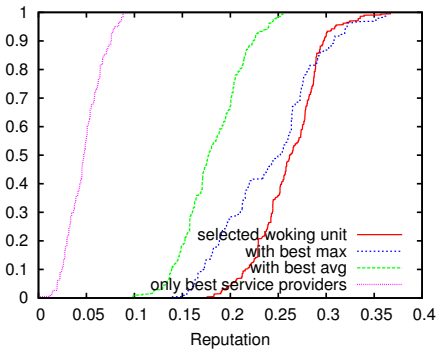


Figure 3.7: Service composition of a working unit execution time

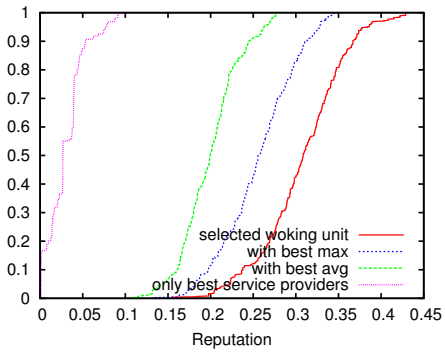
of nodes, but not connected well enough to fall within the search depth limit we used in the tests (5 levels). So our algorithm does not find it, but when we force its presence in the new working set the overall reputation results improve.



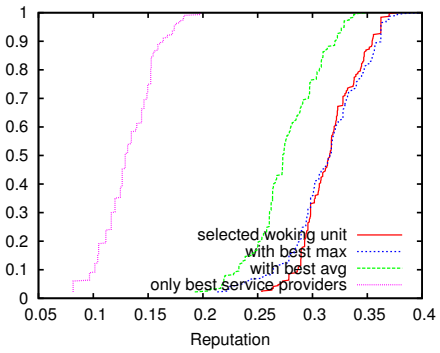
(a) Graph with 50 providers



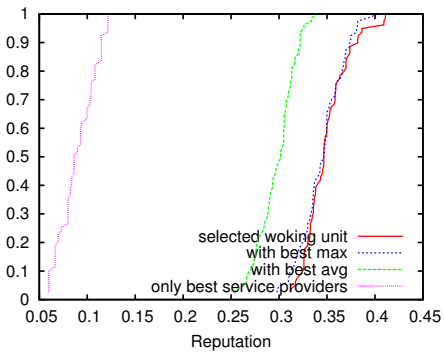
(b) Graph with 70 providers



(c) Graph with 90 providers



(d) Graph with 110 providers



(e) Graph with 130 providers

Figure 3.8: The reputation of selected working unit in different graphs

Chapter 4

Implicit collaborative environments: the wiki case

In the modern Internet we can find a lot of websites and platforms for implicit collaborative environment. According to our definition (in Chapter 1) an "implicit collaborative environment" is where people collaborate without knowing or choosing the other contributors.

Using some wide interpretation of this definition we can consider also newsgroups and forums a sort of implicit collaborative environment: people from very different backgrounds collaborate to a – usually thematic – knowledge base adding small pieces of text in a sort of questions and answers format.

However, thinking about one public place where people from everywhere can collaborate to achieve a common goal the first idea is very likely Wikipedia. Since its opening in 2001, Wikipedia grew in popularity and size becoming one of the most visited website according to Alexa statistics[1].

The growing popularity of Wikipedia leads to an increase of popularity of the wiki platform in general. The Wikimedia Foundation itself, besides supporting Wikipedia, operates several other wiki projects. Moreover, a lot of websites offer free wiki hosting services, so it is easy to find also very small wikis that focus on some narrow topic.

For these reasons in this thesis we focus on wiki systems to discuss about implicit collaborative environments, analyzing two different case of studies.

In the Section 4.1 we will present a novel model for wiki that deals with semi-structured data that we developed to solve some of the tricky issues that arise when we try to store structured data in a plain text format without the ability to add any kind of constraint. Starting from a real-world scenario, we point out such issues and present a simple and efficient framework to store semi-structured data in a consistent way focusing on the following key concept:

[1] `http://www.alexa.com/topsites`

1. define ad-hoc methods for automatic documents evolution upon template change;

2. propose a new kind of revision control system that is centered on how to minimize loss of new data and maximize data coherence in case of template rollback;

3. define a simple, ad-hoc, XML-based data model to store data and an XQuery like update language for it.

The proposed solution has been implemented in an XML-based prototype framework, which has been tested with large, real-world dataset.

In the end of the section we will show how to attach to this new wiki a recommender system that can rank users according to their contribution quality.

In the Section 4.2 we will discuss about the problems of a classical – plain text – wiki system. In the whole section we will use the name wiki as synonymous of Wikipedia for three reasons:

1. it is the biggest and most known wiki around the world;

2. a huge and very active community works day by day to maintain fresh and quality content;

3. it provides its full content for free, so we had the ability to use it as dataset to validate our idea.

4.1 Case of study: semi-structured data Wiki

Developing a wiki for semi-structured data is not a trivial task. Changing radically the kind of data to store raises many open problems to solve. For example, is not trivial to define how a document should react upon a template update; what should happen to a document if someone reverts its template; or how can we rate the documents quality when it strictly depends on the template. Eventually, the biggest problem of all wikis: how to encourage users to contribute. We will propose a solution to these problems in this section.

4.1.1 Project motivation

The community of users of a large data-driven web site may directly contribute to its management by feeding corrections and new additions, thus keeping "fresh" the information provided by the site. However, several issues may arise due to the fact that users may modify data in a more or less controlled way.

In the last years, wiki systems have been widely adopted on the web. In fact, besides Wikipedia, a large and ever-growing number of open source projects use wikis to share documentation. Although many studies demonstrate that large communities can develop and maintain a very high-quality wiki, there is the problem of users accessing information that may be outdated, deliberately vandalized, or simply partial. In order to let users leave a feedback on the perceived quality of the information they access, Wikipedia itself is working to introduce a system to rate articles[2].

The aim of this project is to design methods and techniques to be implemented in a system in order to evaluate the quality of the pages and the trustworthiness of the contributors in a wiki-based repository for fine-grained semi-structured data.

Large electronic commerce sites allow users to actively participate in collecting feedback information about items and products available on such sites. Typically, users may add comments and rank or tag items according to their preferences. Typically, the information provided by e-commerce sites' users is used for providing tailored recommendations to them and as the functionalities offered to the users for adding and manipulating information are very limited: users typically can only add comments to items and rate them, according to some ranking criteria. Following the recent trend of community-based information management [22, 21, 9] – according to which the information may be directly manipulated by end users – we propose to augment the "expressive power" of tools users employ in organizing and managing the semi-structured (XML-based) information they provide about

[2] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Article_Feedback_Tool&oldid=573517780`

items, in order to allow an expressive and useful structuring of the information itself. As such, users may add and structure information about items (or even add new items) of their interest. We adopt an approach inspired by wikis, in which users may structure the data they provide in complex ways and manage it in a collaborative way.

Our application scenario (see Section 4.1.2) deals with a vast number of XML documents containing information about items as shown on a large e-commerce site and, since such information is not completely unstructured, our framework assumes that each document may (possibly in a loose way) adhere to one of a relatively small number of schemas (thereon called *templates*). Henceforth, users may create and modify both documents and their corresponding templates. Furthermore users may interact by modifying documents and templates created by other users as well, thus adding and modifying information in a collaborative way (of course, such interactions have to be regulated by proper access policies and trust/reputation mechanisms stating which users may modify which data. In the present work we do not deal with such issues). Having the users such possibilities entails that our system has to take into account the interplay occurring among a modified template and the corresponding (unmodified) documents. More precisely the system, enforcing the constraints about the document structure, supports users in finding ill-formed documents by allowing them to improve their content.

While supporting a community-based approach for updating documents and templates offers the advantage in keeping information up to date (provided by users' feedbacks), such an approach poses several non-trivial questions about the correct management of such information. In particular, (i) when and how updates on templates are reflected on the corresponding documents? (ii) How to manage the rollbacks of unwanted (possibly malicious) updates *without* erasing subsequent licit ones? And how such rollbacks on templates affect the corresponding documents?

This may be non-trivial tasks, as we will argue in the following. As for point (i) above, the problems we incur in dealing with a community-inspired data management approach come from the existence of integrity constraints that are expressed in a template and that the corresponding documents have to satisfy; whereas for point (ii) the difficulties arise in guaranteeing the maximum possible number of updates without losing validity of documents with respect of their templates.

In fact classical wikis handle structured data through the use of infoboxes[3], but they do not support any kind of constraint enforcement which can guarantee the uniformity of pages which use them (see the example in Figure 4.1).

The framework we present in this work is based on XML Schema [44, 66]

[3] `http://en.wikipedia.org/w/index.php?title=Help:Infobox&oldid=564475822`

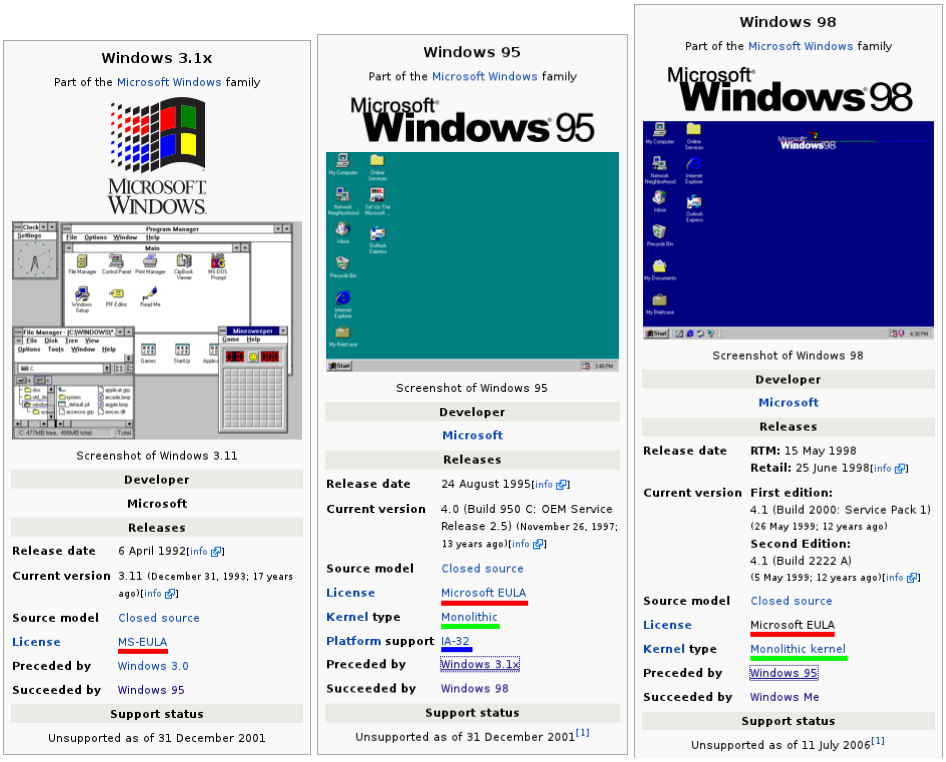


Figure 4.1: Infoboxes do not enforce any constraint about data; in this example, three infoboxes describe similar data in a not uniform way (see the line highlighted)

for the definition of the basic structure of templates, on Schematron [34] for the definition of more complex integrity constraints and on a XQuery Update-like language for the query language.

4.1.2 Motivating scenario

As a real-world example of a large repository of relatively small and relatively uniform documents, we consider the price comparison service ShoppyDoo[4]. This site holds the large majority of the market share in Italy with more than 2 million users and it has a very significant presence in other European countries (e.g. Spain, France, Germany, Netherlands) and non-European as well (e.g. Brazil). More details on data volumes are detailed in Section 4.1.12.

The site stores information about more than 1 million items, grouped in roughly one hundred categories. Items are described in pages containing their technical details. The information is displayed in concise and tabular

[4] `http://www.shoppydoo.it/`

form for letting users quickly find and compare items.

For example, the information about technical details about digital camera must specify brand and model, as well as camera resolution and memory support. Other less relevant – but still useful information – may comprise the presence of features like an image stabilizer or a face detector, etc.

The users of this site form an online community that may create, update and share information about items by interacting with the site itself. At the present time, the site does not allow its users to actively participate in the management of the displayed information. Our long-term goal is to provide community-based information capabilities to a large, e-commerce-based web site.

