Universidade do Minho
Escola de Engenharia

Francisco Miguel Carvalho Barros da Cruz

SocialSeer

Novembro 2009

Universidade do Minho
Escola de Engenharia

Francisco Miguel Carvalho Barros da Cruz

SocialSeer

Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor Rui Carlos Mendes de Oliveira

Novembro 2009

Acknowledgements

To my advisor Prof. Rui Carlos Oliveira, for his support and motivation skills over the months that this thesis has been gestation. His good humour and careful attention to details greatly eased the creation of this project.

To Prof. José Orlando Pereira for his help and enlightening conversations.

I am in debt to Francisco Maia and João Paulo, who generously stepped up to carefully follow all the course of this work.

I would like to thank all my fellow colleagues at the *Grupo de Sistemas Distribuídos* for the cosy environment and especially to Nuno Carvalho, for his support and availability to all my, sometimes not so bright doubts.

To João Granja, Nuno Oliveira, Carlos Pereira, Miguel Pires and Fábio Lima for all the unwounded lunches.

I also want to thank my family for the patience and constant support and encouragement during my involvement with this work.

Resumo

A ubiquidade do actual acesso banda larga à Internet e o uso crescente de diversos e diferentes terminais e tipos de acesso tais como, o computador pessoal, o computador portátil e o telemóvel, significa que estamos expostos a uma enorme quantidade de informação onde quer que estejamos e, portanto, precisamos de mecanismos que ajudem a lidar com a sobrecarga de informação.

Metadados são, etilogicamente, dados sobre os dados, estes podem ser extraídos dos conteúdos dos ficheiros, podendo ser utilizados como base para a organização da informação, tornando-se assim premente que o utilizador possa catalogar os seus dados sobre uma perspectiva pessoal, através de palavras-chave e mesmo, de uma apreciação quantitativa. Com base nos metadados e no padrão de uso do utilizador podemos criar formas de apresentar ao mesmo informação concordante com as suas preferências pessoais.

Sendo que a componente humana social tem um papel preponderante na vida quotidiana é inevitável a partilha de dados e opiniões entre um círculo de amigos, que muito provavelmente têm um elevado índice de sobreposição de preferências.

Assim torna-se clara a importância e necessidade de uma plataforma baseada em metadados, que integre a web no nosso ambiente de trabalho, permitindo a sua interconexão e ajudando o utilizador a melhor lidar com a informação ao seu dispôr.

Abstract

The current ubiquity of broadband access to the Internet and the increasing use of several and different types of terminals of access such as personal computer, laptop computer and mobile phone means that we are exposed to an enormous amount of information wherever we are, and therefore, we need mechanisms that help us to cope with the information overload.

Metadata, literally data about data, can be extracted from the file contents and can be used as a basis for organizing information. Furthermore, metadata gives way to cataloguing our data on a personal perspective, through keywords and even through a quantitative assessment. Based on metadata and the usage pattern, we can create ways of presenting information to the user according to her personal preferences.

The human-social component has a central role in our everyday lives, thus there is the need of sharing data and opinions within our circle of friends, who probably have a high degree of overlap of preferences.

To ease this behavior we identified the need for a metadata-based platform that blends the web into our working set of documents, enabling the interconnection of the user's desktop's and at the same time helping the user to better deal with the information available. With this work we present our approach to implementing such a platform.