Thus, for example, in the case that the user Alice notices that the page describing her preferred camera reports incorrect information about its resolution, she can correct it. Further, Bob is a more active user and notes that almost every digital camera is able to connect to a pc and thus explicitly specifying such information is useless. As such, he decides to modify the digital camera template in order to remove such information from every corresponding document.

In what follows we present methods and techniques that allow users to directly manage such updates and to control the interplay between documents and templates. We will present in a more detailed way the actions performed by such users, as we unfold our motivating scenario in the following sections.

4.1.3 Documents and Templates

We customarily represent a document as an XML tree. For example, a document containing information about a given digital camera may be structured in the following – rather conventional – way: the camera's model is stored in an alphanumeric string, its megapixel capacity is an integer number and the supported memory is a single value chosen from a set of alphanumeric strings. Further, we add a *section* element which is used to group related elements under a section name. Each element name is unique in its section.

Templates are defined in a way similar to what proposed by Examplotron [68]. We have chosen this approach for its ease of use (see Section 4.1.5). Thus, we define a template as an instance of an empty document, where each element has an additional boolean attribute specifying whether it is mandatory or not in the documents.

More formally, a document is composed by *data* nodes and *element* nodes where: a *data node* contains only a string value; an *element node* is a tuple associated with a name, a type and a set of children nodes;

Elements may have simple or complex types, where complex types are in $\{enum, values, section\}$ and simple types are in $\{int, float, str, bool\}$. The type of an element node defines restrictions on the set of children nodes.

That is,

1. simple type elements and *enum* elements children set must contain only a data node;
2. *values* elements children set must be a non empty set of distinct data nodes;
3. *section* element children sets can be only a non empty list of children elements with distinct names (in this way, elements can be found without ambiguity).

As such, the document is an unordered tree of section and value elements, where the root is a section element.

A template is composed by *template elements* which are similar to documents' one. In addition, they have an extra boolean *mandatory* attribute. Every template has to satisfy the following constraints:

1. template elements of simple type must have an empty set of children;
2. template elements with type *values* or *enum* must have a non empty set of distinct data nodes;
3. template elements with type section must have a non empty set of children elements with distinct names.

As for documents, a template is an unordered tree where root element is a template section element.

A document is *valid*, of course we assume the well-formedness, if all data nodes contain values that:

1. an integer, if the type of parent node is *int*;
2. a float, if the type of parent node is *float*;
3. a boolean, if the type of parent node is *bool*;
4. a non empty string, if the type of parent node is *str*.

A document is *valid with respect to a template* if and only if:

1. it is valid (see above);
2. all document's elements are also present in the template with the same name and the same type;
3. all template's elements with mandatory set has a corresponding element in the document;

4. the children of *enum*, *values* and *section* elements of the document are also children of the corresponding template's elements.

More formally, we define a function f between the document root and the template root such as a document element d and a template element t are related if one of

- $name(d) = name(t) \wedge type(d) = type(t) \in simple\ type$
- $name(d) = name(t) \wedge type(d) = type(t) \in \{enum, values\} \wedge children(d) \subset children(t)$
- $name(d) = name(t) \wedge$
  $type(d) = type(t) = section \wedge$
  $\forall d_i \in children(d) \Rightarrow f(d_i) \in children(t) \wedge$
  $\{t' = f(d') : d' \in children(d)\} \supset \{t' \in children(t) : mandatory(t') = true\}$

If the function f exists and the document d is valid, d is valid with respect to the template t.

4.1.4 Our scenario, continued

As one may suspect, documents and templates are stored in XML files. Document elements are serialized in XML nodes where the tag name defines the node type and the name is stored in an attribute. For example an element *Model* of type *string* is serialized as

```
<str name="Model">Z80</str>
```

A full example of a document describing a digital camera follows.

```
<document>
 <template>Digital_camera</template>
 <section name="Digital camera">
  <str name="Brand">Olympus</str>
  <str name="Model">Z80</str>
  <int name="Resolution (Mpx)">12</int>
  <section name="Aspect">
   <float name="Weight (g)">117</float>
   <float name="Width (mm)">96</float>
   <float name="Height (mm)">62</float>
   <float name="Depth (mm)">27</float>
  </section>
  <bool name="Built-in Flash">true</bool>
  <enum name="Supported memory">xD-Picture Card</enum>
  <section name="Lens system">
```

```
   <bool name="Auto focus">true</bool>
  </section>
  <values name="Extra feature">
   <value>Face detection</value>
   <value>Red eye removal</value>
  </values>
 </section>
</document>
```

Templates enforce the information type of complex element, in such a way that the *Wiewfinder* can be one of *optical* or *LCD*, while *Extra feature* must be one or more between a list of valid values. Templates XMLs are very similar to documents' one, the main difference is that simple elements are empty and that every element has an attribute *mandatory* which contain a boolean value. For example the element *Model* in a template is

```
<str name="Model" mandatory="true"/>
```

A template for digital camera can be saved in XML in a similar way

```
<template>
 <section name="Digital camera" mandatory="true">
  <str name="Brand" mandatory="true"/>
  <str name="Model" mandatory="true"/>
  <int name="Resolution (Mpx)" mandatory="true"/>
  <section name="Aspect" mandatory="true">
   <float name="Weight (g)" mandatory="true"/>
   <float name="Width (mm)" mandatory="true"/>
   <float name="Height (mm)" mandatory="true"/>
   <float name="Depth (mm)" mandatory="true"/>
  </section>
  <bool name="Built-in Flash" mandatory="true"/>
  <enum name="Supported memory" mandatory="true">
   <value>Secure Digital</value>
   <value>xD-Picture Card</value>
   <value>CompactFlash</value>
  </enum>
  <enum name="Viewfinder" mandatory="false">
   <value>Optical</value>
   <value>LCD</value>
  </enum>
  <section name="Lens system" mandatory="true">
   <bool name="Auto focus" mandatory="true"/>
   <bool name="Image stabilizer" mandatory="false"/>
  </section>
  <values name="Extra feature" mandatory="false">
```

```
    <value>Face detection</value>
    <value>Red eye removal</value>
    <value>Closed eye detection</value>
   </values>
  </section>
</template>
```

4.1.5 Validation of documents

We use XML Schema [44, 66] to validate documents and templates as serialized XML documents. This first step validates the overall structure of XML documents. Regarding documents, XML Schema is used to check that they are structured in sections containing the named values and the correct types of simple elements' values. Regarding templates, XML Schema is used to check sections, the uniqueness of name into their sections and the presence of a mandatory boolean attribute for each element.

As said before, users specify templates as an empty document. As such, in order to validate a document with respect to some template, templates themselves have to be rewritten in some suitable XML schema language.

We use Schematron [34] since its assertion rule validation style makes error reporting clearer and the usage of XPath constraints allows the definition of constraints over unordered sets of context-dependent elements. Furthermore, Schematron checks the presence of mandatory elements, the absence of illegal elements and the correctness of *values* and *enum* children elements. The conversion from a template to its corresponding Schematron schema is performed by an XSLT stylesheet [16].

As already pointed out, the main advantage in writing templates in the above presented XML format is its compact and easily readable syntax, that can be promptly deployed by the community users. For example, the simple constraint *Viewfinder can contain only 'Optical' or 'LCD'* in our template is defined by

```
<enum name="Viewfinder" mandatory="false">
 <value>Optical</value>
 <value>LCD</value>
</enum>
```

while its translation in Schematron is:

```
<sch:rule context="/document/section[@name='Digital
    camera']/enum[@name='Viewfinder']">
<sch:assert test="(text() = 'Optical') or (text() = '
    LCD')">
"<sch:value-of select="text()"/>" is not a valid value
    for enum Viewfinder
```

```
</sch:assert>
</sch:rule>
```

4.1.6 Evolution of templates and documents

Returning to our motivating scenario, since Bob considers it is very important to know if a digital camera is able to record videos, he adds a new boolean mandatory field called *video recording* in the camera template. We note that this kind of update invalidates all the documents associated to the corresponding template. We patch this problem adding a special page showing all invalid documents to let active users of the community perform updates on such documents, to restore their validity again.

4.1.7 Interacting with templates and documents

Community users can read, create and modify documents and templates. After a document is updated, the following steps are performed: (i) check whether the document is valid according to the corresponding XML Schema, if this is not the case, reject the update; (ii) otherwise get the associated template and translate it into a Schematron document; (iii) validate the document with the corresponding Schematron to check if it complies with the template and return the validation results. A pseudocode executing such steps is shown in Algorithm 6.

```
input  : A document doc
input  : A document update up
output: The status of the update
xup ← TranslateToXQueryUpdate(up);
doc ← XQueryProcessor(doc, xup);
if not ValidateWithW3CSchema(doc) then
    return RejectUpdate();
end
tpl ← FindAssociatedTemplate(doc);
sch ← TranslateTemplateToSchematron(tpl);
if not ValidateWithSchematron(doc, sch) then
    return RejectUpdate();
end
return AcceptUpdate();
```

Algorithm 6: Validation after a document update

A template update can (i) leave all associated documents valid (for example, the insertion of a new value in an enumeration); (ii) invalidate all associated documents (for example, adding a mandatory element in a mandatory section); (iii) require a necessary update and consequent re-checking

of all associated documents (for example, deleting an element); or, finally, (iv) leave the documents in an unpredictable state, regarding their validity (for example, an optional value becomes mandatory). In this case, the only way to discern the documents' validity is to re-check them all. Since the last two cases are similar, because the last one is like the previous with an empty update, we treat them in the same way.

The pseudocode showing the different cases just show is presented in Algorithm 7.

The update language we propose allows community users to create and update elements' types, their name, mandatory fields, add, modify and delete elements.

Since the community users may perform updates on templates and documents defined by the previously defined data model, we do not need the full expressive power of XQuery Update Facility [12] and, thus, our update language is basically a simplified version of XQuery Update. First, we define the element selector as a string that allows finding an element in unambiguous way. It is formed by the element name preceded by all sections name separated by the slash sign (e.g. `/Digital camera/Brand`). We also define a data selector as the selector of its container followed by a slash sign followed by the data text value wrapped by brackets (e.g. `/Digital camera/Supported memory/[CompactFlash]`). Those selectors can be easily translated into XPath[6] expressions. The last example in XPath is written `/*/*[@name='Digital camera']/*[@name='Supported memory']/*[text()='CompactFlash']`.

4.1.8 Evolution of documents

In this section we describe our document update language and how each command can be translated in an XQuery Update statement.

Every command that we define takes a selector as parameter. Whether it is an element selector or a data selector is first transformed into an XPath selector while the command is recognized and translated into a valid XQuery Update statement.

A user may delete either a data or element node with the command **delete** `node <selector>`. In this case we need only to translate the selector to have a valid update.

Add new data into a document let a user to fill values list. This operation is performed by the command

```
insert data(text) into <elementSelector>
```

which is converted into

```
insert node <value>text</value>+\phantom{m}\lstinline+
   into <xpathSelector>
```

```
input  : A template tpl
input  : A template update up
output: The status of the update
xup ← TranslateToTemplateXQueryUpdate(up);
tpl ← XQueryProcessor(tpl, xup);
if not ValidateWithW3CSchema(tpl) then
    return RejectUpdate();
end
switch StatusOfDocuments(tpl,up) do
    case all valid
        break;
    endsw
    case all invalid
        foreach doc in GetAssociatedDocuments(tpl) do
            SetValidity(doc, false)
        end
        break;
    endsw
    case must recheck
        sch ← TranslateTemplateToSchematron(tpl);
        xup ← TranslateToDocumentXQueryUpdate(up);
        foreach doc in GetAssociatedDocuments(tpl) do
            doc ← XQueryProcessor(doc, xup);
            if not ValidateWithW3CSchema(doc) then
                SetValidity(doc, false)
            end
            else
                SetValidity(doc, ValidateWithSchematron(doc,
                sch))
            end
        end
        break;
    endsw
endsw
return AcceptUpdate();
```

Algorithm 7: Validation after a template update

Since the most of elements node need a data child to be valid (all simple and *enum* types) we provide the command

```
insert node(type, name, value) into <elementSelector>
```

to insert both of them. When translated into XQuery Update, such command becomes

```
insert <type name="name">value</type>
       into <xpathSelector>
```

Differently, *section* and *values* elements have may child nodes, so we need an overload for this command that does not require a value.

The last command we describe is for replacing the value of a data node. Its syntax is

```
replace value of <selector> with <newval>
```

and is translated into XQuery Update in

```
replace <xpathSelector>/text() with 'newval'
```

4.1.9 Evolution of templates

Template evolution is more difficult because for every defined command we need not only to update – of course – the template but also to decide whether the documents associated have to be modified as well and, in the affirmative case, perform such updates. It is important to note that our node selectors (and the translated XPath equivalents) are valid both on documents and templates.

To add a new field to a template we have to specify the name, the type and if this information is considered mandatory. This operation is performed with the instruction

```
insert node(type, elName, isMandatory)
       into <elementSelector>
```

which we translate into XQuery Update with

```
insert <type name='elName' mandatory='isMandatory'/>
       into <xpathSelector>
```

If the inserted node is not mandatory, after this update documents are still valid. If the inserted node is mandatory and all ancestors sections are mandatory too, after this update all associated documents are no longer valid. Otherwise all associated document must be re-checked to define if they are still valid.

Another useful command is similar to the last one but lets user specify a default value for the new elements. The template update is equal to the last one, but in this case we have also the following document update.

```
insert <type name='elName'>defaultVal</type>
       into <xpathSelector>
```

New data nodes can be added to an enum or values element with

```
insert data(val) into <elementSelector>
```

This update preserves validity of documents and can be written in XQuery Update as

```
insert <value>val</value> into <xpathSelector>
```

The command

```
change type <elementSelector> with newType
```

is useful to change the type of an element. The template and document XQuery Update is

```
rename node <xpathSelector> as newType
```

It is important to note that, depending on the update and on the old data, document validity could be preserved after this update.

In a similar way, to change the name of an element we use

```
changename <elementSelector> with newName
```

and we apply to the template and the corresponding documents the instruction

```
replace <elementSelector>/@name with newName
```

The command

```
changemandatory <selector> with (true|false)
```

is used to change the mandatory value of an element. It is translated in

```
replace <elementSelector>/@mandatory with (true|false)
```

After this operation, if the updated element is not mandatory all documents are still valid. If it is mandatory, as well as all the section ancestors, all documents are marked as invalid. Otherwise, all documents have to be revalidated.

The simplest operation is the deletion of a node, that is performed in the same way both on templates and documents by the XQuery

```
delete node <xpathSelector>
```

The last operation is

```
replace value of <selector> with newVal
```

that uses XQuery functions to compute `newVal`. This operation is used to perform an update to a field on all documents associated to the current template.

It is important to note that particular updates sequence can leave the system in an inconsistent state. For example we have a template with a not mandatory element, then some documents will contain this element while some other will not. All documents and templates are valid. This is the initial state. A user decides that the element must be mandatory and update the template. As described previous, after the template update, all documents are revalidate. So we have some valid documents (those have the mandatory elements) and some invalid ones (all the other). In this state another user sees the new template and changes back the same element to be optional. An unskilled or heedless user can perform this operation as a normal update, instead of using the correct feature of the versioning system. So the template is updated however no operation on document is performed because this kind of update does not compromise their validity. But all documents that was marked as invalid is now valid again and in this final state we have all templates and documents valid but with some document marked as invalid. Therefore we need also a process scheduled for re-checking at regular intervals the validity of documents to achieve an eventual consistency of data.

4.1.10 Revision control support

Revision control is a very relevant feature for wikis. It is useful for monitoring a page evolution and dealing with modifications performed by malicious users.

Implementing a revision control system for our wiki-based repository is not trivial, since documents are represented as XML trees with complex constraints occurring among their elements, as specified by their corresponding templates.

In order to illustrate the problems in building a revision control system fulfilling the above mentioned conditions, we consider the scenario in which a user wishes to undo a template update, but – in the meanwhile, after the template update – the documents associated to this template have been modified, so they are no more valid with the old template.

In this scenario we can operate in two different and completely opposite way: (i) we can revert all documents to their valid versions with the old template or (ii) we can leave all documents to the last revision and revert only the template. Both solutions have pros and cons: The former maintains consistency between documents and templates but it can potentially lose many useful document updates. The latter does not lose document updates, but it can potentially leave all documents in an invalid state.

There is no single best solution to the problem of how to retain document updates while satisfying template consistency too, since it may depend very heavily from the context. For example, if one wishes to undo revert a malicious template update (possibly the outcome of a deliberate act of van-

dalism), it is pointless to save the corresponding document updates because documents contain information corrupted by the vandalism act, as well. In this case, the first solution appears to be the best fit.

On the other side, inconsistencies between a template and the corresponding documents arising from a minor change in the template structure (e.g., an element may become optional) seem to be viably managed by the second solution.

Given the extreme sensitiveness to the context of the chosen solution, our aim is to define helpful techniques and tools that support the user in adopting the best possible solution that fits its needs.

In particular, we define a hierarchy of possible solutions formed by the two scenarios introduced above, plus other two intermediate levels that offer a sort of "interpolation" between such two extremes. In more detail, when a user has to "revert" a template to a previous version, we can choose to:

1. revert all the corresponding documents to their previous versions, in agreement with the reverted template
2. revert only the structure (as dictated by the reverted template) of the documents but not the content.
3. revert only the document involved by the template update.
4. leave the document unmodified.

The following examples show a real-world scenario the four options just introduced.

As a deliberate act of vandalism, consider the following: a user deletes a large part of a template and renames the elements of the remaining part with unrelated content. A document update performed with the new template, can only try to limit the damages, but cannot improve the document quality. In this case the best solution is ignoring the document updates and revert documents as they were before the template update.

As for the second option, consider a car radio template, describing its features. In the *Car Radio* template, a user rename the *Audio* section in *Speaker system*. The semantic of the section does not change. So every document update is legit. But all the other templates which describe consumer electronics stuffs contain an *Audio* section. So, to uniform the templates, is better revert this update undoing the rename but without lose any update to the documents (Level 2).

Consider a *Notebook* template which contains a *Memory* and a *HD* sections. Both of them contains two integer *Total size* and *Max size*. An unskilled user, who does not know the difference between RAM memory and Hard Disk, can rename the *Memory* section in *Disk sizes* section hoping to make the template better. This update changes the semantic of the

section. So every document update which involves this section is compromised. When someone reverts the template update, the best solution should revert all documents updates which involved this section, but should keep all the others (Level 3).

Suppose that someone adds a new data *Autofocus* under the multi-values *Extra features* in a camera template. This is a useful information, but it is redundant since there is an optional boolean field *Auto focus* in the section *Lens system* yet. In this case document updates happen after the template update contains useful information but in the wrong place. The better solution consists of reverting the template but not the documents so no information is lost (Level 4). In this way we can encourage the community to correct the documents moving the information in the right place.

4.1.11 Our framework

Our framework, as shown in Figure 4.2, is composed by four components: (i) the XML repository, (ii) the query/update processor, (iii) the document and template validity checker and (iv) the user interface.

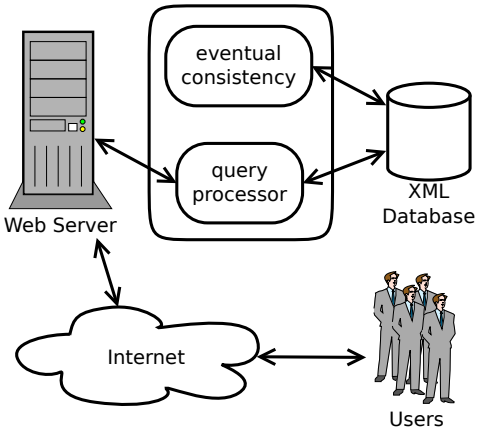


Figure 4.2: Framework schema for semi-structured wiki

For the XML repository we need a system that can efficiently handle many XML files and that supports XQuery Update. We choose eXist[5] because, as of now, is the most standards-compliant and extensible among free native XML databases. This component provides the persistence of data, the XQuery Update engine and the XML validation.

The query processor component takes as input the user queries and translates them into XQuery Update. It communicates with the database by sending updates, asking for validations and performs commit or rollback

[5] `http://exist-db.org/`

depending on validation results. Algorithms described in Section 4.1.3 are implemented in this component.

According to the XQuery Update Facility recommendation, updates are not performed immediately but are stored into a *pending update list* which is an

> *unordered collection of update primitives, which represent node state changes that have not yet been applied* [13]

and they are executed at the end of a query in a possibly different order than as they were written. So following updates

```
insert node <section name="new section"/>  into /,
insert node <int name="new int">21</int> into /section
   [@name="new section"]
```

will raise an error because when they are parsed the new section does not exist yet. We need to normalize them as

```
insert node <section name="new section">
               <int name="new int">21</int>
            </section> into /
```

to preserve the semantics defined in the previous section. This normalization is performed by the query processor. Since eXist update language – although it is quite close to XQuery Update Facility – does not use the pending update list paradigm, in our framework we had not implement the query normalization engine yet.

The user interface provides a set of features which let the user to intuitively interact with the prototype without the requirement of a formal training. Since the purpose of the prototype is to build a collaborative system, the user interface is a web application. It lets the user to (i) easily find documents and templates based on different criteria, (ii) read a rendered version of templates and documents and (iii) edit templates and documents in an interactive way.

4.1.12 Experimental results

Since the relatively small size of documents and templates and since updates and validations are performed on a native XML database, we can safely assume that a single update operation is performed in a relatively short, constant time roughly equal to 10 ms.

Thus, the performance issues of our framework depend on the number of documents that have to be checked, depending on the kind of update operation that has been issued.

All tests have been performed on a PC with an Intel® Core™2 Duo P8700 CPU and 4GB of RAM running Windows Vista™. The framework is implemented in C♯ 3.0 and deploys eXist 1.4 as the XML native database.

As already mentioned in the introduction, the dataset used in the experiments come from the ShoppyDoo online price comparison service which is visited by more than two million of unique user per month and compare prices of four million offers from fifteen hundred merchants. Every document describes the technical details of a single product and there is one template for product category. In total, our dataset contains 9605 documents and 72 templates, totaling about 65 MB of XML files. The less populated category (Digital photo frames) contains only 3 documents, while the most populated one (LCD, LED and plasma TVs) contains 798 documents. We want to remark that the current dump of Wikispecies[6] is approximately 370 MB of XML data. It's interesting for us show that our test dataset is big about the 18% of a small Wikimedia Foundation project.

As explained in Section 4.1.7, a template update can fall into three different categories, depending upon its impact on the associated documents: it can leave them all valid, it can invalidate all of them or it can make necessary to update and recheck all of them. As such, we performed editing templates tests with different quantities of documents and with the three different kinds of update. The results are shown in Figure 4.3.

The first type of update is very fast and it is not affected by how many documents are associated to the modified template. Since the Algorithm 7 needs only to update one (typically) small XML document (the template itself) it executes in about 100 milliseconds.

We point out the in our current prototype, all documents have a metadata field that records their validity with respect to their templates. In this way, the second type of update performs in a time linear in the number of documents. Since updating the validity to given value is very fast, all our tests ended within 150 milliseconds.

The last type of update is the worst because, after the template update, the prototype has to perform an update to all the documents and they all have to be revalidated. The time is linear to the number of documents, and since validation process is slower than the update execution, the worst case takes about two minutes to execute.

4.1.13 Reputation system for a semi-structured Wiki

The purpose of the recommender system is to rank pages based on their quality (like Wikipedia Featured articles[7]) We want to develop a system that supports the community to (i) find low quality pages in order to improve them and – e.g.– quickly react to deliberate act vandalism and (ii) to find better contributors in order to support election of wiki administrators.

[6] `http://species.wikimedia.org/`

[7] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Featured_articles&oldid=482633582`

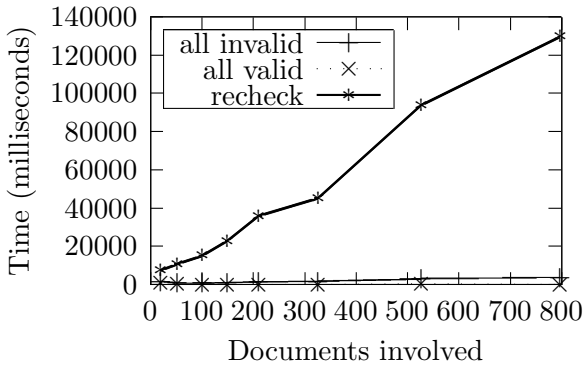


Figure 4.3: Template update performance test

Each step of the mechanized approach developed to evaluate Wikipedia articles' quality starting from revision history, is based on some criteria to find similarities in different revisions.