Contents

Contents . . . viii

List of Figures . . . ix

List of Tables . . . xi

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Thesis contribution . . . 5

1.3 Dissertation outline . . . 6

2 Related work **7**

2.1 Metadata . . . 7

2.1.1 Metadata in the Web . . . 7

2.1.2 Desktop metadata indexing . . . 8

2.2 Personal recommendation systems . . . 10

2.3 Social networking . . . 12

2.4 Undergoing projects . . . 13

2.4.1 Nepomuk . . . 14

2.4.2 Social desktop . . . 15

2.4.3 Social desktop – Microsoft Research . . . 15

3 SocialSeer **17**

3.1 Architecture . . . 18

3.2 Prototype 19
3.2.1 Bolt and SocialSeer 20
3.2.2 Metadata: gathering and manipulation 21
3.2.3 Personal metadata indexing 26
3.2.4 Dynamic user profile 30
3.2.5 Circle of friends 31
3.2.6 Sharing of document rates 33
3.2.7 Document recommendations 35
3.2.8 In-memory index holder 39
3.2.9 Photo album synchronization 40
3.2.10 Implementation considerations 41

4 Experimental evaluation **43**
4.1 Experimental scenario description 44
4.2 Load test results 46
4.3 Latency test results 48

5 Conclusion **51**
5.1 Conclusions 51
5.2 Contributions 53
5.3 Future work 54

References **56**

List of Figures

3.1 Bolt's isolation level 20

3.2 SocialSeer's isolation level 21

3.3 Introduced modules in Bolt's client application 22

3.4 Indexing process in SocialSeer's server 27

3.5 KMDKind index 29

3.6 Dynamic user profile 32

3.7 User's social graph retrieving process in SocialSeer's server 33

3.8 Example of a multimedia document rate sharing . . . 35

3.9 Automatic recommendations algorithm example, where $S_{total} = 5$ 38

3.10 *In-memory Index Holder* and how it relates to other modules 41

4.1 CPU usage 47

4.2 Total of memory used in MB 47

List of Tables

4.1 Latency table 49

Chapter 1

Introduction

In this introductory chapter we present the motivation that led to this dissertation. In addition, we try to describe its context and its relevance towards the current IT scenario. Finally, we present the dissertation outline.

1.1 Motivation

Nowadays, the Internet plays a very important role in our daily lives. Every day we surf the web visiting our favourite websites, sending and receiving emails, downloading and uploading files. As a result, we are exposed to an enormous amount of information which flows up and downstream but asymmetrically. Invariably, the amount of downloaded information is much larger than the one that is uploaded and, as a consequence, the data to be stored is growing rapidly and most of the times in an unstructured fashion. In place, informed and unobtrusive mechanisms to *catalogue* and *organise*, *correlate*, *share* and *filter* incoming data are thus becoming a major necessity.

Cataloguing and organising

During our lifetime, there comes a time when we feel that our home is too small for all our stuff, stuff that we consider essential. When we finally move into the bigger house, after a while we found that despite having more absolute space, we end up with roughly the same relative space occupied. But this time with even more stuff. The same phenomenon is happening with our computers. To meet our growing need for storage space, physical devices with increasing capacity emerged.

With more and more data stored, the process of finding what we need becomes time-consuming. Local search engines try to overcome this problem, but still we, as users, find it hard to remember file attributes like the *filename*. Therefore, we need tools that allow us to manually and automatically catalogue our data, which can greatly improve our user experience and productivity. Cataloguing is regarded as metadata [39], more specifically descriptive metadata [35].

Cataloguing can be automatic or manual. On the one hand, the main source of the information stored on our personal computers is the Internet, hence it would make sense to automatically mark all incoming data with their origin. This is an example of automatic cataloguing.

On the other hand, manual cataloguing consists of all the attributes that the user can add to a particular document: tags and ratings. The usefulness of this form of cataloguing can be illustrated by the following example: sometimes when we create or download a file, we would like to add keywords that may not be directly related to the contents of the document. This mechanism would be very useful for non-text documents such as pictures or videos as we could classify a picture as *funny* or add the names of people appearing in that picture. So, if those tags were indexed by the local search engine it would reduce search time.

It is not hard to imagine applications that can take advantage of descriptive metadata in a more complex way, to further improve the organisation of information. The current file system hierarchy is based on the location of files and somewhat restricts our freedom when comes to metadata. We can create a folder to group all the documents that are related to same subject but, very often, one document can be related to more than one subject. As a result, we could create a virtual folder that would dynamically aggregate all documents according to some criteria based on descriptive metadata.