Typically, text is tokenized, a predefined list of stop words and the wiki markups are ignored, and finally common text between two successive versions is searched with algorithms such as *longest common subsequence* [51]. A challenge in this approach consists in correctly identify paragraphs that have been moved around in different versions. A fine tune is also needed to identify correctly the list of stop words and the minimum length threshold of two sequences of tokens to identify them as the same text.

Our scenario is very different from classical wikis, but we can borrow many things by those approaches with the necessary corrections.

Firstly we do not need to tokenize text because we managed semi-structured data, so we have a minimum unit of meaningful text yet. On the other side, we have to deal with hierarchic data, so a change to an inner node must weigh more than a change to a leaf node.

In order to work, a wiki system needs the assumption that the majority of users are trustworthy. It follows that good content should survive longer than worst one.

The reputation system is modeled starting from this assertion: (i) every user can rate the quality of a document in the form of a "like" or "dislike"; (ii) the reputation of a user depends on the quality of his work, which can be either edit pages or evaluating other users' work; (iii) every contributor of a page should be rewarded in term of reputation for his work in a proportional way of his contribution; (iv) also small edit can heavily change the quality of a page;

Following the classification made in [19] our system is:

- *User-driven*, it relies on ratings provided by users instead of analysis of content and user interactions;

- *Visible to users*, users are aware of it, because the aim is to stimulate their contribution;
- *Weak identity*, users can easily acquire new identity with a new registration;
- *Existence of a ground truth*, since our wiki is designed for semi-structured data when a document contains all information expected by its template, and this data is correct there is no reason to edit the document again unless the template change.
- *Global*, it considers the whole system operating in batch mode at regular intervals.

We decided to make the system user-driven because the existence of a ground truth enables the documents to quickly converge to a stable version. We made some analysis on Wikipedia infoboxes and noticed that they usually do not have great changes between revisions, most of them are quite complete on the first one. Techniques like text survival are pointless in this context because we have very simple text which describes a concept, is not like in Wikipedia where the same concept can be explained in a lot of way and the reputation system must reward the best one. We decided to let users choose between only two ratings (like and dislike) because the purpose of this system is also to help community to find best and worst articles, like Wikipedia with featured and stub pages. So users should express judgment like "this page contains only correct and complete information" or "this page is useless or wrong", we are not interested in interpolation between this two opposite approaches, if a user considers a page as "average", the user simply does not vote for it.

Since a page is written collaboratively, our system has to redistribute among all the authors' page the reputation votes received by the page. This can be done exploiting the fact that a page is a fine-grained semi-structured document: we simply attach proper metadata to every element that form the page that stores who is its author. In this way, knowing who has authored what at a very fine-grained level, we redistribute reputation among different authors. But redistributing reputation is not as trivial as it could seem, because, as previously said, also small edits can change the overall reputation. Consider for example a page contains information about Paris, someone changes the field *Country* from France to Texas; since a city whit this name exists in both of these countries, the page seems valid but contains only wrong data. The system must recognize rapid changes of reputation of a page produced by updates and avoid to decreasing the reputation of old authors. The system should also reward equally good authors which operate on less viewed (and so voted) pages than good authors which operate on popular ones. We want to also avoid that a very small edit on a very popular page can pay better than some big contribution on less viewed pages.

Since also user votes bring information to the system, users' reputation must be affected also by their votes. In this case the system must reward users which rate a page correctly, where "correct" is defined as the common sense of the community. So if the majority of users rate a page as good, the system must increase reputation on users who rate the page good and decrease it on users who rate it as bad. But the system must reward more the "riskier" votes than the votes which follow a well established trend, because the first ones bring new information and furthermore we want to avoid that users gain too much reputation by simply ranking pages.

More formally

We consider the reputation of a user as formed by a set of "units" (or *points*). Then, a user gains a single unit of reputation for every *like* associated with an editorial change the user has made. Analogously, a user loses the same amount for every *dislike* he gets for his editorial changes. This unit must be shared by all the authors of the page with respect to their contributor. For example, consider the revision history of a page as shown in Table 4.1. The page is created by the user A_1, so the revision R_1 (the first version of the page) belongs exclusively to him. Furthermore, at the beginning, the reputation of every user A_i is set to zero.

After some time, the revision R_1 gains 10 *like*s and 1 *dislike*. In this way, the reputation of A_1 increases by 9 units. Afterwards, the page is updated by the user A_2, which edits the 10% of it. The 4 likes gained by this revision are shared for the 10% to A_2 and for the 90% to A_1. So after this update the reputations will be

$$\mathcal{R}(A_1) = 9 + 4 \cdot 0.9$$

$$\mathcal{R}(A_2) = 4 \cdot 0.1$$

	Votes	Contributors			
Revision	(like/dislike)	A_1	A_2	A_3	A_4
R_1	10/1	100%	-	-	-
R_2	4/0	90%	10%	-	-
R_3	20/3	50%	10%	40%	-
R_4	100/5	40%	0%	40%	20%

Table 4.1: Different revisions of the same page with authors contributor and users vote

We define $\mathcal{P}_A(R_i)$ as the contribution of author A to the revision R_i and $\mathcal{L}(R_i)$ the sum of users' votes (likes minus dislikes) for revision R_i. We can

then define the reputation $\mathcal{R}(A)$ gained by an author A from a page R as

$$\mathcal{R}(A) = \sum_{i \in \text{revision of R}} \mathcal{P}_A(R_i) \cdot \mathcal{L}(R_i) \tag{4.1}$$

A problem we encountered and still present in this preliminary version is that the reputation of a user is linked to a page until every part he updated is deleted. In this way, a highly popular edit which survive for a long time span on a very popular page can increase indefinitely the reputation of the corresponding author, even if it is no more active. A solution could be to introduce an 'half-life'-like mechanism that 'weakens' the link between a user and the edits he did, as time passes.

We are aware that the proposed reputation mechanism is very coarse and presents several problems. More precisely: (a) a small, positive edit on a very popular page can excessively increase the reputation of the corresponding user with respect to more active users on less popular pages; (b) it is an oversimplification to assume that a small change in a page yields small changes in how that page will be evaluated later on (that is, the number of *like*s and *dislike*s). Problem (a) can be easily solved surrounding the Formula 4.1 by a sublinear monotonically increasing function. For example, in our preliminary test, we achieved good results whit:

$$\mathcal{R}'(A) = \begin{cases} ln(\mathcal{R}(A) + 1) & \text{if } \mathcal{R}(A) \geqslant 0 \\ -ln(-\mathcal{R}(A) + 1) & \text{otherwise} \end{cases} \tag{4.2}$$

Problem (b) is the idea that after an update, if the first n votes show an inversion of trend, we cut the link to the previous authors for computing the reputation. Supposing the system updates the reputation every day, n can be set as a fraction of average daily votes so it is adaptive as the system grown.

One of the biggest problems is how the system should update reputation when a user restores a previous version of an article. The simpler solution is to not manage this situation: since the system knows both revisions, there is no new information so the reputation should not change. The problem of this approach is that reverting a revision brings little information *"the previous version is better"*, and this information must be rewarded. But how much rewarded? We cannot consider a reverted version like it was written by the user, because we expose the system to a sybil attack: a user can create a fake identity to compromise many documents and restore them with the main identity to gain reputation artificially. To solve this problem our idea is that a revert should be rewarded by the amount of reputation lost by the authors of the reverted version. For example, a user U reverts the last tree versions from authors A_1, A_2 and A_3. Suppose that the reputation gained by them for this updates is $\mathcal{R}(A_1) = 5$, $\mathcal{R}(A_2) = -3$ and $\mathcal{R}(A_3) = -10$ (where a negative value means that they lose reputation). The user U earns

from this revertion $\mathcal{R}(U) = -(5-3-10) = 8$ points. In this way to perform a sybil attack a user must create a fake identity, wait enough to receive bad votes and revert the document. But if he waits too much someone else can revert and gain reputation, if he waits too little he does not earn reputation. Since this system is a zero sum game, it works well also when two or more authors start an edit war reverting each the revision of the others because the system does not create reputation.

4.2 Case of study: Wikipedia

Our preliminary analysis was made on the Italian Wikipedia mainly because it lets us deal with a "small" dataset from an active community. The Italian Wikipedia, in the dump[8] of April 2013, is formed by:

- 1.522.452 encyclopedic articles (usually referred as *namespace 0* pages);
- 41.646.947 total revisions in the namespace 0;
- 179.955 users with at least one contribution in the namespace 0;
- more than 700 GB of complete XML dump.

We now give a brief description of the MediaWiki platform and how it is used by the Wikipedia community, followed by our analysis.

4.2.1 MediaWiki and the Wikipedia community

In order to fully understand the dynamics of Wikipedia, it is essential to have a brief overview about the technology that it uses, the organization (a foundation, namely) that supports it and the ideas of its community.

MediaWiki

Wikimedia Foundation[9] is a US-based charitable organization headquartered in San Francisco, California. Its stated goal is to develop and maintain open content, wiki-based projects and to provide free of charge the full contents of those projects to the public.

MediaWiki[10] is a free and open source wiki software developed mainly by the Wikimedia Foundation. It runs all the Wikimedia wikis and thousands of other websites as well [5]. Even though it offers a quite wide set of features that help the community to organize, grow and preserve the wiki content, a lot of work is still made by hand by the most involved users. The purpose of this section is to make a quick overview about the features that this platform provides out of the box, give all the definitions we need to discuss about wikis – with a particular consideration to Wikipedia – and present the tasks that still lack a support from the platform and so are mostly performed by community handwork.

As easily expected, MediaWiki provides tools to edit the content, represented as an ad-hoc lightweight markup language designed to be easier to use and learn than HTML. It also supports integration with other content

[8] `http://dumps.wikimedia.org/itwiki/`
[9] `https://wikimediafoundation.org/`
[10] `http://www.mediawiki.org/`

different from wiki markup, for example it is possible to enable LaTeX support for mathematical formulas or to read and manage Exif[11] metadata from images. When some functionality is not available by default it is possible to write a corresponding plugin. There is a large number of such plugins[12] ranging from the support to mathematical plotting or to Egyptian hieroglyphs rendering.

For every page MediaWiki saves all the associated revisions, with also some related metadata like the user who made it (if registered), the IP address and the time. It lets also the user to add a short edit summary that helps to read the revision history page. Keeping track of all the revisions helps the community to track the growing of a page and to preserve it from vandalism.

Templates are wiki text block that are designed to be dynamically loaded inside another page. As discussed in the Section 4.1.1, templates are generally used to create infoboxes that are useful to display structured data in a uniform way in different pages, because – thanks to a template processor – it is possible to call templates by passing parameters that can be used to generate the content.

Categories are used to organize the pages. Creating a category is very easy since it is sufficient to add a special tag in the pages that belong to it. Adding these tags automatically creates a link to the bottom of the page that points to an automatically maintained list of all the pages in the selected category.

MediaWiki organizes the content in different namespaces also, which can be viewed as folders that separate different basic types of information or functionality. The main difference between categories and namespaces is that the latter are used to organize meta information. For example there are namespaces for the core articles (usually referred as namespace 0); for user pages, personal pages used by registered users to write their interests or their preferred activity in the community; for help pages used to describe how to use MediaWiki and its tools and many other. In total there are 16 default namespaces. Each namespace is numbered with the convention that content page namespaces have even numbers and their associated talk page namespaces have odd numbers. Talk pages are support pages where community can propose and discuss about sensible changes regarding the relative main page.

The Wikipedia community

The Wikipedia community is formed by normal people who not only read the website, but also contribute to it; they are usually referred as Wikipedi-

[11] `http://en.wikipedia.org/w/index.php?title=Exchangeable_image_file_format&oldid=578374490`

[12] `http://www.mediawiki.org/wiki/Category:Extensions`

ans[13]. There is no barrier to write in Wikipedia, everyone with an internet connection can – potentially – do it. This can be problematic when the community has to deal with harmful users. Overall Wikipedians showed that they are quite good to deal with such kind of users keeping a high quality content. This is done mainly by maintaining a sort of hierarchy of users where everyone, with different roles and rights, helps the project to flourish. The complete list of user rights is quite long[14] because its aim is to create a role for every special need. As some examples consider: the right *CheckUser* which lets the owner to see the list of IP addressed used by an account, this role is intended to control and block potentially harmful users that try to whitewash their reputation creating new accounts; the *File mover* right that allows users experienced in working with files to rename them, the idea is that is more complex to deal with files than text also because they can be saved both in Wikipedia and Wikimedia Commons[15]. For the purpose of this thesis we focus on four different user categories: administrators, unregistered users, bots and autoconfirmed users.

Unregistered users, often refereed as anonymous users, are Wikipedia contributors who are not registered themselves. They can contribute almost as a registered user, the main difference is that they cannot create articles (but they can edit existing ones), upload files or rename pages. It is however possible to semi-protect pages against the edit from anonymous users. The biggest restriction against unregistered users is that their participation to the community life is limited: they cannot vote or be elected as administrators or similar roles (but they can participate in discussions).

Autoconfirmed users are registered users with accounts older than four days who have made at least 10 edits. Since the registration is free and there is no need of any sort of other method for authentication (like providing a real ID or a credit card number), without this limitation an unregistered user could just create a new identity to elude the limitation of his/her status. Since such requirements are very light and do not really impose tight conditions, for simplicity we will refer to autoconfirmed users just as registered users.

Bots are a special category that groups accounts that are connected to some automated or semi-automated software. Since a bot is capable of making a huge amount of edits in a very short time, this category is used to know the active bots and monitor their activity. Everyone can propose a bot and register it as official one, but the process is quite long because it must be proved that the bot is useful and harmless, adheres to the Wikipedia

[13] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Wikipedians&oldid=581988854`

[14] For a complete list see `http://en.wikipedia.org/w/index.php?title=Wikipedia:User_access_levels&oldid=573036838`

[15] A database of freely usable media files, accessible from `http://commons.wikimedia.org/`, that every localized Wikipedia can use out of the box.

policies and performs some task that the community agree must be done but without wasting resources. Generally, bots are used to perform some easy and repetitive task like correcting common typos, update interwiki links[16] or perform bulk updates.

In this thesis we will focus on administrators, sometimes referred as admin or sysops, because it is the group of users with highest rights. Administrators have the ability to block and unblock user accounts and IP addresses from editing, protect and unprotect pages from editing, delete and restore pages, rename pages without restriction, and – in general – they have access to a set of tools that can monitor, control, interfere with the work of other wikipedians. Administrators are not employees of Wikimedia Foundation, they are just volunteers, as all the other Wikipedians. Since the administrator role is so powerful, choosing good candidates is a very sensible problem. There are no official requirements to become an administrator, the only expectation is to have the trust and confidence of the community, so – usually – administrators are users who have a considerable experience and popularity among Wikipedians. The exact rules to become an administrator change in time and in different Wikipedia localizations, but the general requirement is to have a strong consensus of the community; this consensus can be achieved after a formal vote or just a discussion.

The typical procedure to became an administrator is as follows:

1. propose the candidate, another person or oneself, to a special page[17] – formally a *request for adminship*;

2. a discussion is opened for a fixed number of days, where every wikipedian can participate and expose her/his opinions about the candidate;

3. an optional – according to the current Wikipedia rules – vote is proclaimed to gather community consensus, only registered users – and sometimes further constraints are required – can vote supporting, opposing or expressing a neutral opinion about the candidate election;

4. according to the results of the discussion, the optional vote and the current rules, a bureaucrat[18] or an administrator promotes the candidate to admin or closes the request for adminship.

It is also possible to lose the administrator rights and this may happen usually for three different reasons: (i) voluntary removal, (ii) inactivity or (iii) losing the trust of the community. Different Wikipedias measure inactivity in different ways: it can be complete inactivity – i.e. no logins or contribution – or lacking of administrative actions, the time to measure

[16] Links between different language Wikipedias or different Wikimedia projects.

[17] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Requests_for_adminship&oldid=576527268`

[18] A user with the right to change other users' role.

inactivity usually is from some months to a year. The exact procedure to evaluate when or if the administrator has lost the trust of the community depends on the Wikipedia considered, and also inside the same Wikipedia rules change quite frequently. As rule of thumb we can say that the process to dismiss an administrator consist in something similar to:

1. a user reports the admin in some page (usually called *Requests for comment/User conduct* or *Problematic user/administrator*);
2. if there is good evidence of some problem or the support of other users a discussion or a formal vote is opened;
3. analyzing the results of the previous step, and applying the current rules, a bureaucrat may establish that there is no more consensus from the community regarding the admin's work and dismisses him/her or declares the report as an isolated case and close it without any consequence.

Former administrators can usually ask to be reintroduced in their role with another request for adminship. Some Wikipedias require that they did not lose their position due to a lack of trust by the community, while some has faster procedures to reintroduce administrators that were downgraded due to inactivity.

Moreover, some Wikipedias require a recurring verification of administrators, usually annual and with mutual renewal unless someone opposes.

4.2.2 Project proposal

As showed in the previous section, administrators have a very important role in Wikipedia because their additional rights grant them the power to supervise the work of the community. Administrators are in charge to control – in a democratic way – the community, preventing the participation of malicious users and guiding the newer ones.

All this power brings with it the old question *who watches the watchmen?* In the fundamental principles of Wikipedia – referred as Five pillars[19] – the fourth rule is:

> *editors should treat each other with respect and civility*

and in its explanation there are many references on how to obtain consensus[20]. Although democracy may be not directly mentioned, we can see that the community uses typical tools used in a democratic setting (mainly votes

[19] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Five_pillars&oldid=571244060`

[20] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Consensus&oldid=574030593`

and discussion) to achieve it. So we can say that the community itself is responsible to watch the administrators.

The problem is that, even though the community is huge, the active users are not so many and the users that actively participate to discussions and votes are even less because participating them requires a big effort since the user's history on Wikipedia must be manually scrutinized in order to evaluate his/her previous work and involvement in the community.

So promoting users to administrators and downgrading them from their role is not an easy task, and it is easy to find in every Wikipedia some scandals involving malicious administrators; for example administrators who proposed themselves to edit blocked pages for a bribe[21].

The goal of the work presented in this section is to develop a language independent tool, which can be thus applied to the different language specific versions of Wikipedia, to support the election and the evaluation of administrators that is usable in big and small communities. The idea is that in large communities it can support the process freeing human resources that can be used to improve the wiki content; on the other hand, small communities can use it to guarantee the constant monitoring that is essential to maintain an high quality standard of their content.

4.2.3 Wikipedia analysis

The first step of the analysis consisted in comparing the statistics – mainly about number, frequency and type of edits – from Italian Wikipedia with the literature (especially in [71]) to be sure that the dynamics are the same both in the Italian and in the English one.

From this analysis we confirm that the distributions that describe Wikipedia are very asymmetric in almost every aspect. In the Figure 4.4, the log-log graph shows the probability that one page has more than y edits in x days. As we can easily see, the curve shows that the edit frequency of both pages in namespace 0 (from now on called just *pages*) and discussion pages (support pages that are used to discuss how to change an encyclopedic page) follow almost an exponential law, where very few pages have a very high attention from the community, with more than 10 edits per day, and a very long tail of pages that have less than one edit per month.

We can see a similar trend in the Figure 4.5, that shows the registered user activity, excluding the automated bot. In this graph we can see a trend where very few users have made the most edits in Wikipedia.

In Figure 4.6 we analyzed the work of the community in different categories. We divided the community according to the user privileges, so for each graph we can see the activity of administrators, unregistered users, registered users and automated bot. From these graphs we can see that – with

[21] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Requests_for_comment/Paid_editing&oldid=412481447`

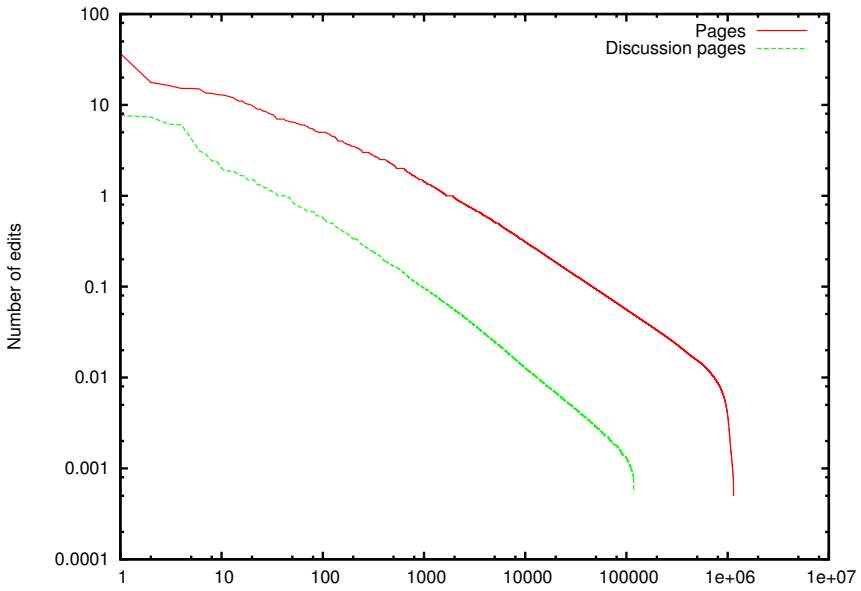


Figure 4.4: Wikipedia edits per day

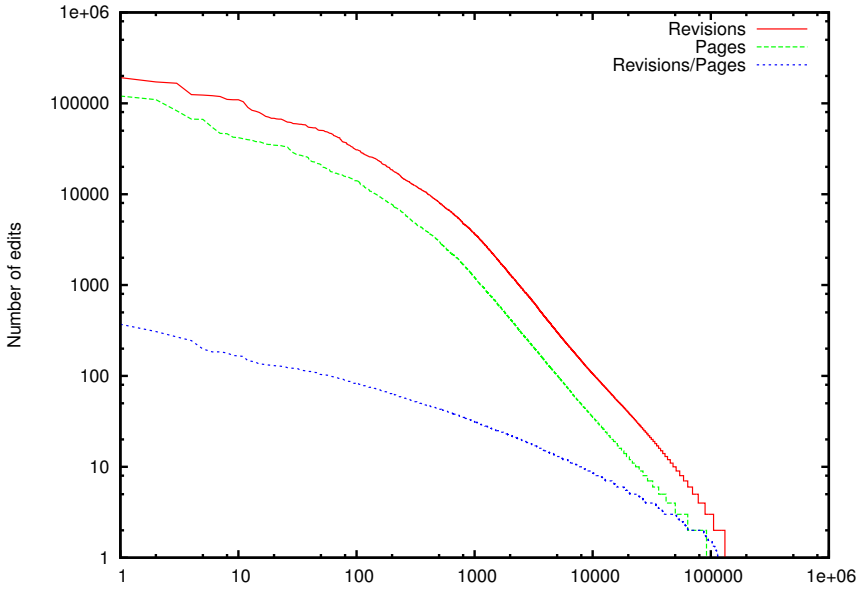
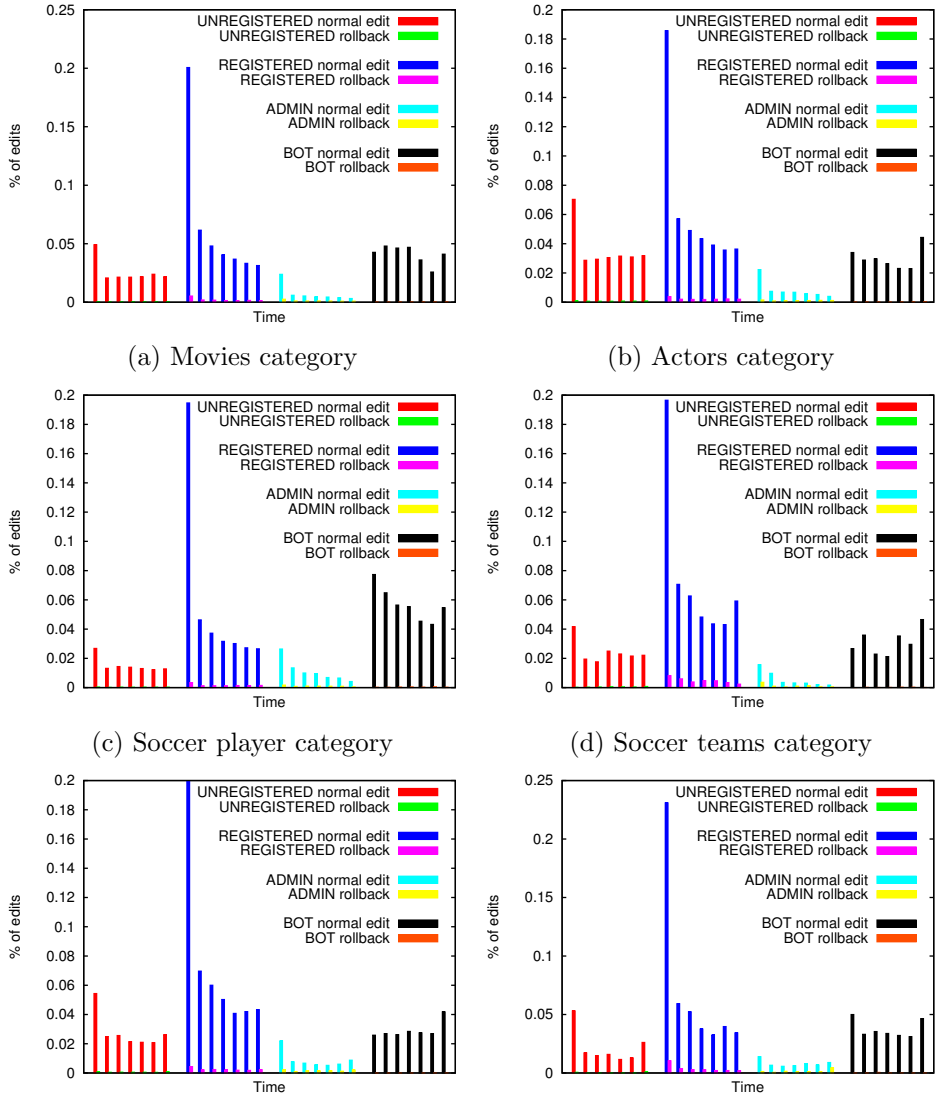


Figure 4.5: Registered user (not bot) activity in Wikipedia

the only exception of bots – usually the activity of the different subcommunities is the same in the analyzed categories. In fact we can see that – although the absolute number of edits is different – the attention that every category receives from Wikipedia editors is the same: if we do not consider bot activity, it is difficult to spot significant differences among these graphs.

To produce these graph we analyzed the edit history of each page in the selected category, and counted the number of edits it received in a fixed number of different time slots of equal length (in the graphs showed here 7 time slots), where the first starts from the page creation and the last ends with its last edit. We computed the percentage of edit each page received in each time slot and aggregated this data per category. We expected to see that some categories stabilize their data faster than others: for example, when the page related to a movie contains the plot, the complete list of the actors and some data about its popularity and awards its information is almost complete and typically will not change, so future edits can just change its presentation but not its content; vice versa we expected that pages related to actors changed more frequently, at least once for every new movie released. Instead, from the graphs we can see that, on average, pages gain a lot of attention at the beginning of their life and tend to stabilize quickly within a small number of edits. The exception in this trend is produced by bots and it is quite easily explained since they are not part of the community but automated scripts that make some sort of repetitive work – like changing the formatting or updating interwiki links – so they do not focus on *most viewed or modified (that is interesting) pages* but just process each page with the same attention. According to this analysis we can say that text survival techniques should weight time in different ways because it is normal that text changes more frequently in the beginning. So a text fragment that survives one day in the first weeks of a page very likely is qualitatively better than a similar fragment that survives some weeks when the page is already in a quite stable version.

The two graphs in Figure 4.7 are intended to show how important it is to elect the right administrator. The pie charts compare the activity in Wikipedia (measured as number of edits) per users group and the cardinality of the groups. As we can see in Figure 4.7a, registered users are responsible of almost half of edits in Wikipedia; bots are very important too and perform about a quarter of the total edits as anonymous users; administrators make a relevant number of edits, beside smaller if compared to the other groups. We are aware that this one is just a rough approximation of the real activity, the point is that any sort of weight would be just a different approximation too: a difference of few characters compared to the previous version does not tell us if the text was just reorganized or heavily rewritten; counting only words (skipping the wiki markup language) would mean do not consider the presentation of the information, but a lot of work is made to present different pages in a coherent way. So we decided to use the



(a) Movies category

(b) Actors category

(c) Soccer player category

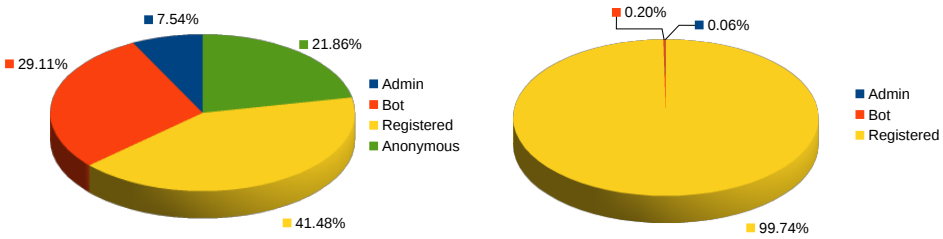
(d) Soccer teams category

(e) Politicians category

(f) Political parties category

Figure 4.6: Number of edits in different Wikipedia categories compared to the page life

simplest approximation, namely counting only the number of edits, without any weight. It is very interesting to compare this graph with the one in Figure 4.7b, in this second we present the number of users per groups. It is important to note that in the latter graph we miss the information about anonymous contributors, this is because – by definition – it is not possible to track them. The only information available is their IP addresses which is not enough to infer any reasonable evaluation. What we can see from this graph is that the registered users outnumber largely both administrators and bots. The administrators are so few that it is impossible to identify their sector in the chart. With the help of these graphs we underline that very few administrators are responsible of a huge amount of edits. It is vital for the community to elect the right administrators because big amount of work depends on this choice.



(a) Number of edits per user group (b) Cardinality of user groups

Figure 4.7: Overview about activity of different user groups of Wikipedia

The histogram in Figure 4.8 analyzes the contributions, in a more exhaustive way, partitioning them in six – not mutual exclusive – categories:

- *edit*: each change in a page that deletes, adds or reorders text creating a new revision that did not exist in the history of the page itself;
- *rollback*: the act of restoring a previous revision of a page;
- *rolled back*: any revision that will be overwritten by a future rollback;
- *restored*: a revision that will be restored by a future rollback;
- *deleted*: a revision that is deleted (hidden to non administrators) from the history, usually because it contains some text that is copyrighted and thus incompatible with the license used by Wikipedia;
- *war*: a revision that is marked both rolled back and restored, this means that it was reverted by someone and later restored by someone else (this is a good approximation to detect edit wars).

In this graph we can see that different groups act in a very different way: for example administrators edit very few pages that will be deleted compared

to the other categories and perform more rollbacks with respect to normal edits as anyone else.

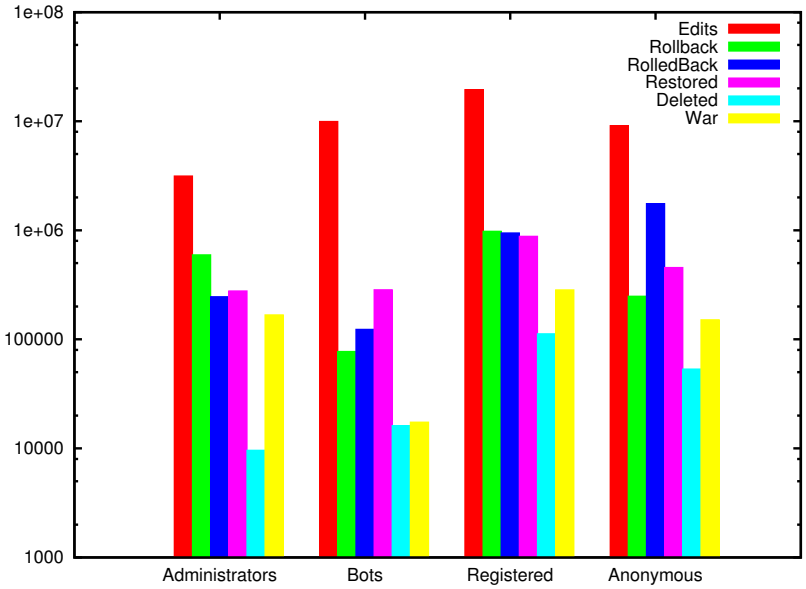
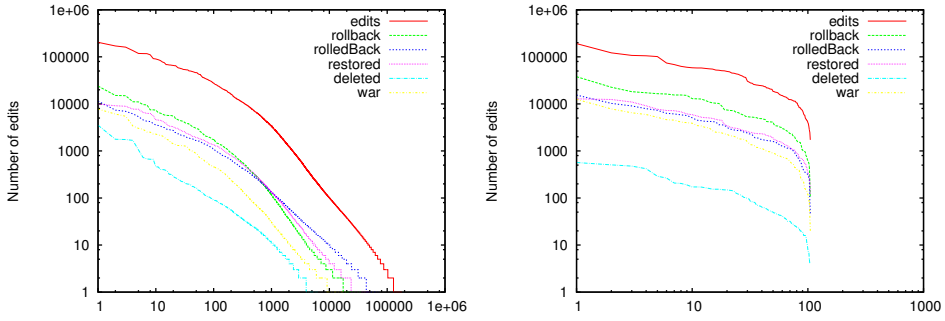


Figure 4.8: Different activity per user group in Wikipedia

The graphs in Figure 4.9 show the same data of the previous one but in a cumulative distribution function and only for administrators and registered users. We can see that the administrator contributions is more regular and does not present the long tail of a lot of users that perform few edits that is very visible in the registered users group.



(a) Registered users activity

(b) Administrators activity

Figure 4.9: Registered users' activity compared to administrators' one

4.2.4 Results

In this analysis we showed that dynamics in Italian Wikipedia are comparable – in smaller scale – with the English one. This emphasizes the idea that some dynamics – for example the difference between administrators and normal editors – are language independent leading to the concept that reputation systems should be developed starting from a language-independent analysis tool.

The analysis shows also that the community has found a quite robust and well trained way to control and administrate itself. Unfortunately, especially in smaller Wikipedias like the Italian one – it is easy to see the community discusses the lack of administrators or the difficulty to efficiently monitor all the aspects of the project. In this section we showed that is possible to discern normal editors and administrators observing statistics about their edits only, so it is possible to use these statistics to support the administrators' election and have a hint of their work quality.

4.2.5 Reputation system architecture

Considering a standard wiki architecture – composed by one web server and one database server – a reputation system which takes into account all the aspects discussed could be structured in three modules, as showed in Figure 4.10: (i) a page viewer counter, (ii) a reputation collector and (iii) an offline analyzer.

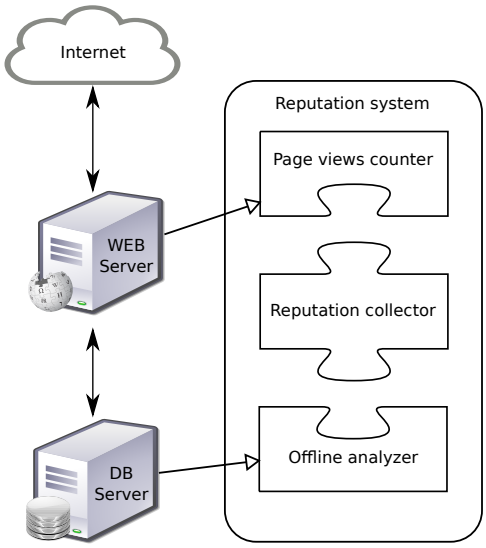


Figure 4.10: Wiki reputation system architecture

The page viewer counter is a module that analyzes data from the web server and counts how many users see each revision of each page. It is useful

to have a measure of the "time lapse" between two page revisions which is independent from the popularity of the page itself.

The offline analyzer has direct access to the database (for performance reasons) and performs batch analysis on the wiki pages. This module is responsible to analyze the difference between two consecutive revisions and track the editors work. How often this process executes depends on the real context: the computation can be very heavy for popular wikis (like Wikipedia) but could be optimized by doing incremental computation on previous data. Ideally, to have better and fresher reputation values, this process must execute very often, like daily. To reduce the required computational power, we can add extra functionality to this module to identify the most sensitive data and trigger updates on it more frequently. For example it is more important to have a very fresh administrators' reputation than normal users' one.

The reputation collector is the module that aggregates information provided by the previous two to compute the final reputation value. Having access to both edit time and page views it can weight correctly the impact of each edit, and thereon compute the reputation according to its impact as seen by the community.

A system like this could provide both reputation to users and quality of pages (as we will discuss better in Section 5.1). We identified different – but not mutually exclusive – scenarios that could take advantage of this data: (i) user reputation, (ii) page quality, (iii) edits monitoring and (iv) access control.

In the first scenario the system provides only users' reputation. This is the case we originally analyzed and the system could be used to support administrators elections. Moreover, the system computes reputation of every user so this value could be used to create a sort of reward badges to display on most active users' personal pages. This triggers a gamification process that could stimulate users to participate more actively in the community life and, doing so, improve the wiki content.

User reputation can be used to infer page quality: if a page is reviewed by high reputation users the probability that the page is well structured and contains good data is high. So it is possible to use editors' reputation to compute an index that estimates the page quality. This index could be useful in different way: if it is shown on the page itself new users could consider it to have a better idea about how much to trust the page content; on the other hand, the community could use it to rank the worst pages and concentrate the effort to improve them or rank the best pages to select easily featured articles to put in home page.

Manually reviewing recent changes is a normal task in Wikipedia: some voluntary editors constantly look at the *recent changes* page to identify –

and revert – harmful edits; this process is called *recent changes patrol*[22]. A reputation system that computes both users' reputation and pages' quality, could mix these information and raise a warning when a user edits a page reviewed by higher reputation users. This helps patrollers to focus their effort better.

Eventually, extending the previous proposal to a radical approach, the wiki could refuse an edit made by a user who have the reputation lower than the quality of the page he/she is willing to modify. This approach creates an access control policy which describes, in an adaptive way, who has the right to change what portion of wiki content. This last point could be highly debated because it is against one of the Wikipedia fundamental principles

> *Wikipedia is free content that anyone can edit, use, modify, and distribute*[23]

since it actually creates a barrier for new users: they have to increase somehow their reputation (i.e. contributing on talk pages) before being able to modify some pages and they have to be registered too (because it is impossible to track the reputation of an anonymous user). But we believe that this approach could be useful for other wikis that have stricter approaches regarding edit policies: for example a lot of open source projects use wikis for their documentation that already accept contributions only from registered users. In this case an access control policy that guarantees edits only from high reputation users could help to maintain an high quality documentation wiki.

[22] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Recent_changes_patrol&oldid=582849929`

[23] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Wikipedia_is_free_content&oldid=581571571`

Chapter 5

Conclusions

In this thesis we proposed novel approaches to reputation systems and we focused on showing how this name groups a vast variety of very different technologies and models together.

We are confident that a lot of research will be made in this field in the near future: online interactions are becoming more important and frequent in the everyday life of everyone, reputation systems are a good tool to support decisions about future works.

Furthermore collaborative platforms to share knowledge are growing in popularity and – as we shown – automatically evaluating the quality of text is not a trivial task. Furthermore evaluating the quality of some more complex data – like maps of OpenStreetMap[1] or Waze[2] – requires new and fundamentally different approaches.

We will conclude this thesis showing how and when it is possible to modify and adapt a reputation system as a recommendation system, we will show also that in particular domains the difference between them is so small that in literature they are often mixed together and, eventually, we will close with a short discussion to summarize each scenario we presented in this work.

5.1 Discussion: similarity with recommender systems

At a first sight, recommender and reputation systems appear to have very different goals and so they seem to share almost nothing in common. The purpose of this chapter is to show their difference and similarity and how, when used in some particular domains, they are just two different approaches to solving the same problem.

[1] `http://www.openstreetmap.org/`
[2] `https://www.waze.com/`

We can define a recommender system by the goal that it is supposed to reach: generate meaningful recommendations to a collection of users for items or products that might interest them. Its actual design is strictly dependent on the domain and the particular characteristics of the data available.

Recommender systems are widely used in websites, especially in online marketplace. For example Amazon[3] uses a recommender system to suggest to buyers interesting products based on their previously watched or bought items; eBay[4] has a feedback mechanism to highlight best shops; Google itself provides a recommender system – Personalized Search[5] – that uses users' search history and information from their personal account to provide better results.

According to the categorization made in [48], recommender systems can be classified in two main categories: content-based recommending and collaborative filtering.

Content-based recommending systems – sometimes referred as content-based filtering – use information about users and items to perform recommendations. For example, if a user likes science fiction movies – according to what is declared in his/her profile – the recommender system can use this information to suggest him/her some popular science fiction movies. Even better, also if there is no information in the profile, but the user has liked *Star Wars* and *2001: A Space Odyssey*, the system can infer that he/she likes science fiction movies and suggest them.

Collaborative filtering systems work collecting user feedback and looking for similarities in their rating behavior to determine how to recommend items. Following the previous example, if a user has liked *The Lord of the Rings* the system can suggest *The Hobbit* not because it knows that both of them are fantasy movies or that the user likes that sort of movies, but because it sees that these two movies are usually liked together by other users.

As it is easy to imagine, these groups are just the two extreme different kinds of recommender systems; a system can be a hybrid, leveraging on the strong aspects of both the categories.

In our context of team formation system (as described in Section 3.1) it is difficult to spot similarity with a recommendation system: the proposed system is designed to compute users' reputation and to compose a team according to the required skills. From this point it is difficult to imagine that the reputation can be used to compute the quality of the work performed and use it to rank or aggregate the final results. This is mainly because we do not have any extra information about the work proposed: it can

[3] `http://www.amazon.com/`

[4] `http://www.ebay.com/`

[5] `http://googleblog.blogspot.it/2005/06/search-gets-personal.html`

be anything, also if we constrain the system to a particular domain – for example a musician reputation system – we do not know if the purpose of the team is to make some new and experimental music, compose a traditional song or just play an existing composition.

One may object that information like the director of a movie is successfully used to recommend movies to a user and the director's reputation is considered a quite good indicator of the quality of the movie itself. But our system is different because it does not assume that the work is made to be sold: a movie is planned with the idea to sell it, so a target audience is analyzed and used to design the plot; our model is more suitable for small teams that develop something like a short film. This scenario is closer to the YouTube style – with a lot of people working on very different projects for passion – than the Hollywood one – where a relatively small number of very high reputation people work in a well known way. It is easy to see that the high competition, the different styles that the same author can achieve in the first – mostly amateur – model and the usually short life cycle of products lead to a very fragmented market: a recommendation system like the YouTube one [18] cannot make meaningful recommendations based only on connections among users, nor suggest only videos published from the liked authors.

We can do similar considerations for the service composition system described in Section 3.2: it is easy to guess that there is no meaning in recommending a service only because it uses high reputation infrastructures. We can just infer something about its availability: a website hosted in some high reputation farm is more willing to be always online than a website hosted in a very low level farm, but there is no reason to expect that the high reputation farm hosts only good websites.

But if we change perspective, we can consider also these systems as a strange sort of recommendation systems that, instead of recommending items, recommend people or services. The main difference is that traditional recommendation systems are made to push suggestions to the user: Netflix or Amazon suggest day by day items to their users because it makes their business; the model proposed is a pull system, where a user requires for a skill or services and the reputation system ranks a set of recommendations. From this point of view we can consider the two models proposed as collaborative filtering systems, where recommendations are made according to what users – coworkers or services – and their neighbors liked in the past.

Differences between recommender and reputation systems become so minor to have just a very blurred border between them when we deal with wiki systems. As already seen in Section 2.3, literature often does not differentiate between them. The idea here is that people share a common goal – improve and increase the information stored on the wiki – so high reputation users will contribute with good content and good content is usually made or reviewed by high reputation users. Moreover, a recommendation system in

a wiki context has a slightly different definition of *interested item*: in traditional recommendation systems, the recommendations are made to suggest something tailored to the user interests mainly because they are used to suggest items to buy. An approach like this is not very meaningful in a wiki environment, firstly it is not useful for the wiki business, secondly a user can find related interesting topics simply navigating across the links.

In a wiki recommender system, an interesting item is a page that is qualitatively above the average, something that the community is proud to show to the newcomers and that is used to encourage people to trust the wiki and the community itself.

5.2 Team formation

We presented in Section 3.1 new reputation-based algorithms for team recommendation and formation. Such algorithms maximize the overall reputation between the team members. We empirically validate the algorithms on synthetic data. Such data have been generated in a way that the obtained reputation graphs are similar to real-world social graphs. With the term "similar" we mean that the generated and real-world graphs present characteristics like node connectivity, distribution of number of connections, that follow closely those of real-world graphs.

We are currently implementing the algorithms as module of a web-application for team recommendation [8]. The reason to do that is twofold. First, to use the application to collect real-world team data that are required to further validate the proposed algorithms. Second, using crowdsourcing techniques, we will be able to extend the algorithms in order to include a feedback mechanism. More precisely, we are working on the possibility to extend the algorithms such that the result of one execution will update the social graph to provide better results upon subsequent executions. The final goal of this research is to define a proper heuristic for limiting the search space and speedup the team formation algorithms with the minimum loss of quality, since we are aware that the current approach becomes very slow when the graph increase in size.

5.3 Service composition

In Section 3.2 we showed how it is possible to use the transitive closure of trust in another – and more complex – context than the one exposed in Section 3.1.

We believe that this is a powerful approach to the problem of service composition and can be extended to handle more difficult scenarios as well. For example, including the extension that handles reputation providers – as entity who provide reputation but no services – in the team formation

example we can describe, with minor modification only, some new and interesting case studies such as: represent individuals that we trust for their opinion but – for some reason – not for their work, like a very clever surgeon now retired; or, as the opposite, represent people we trust their work but not their recommendation, mainly because they are overenthusiastic about other people's work.

We are also aware that this approach is computationally very demanding, as seen in Section 3.2.4, and not very suitable for big environment like a world wide service composition scenario. We think that further research could be made to decentralize the reputation computation to compute it in a p2p fashion by each node in the graph. In this scenario new and interesting problems arise mainly related to the fact that, without the centralized reputation manager, security checks must be added to be sure that each node truthfully reveals the reputation of its neighbors without altering it using a sybil attack.

5.4 Semi-structured data wiki

Although we are aware that the framework presented in Section 4.1 is still under development, the first experimental results show that updates of templates associated with small – yet meaningful – sets of documents execute in reasonable time; still, there is ample space for optimizing the proposed procedures. The optimization process may involve as well a phase of logical redesign of large XML schemas into smaller, more manageable ones. Along with optimization issues, we are dealing with the realization of a suitable web application that allows users to easily interact with the repository.

We showed also how to attach a reputation system to this kind of wikis and how we can use this particular kind of data to have a more fine grained analysis. Since the system is quite complex already, there are many open problems which must be investigate before it can be usable.

We are confident that such kind of wiki will see some practical usage in the next years because it solves a lot of tricky problem in some well-defined context; for example Wikispecies[6] – a Wikimedia project whose aim is to create a catalogue of all living species – a semi-structured wiki would fit better instead of a classical one.

Another very complex open problem is how to evaluate a template update. Since a template describes the structure of many documents, a high-reputation work on a template should reward more than a high-reputation work on a document. The problem is to decide how much. The first answer is that it should reward more proportionally on how many documents use this template. But in this trivial solution we do not address the problem that a normal user does not see templates and does not vote it. Unfortunately

[6] `http://species.wikimedia.org/`

evaluating the work on a template is even more difficult than evaluating the work on a document, also because it is strictly related to the operation allowed on it.

Therefore it is even more difficult to propose reputation system models that deal with it because, without real data, is difficult to infer what operations will be more used or more required by the community and create a model that considers all the possible operations that could be performed on a template is troublesome, weigh them correctly in a reputation system is even worst.

5.5 Wikipedia reputation system

In Section 4.2 we presented several statistics regarding Italian Wikipedia that show the Italian community has dynamics similar to the English one. We showed also that the behavior of administrators is different from normal editors, also considering only the most active editors. This is because administrators have a special role in the community: they are not considered only as advanced contributors, but as supervisors that control and address the work of the community in a peaceful and constructive way. Their work consists also of supervising newer editors, control and limit problematic users and bring consensus in edit wars.

Also considering that most of this work is made editing special pages and using administrator tools, we showed that is somehow reflected in analyzing their edit history – mainly seeing they have higher ratio of rollback per edit – and so we argued that is possible to create a language independent history based reputation system for Wikipedia, and – plausibly – for other wikis too.

Bibliography

[1] Lada A. Adamic and Bernardo A. Huberman. Power-Law Distribution of the World Wide Web. *Science*, 287(5461):2115, March 2000.

[2] B Thomas Adler and Luca De Alfaro. A content-driven reputation system for the wikipedia. 2006.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[4] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science (New York, N.Y.)*, 286(5439):509–512, October 1999.

[5] Daniel J. Barrett. *MediaWiki - Wikipedia and beyond.* O'Reilly, 2009.

[6] Anders Berglund, Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML path language (XPath) 2.0 (second edition). W3C recommendation, W3C, December 2010.

[7] Loris Bozzato and Mauro Ferrari. Composition of semantic web services in a constructive description logic. In *Web Reasoning and Rule Systems*, pages 223–226. Springer, 2010.

[8] Stefano Braghin, Jackson Tan Teck Yong, Anthony Ventresque, and Anwitaman Datta. SWAT: Social Web Application for Team Recommendation. In *Proceedings of IEEE International Workshop on Scalable Computing for Big Data Analytics (SC-BDA)*, 2012.

[9] Peter Buneman, James Cheney, Sam Lindley, and Heiko Müller. The database wiki project: A general-purpose platform for data curation and collaboration. *SIGMOD Record*, 40(3):15–20, 2011.

[10] Barbara Carminati, Elena Ferrari, and Patrick CK Hung. Security conscious web service composition. In *Web Services, 2006. ICWS'06. International Conference on*, pages 489–496. IEEE, 2006.

[11] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), June 2006.

[12] Don Chamberlin, Jonathan Robie, Daniela Florescu, Jim Melton, Jérôme Siméon, and Michael Dyck. XQuery update facility 1.0. Candidate recommendation, W3C, June 2009. http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/.

[13] Don Chamberlin, Jonathan Robie, Daniela Florescu, Jim Melton, Jérôme Siméon, and Michael Dyck. XQuery update facility 1.0. Candidate recommendation, W3C, June 2009. http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/.

[14] Meenal Chhabra, Sanmay Das, and Boleslaw Szymanski. Team formation in social networks. In *Proceedings of the 27th International Symposium on computer and Information Sciences*, 2012.

[15] Bruce Christianson and William S. Harbison. Why isn't trust transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176, London, UK, UK, 1997. Springer-Verlag.

[16] James Clark. XSL transformations (XSLT) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[17] Sanmay Das, Allen Lavoie, and Malik Magdon-Ismail. Manipulation among the arbiters of collective intelligence: How wikipedia administrators mold public opinion.

[18] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 293–296, New York, NY, USA, 2010. ACM.

[19] Luca De Alfaro, Ashutosh Kulshreshtha, Ian Pye, and B Thomas Adler. Reputation systems for open collaboration. *Communications of the ACM*, 54(8):81–87, 2011.

[20] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, and Katia Sycara. Security for daml web services: Annotation and matchmaking. In *The Semantic Web-ISWC 2003*, pages 335–350. Springer, 2003.

[21] AnHai Doan. Data quality challenges in community systems. In *QDB*, page 9, 2007.

[22] AnHai Doan, Raghu Ramakrishnan, Fei Chen, Pedro DeRose, Yoonkyong Lee, Robert McCann, Mayssam Sayyadian, and Warren Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64–72, 2006.

[23] AnHai Doan, Raghu Ramakrishnan, and Alon Y Halevy. Mass collaboration systems on the world-wide web. *Communications of the ACM*, 15(2):196–216, 2010.

[24] David Eppstein and Joseph Wang. A steady state model for graph power laws. In *INTERNATIONAL WORKSHOP ON WEB DYNAMICS*, 2002.

[25] Dan Frankowski, Shyong K. Lam, Shilad Sen, F. Maxwell Harper, Scott Yilek, Michael Cassano, and John Riedl. Recommenders everywhere: : the wikilens community-maintained recommender system. In Alain Désilets and Robert Biddle, editors, *Int. Sym. Wikis*, pages 47–60. ACM, 2007.

[26] Eric Friedman, Paul Resnick, and Rahul Sami. Manipulation-resistant reputation systems. *Algorithmic Game Theory*, pages 677–697, 2007.

[27] Florent Garcin, Boi Faltings, and Radu Jurca. Aggregating reputation feedback. In *Proceedings of the First International Conference on Reputation: Theory and Technology, Gargonza, Italy*, volume 1, pages 62–67, 2009.

[28] Jim Giles. Internet Encyclopedias Go Head to Head. *Nature*, 438:900–901, 2005.

[29] Markus Glaser, Richard Heigl, and Alexander Warta. *Wiki: Web Collaboration.* Springer, 2008.

[30] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *Commun. Surveys Tuts.*, 3(4):2–16, October 2000.

[31] Giovanna Guerrini, Marco Mesiti, and Daniele Rossi. Impact of xml schema evolution on valid documents. In Angela Bonifati and Dongwon Lee, editors, *WIDM*, pages 39–44. ACM, 2005.

[32] Dominique Guinard. *A web of things application architecture: integrating the real-world into the web.* PhD thesis, 2011.

[33] Shaili Jain, Yiling Chen, and David C Parkes. Designing incentives for online question-and-answer forums. *Games and Economic Behavior*, 2012.

[34] R. Jelliffe. Schematron. Web page, October 2000. http://www.ascc.net/xml/resource/schematron/.

[35] Audun Jøsang, Ross Hayward, and Simon Pope. Trust network analysis with subjective logic. In *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ACSC '06, pages 85–94, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[36] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[37] Audun Jøsang and Simon Pope. Semantic constraints for trust transitivity. In *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling - Volume 43*, APCCM '05, pages 59–68, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[38] Roula Karam and Michele Melchiori. Improving geo-spatial linked data with the wisdom of the crowds. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 68–74. ACM, 2013.

[39] Ahmad Kardan, Amin Omidvar, and Farzad Farahmandnia. Expert finding on social network with link analysis approach. In *Electrical Engineering (ICEE), 2011 19th Iranian Conference on*, pages 1 –6, may 2011.

[40] Przemysław Kazienko, Katarzyna Musiał, and Tomasz Kajdanowicz. Multidimensional Social Network in the Social Recommender System. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 41(4):746–759, July 2011.

[41] Meike Klettke. Conceptual xml schema evolution - the codex approach for design and redesign. In Matthias Jarke, Thomas Seidl, Christoph Quix, David Kensche, Stefan Conrad, Erhard Rahm, Ralf Klamma, Harald Kosch, Michael Granitzer, Sven Apel, Marko Rosenmüller, Gunter Saake, and Olaf Spinczyk, editors, *BTW Workshops*, pages 53–63. Verlagshaus Mainz, Aachen, 2007.

[42] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *KDD*, pages 467–476. ACM, 2009.

[43] Shalil Majithia, Ali Shaikh Ali, Omer F Rana, and David W Walker. Reputation-based semantic service discovery. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004. 13th IEEE International Workshops on*, pages 297–302. IEEE, 2004.

[44] Ashok Malhotra and Paul V. Biron. XML schema part 2: Datatypes second edition. W3C recommendation, W3C, oct 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[45] Iulia Maries and Emil Scarlat. Modeling trust and reputation within communities of practice. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 2192–2199. IEEE, 2010.

[46] Paolo Massa and Paolo Avesani. Trust-aware bootstrapping of recommender systems. In *Proceedings of ECAI 2006 Workshop on Recommender Systems*, pages 29–33, 2006.

[47] Judith Masthoff. Group recommender systems: Combining individual models. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 677–702. Springer, 2011.

[48] Prem Melville and Vikas Sindhwani. Recommender systems. *Encyclopedia of machine learning*, 1:829–838, 2010.

[49] Nicola Mezzetti. A socially inspired reputation model. In *Public Key Infrastructure*, pages 191–204. Springer, 2004.

[50] Sudip Misra and Ankur Vaish. Reputation-based role assignment for role-based access control in wireless sensor networks. *Computer Communications*, 34(3):281–294, 2011.

[51] Eugene W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[52] Surya Nepal, Sanat Kumar Bista, and Cécile Paris. An association based approach to propagate social trust in social networks. In *UMAP Workshops*, 2012.

[53] Surya Nepal, Wanita Sherchan, and Cecile Paris. Building trust communities using social trust. In Liliana Ardissono and Tsvi Kuflik, editors, *Advances in User Modeling*, volume 7138 of *Lecture Notes in Computer Science*, pages 243–255. Springer Berlin Heidelberg, 2012.

[54] Eva Tardos Noam Nisan, Tim Roughgarden and Vijay V. Vazirani. *Algorithmic Game Theory*, chapter 27. Cambridge University Press, 2007.

[55] National Research Council (U.S.). Committee on the Fundamentals of Computer Science: Challenges and Opportunities. *Computer Science: Reflections on the Field, Reflections from the Field.* National Academies Press, 2004.

[56] Luiz Pizzato, Tomasz Rej, Joshua Akehurst, Irena Koprinska, Kalina Yacef, and Judy Kay. Recommending people to people: the nature of reciprocal recommenders with a case study in online dating. *User Modeling and User-Adapted Interaction*, pages 1–42, 2012.

[57] Luiz Pizzato, Tomek Rej, Thomas Chung, Irena Koprinska, and Judy Kay. Recon: a reciprocal recommender for online dating. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 207–214, New York, NY, USA, 2010. ACM.

[58] Luiz Augusto Sangoi Pizzato, Tomek Rej, Kalina Yacef, Irena Koprinska, and Judy Kay. Finding someone you will like and who won't reject you. In *UMAP*, pages 269–280, 2011.

[59] Reid Priedhorsky, Jilin Chen, Shyong K. Lam, Katherine A. Panciera, Loren G. Terveen, and John Riedl. Creating, destroying, and restoring value in wikipedia. In *GROUP*, pages 259–268, 2007.

[60] Steven Reece, Stephen Roberts, Alex Rogers, and Nicholas R. Jennings. A multi-dimensional trust model for heterogeneous contract observations. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, AAAI'07, pages 128–135. AAAI Press, 2007.

[61] Steven Reece, Alex Rogers, Stephen Roberts, and Nicholas R. Jennings. Rumours and reputation: evaluating multi-dimensional trust within a decentralised reputation system. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 165:1–165:8, New York, NY, USA, 2007. ACM.

[62] Mikalai Sabel. Structuring wiki revision history. In *Proceedings of the 2007 international symposium on Wikis*, WikiSym '07, pages 125–130, New York, NY, USA, 2007. ACM.

[63] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system – a case study. In *IN ACM WEBKDD WORKSHOP*, 2000.

[64] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 2010.

[65] Jason Swarts. The collaborative construction of fact on wikipedia. In *Proceedings of the 27th ACM international conference on Design of communication*, pages 281–288. ACM, 2009.

[66] Henry S. Thompson, Murray Maloney, David Beech, and Noah Mendelsohn. XML schema part 1: Structures second edition. W3C recommendation, W3C, oct 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[67] Vlad Mihai Trifa. *Building blocks for a participatory Web of things.* PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 19890, 2011, 2011.

[68] Eric van der Vlist. Examplotron. Technical report, 2003. http://examplotron.org/.

[69] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, M Eisenhauer, et al. Internet of things strategic research roadmap. *O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, et al., Internet of Things: Global Technological and Societal Trends*, pages 9–52, 2011.

[70] Adam Wierzbicki, Piotr Turek, and Radoslaw Nielek. Learning about team collaboration from wikipedia edit history. In *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*, page 27. ACM, 2010.

[71] Taha Yasseri, Robert Sumi, András Rung, András Kornai, and János Kertész. Dynamics of conflicts in wikipedia. *PloS one*, 7(6):e38869, 2012.

[72] Honglei Zeng, Maher Alhossaini, Li Ding, Richard Fikes, and Deborah L. Mcguinness. Computing trust from revision history. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. Proceedings of the 2006 International Conference on Privacy, Security and Trust, 2006.

[73] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421. ACM, 2003.

[74] Xiaoqing Zheng, Zhaohui Wu, Huajun Chen, and Yuxin Mao. Developing a composite trust model for multi-agent systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 1257–1259, New York, NY, USA, 2006. ACM.