When organising and cataloguing, we can draw patterns and relations between data - correlating - that can be used to provide a better experience to the user.

Correlating

Personal computers have become our personal workbench storing a great volume of personal and professional information. A closer look allows us to conclude that on a daily basis we execute the same kind of operations like: check the news of the day and our favourite websites, send email, download/upload files, etc. In other words, the personal use of computers tends to follow a pattern, which tends to settle down and even become routinary - *user pattern*.

The information stored in our working set of documents tells much about our interests. Providing the user with tools to add descriptive metadata to documents not only allows to better manage information, but also opens the doors to develop new applications to take advantage of semantic information (i.e. metadata).

The working set of documents provides the main source of metadata as a means to infer correlations. However, other sources can be considered namely, the searches both in the local and the web search engine, the most visited websites or the personal social network profile. By correlating the available metadata with the *user* *pattern* we will be able to infer user's interests and ultimately get the *user profile*. This is useful when thinking of providing tailor made utilities.

Sharing

At the present time we are living in a social paradox where we have never been so physically apart from each other but at the same time we have never been so virtually close. Therefore, ermeges the need of staying connected through social networks, instant messaging, document and email exchange. When we rate or append a comment to a document, that document gets a personal dimension, because the inserted metadata represents our personal opinion about it.

Our friends' opinion is highly valued, thus whenever we need to buy a book or a CD we enquire them about the *rating* they gave to that book or CD. So, it would make sense to bring a similar mechanism into our working set of documents. In other words, if we have documents in common with our friends, we can spread the rating or comments we appended to that documents to all of them.

Furthermore, we can think of other mechanism that could prove as useful: whenever we come cross with an interesting document from our working set, we should be able to simply and directly suggest that document, which would end up in our friends' working set of documents.

Filtering

The human being when exposed to hearing stimulus, learns how to ignore the sounds that are not of interest - the so called *white noise*. We are exposed to overwhelming amounts of information and thus we need automatic mechanisms to filter the adequate part of it.

Every day we use filters without even realising we are using them. One of such filters, is the spam filter, which is embedded in most of email providers. Recently, web search engines also provided embedded filtering i.e. search results are filtered taking into account our previous searches.

As a matter of fact, recommendation systems can be regarded as filters, since they provide us information we may be interested in, based on our profile and/or on people with similar interests.

Now, that our working set of documents is so close connected to the web and thus exposed to huge amounts of information, similar approaches have to be brought to our desktop. One possible useful example of a desktop oriented filter, would be to recommend documents we may be fond of based on our inferred user profile and friends' network.

1.2 Thesis contribution

With this work, we undertake to meet the objectives of conceiving and prototyping a set of mechanisms based on metadata, in order to catalogue, organise, correlate, filter and share our working set of documents. These set of mechanisms will form a conceptual and experimental system called SocialSeer.

In order to catalogue and organise data, automatic and manual cataloguing will be used. With regard to manual cataloguing, we will devise a mechanism to allow an easy way of rating data and a simple method for tagging data. With regard to automatic cataloguing, we will make use of already existing methods.

Descriptive metadata extracted from our working set of documents will be combined with the *user pattern*. The point of this correlation is the user profile inference, which takes into account what we were doing during the week but gives special relevance to our actions within the current day.

Another important aspect of this work is the social component. We will conceive two social-based features. The first one is the *document rate sharing*, which aims to spread descriptive metadata among our circle of friends. The second one, the *user-driven document recommendation* tries to mimic the habit of sharing one document by sending an email to the mailing list. This mechanism can be achieved by marking a document to be shared and push a copy to our friends' working sets of documents.

Other goal of this project is to filter information, thus we will design a mechanism to allow an *automatic document recommendation system* based on our personal interests, the source of these suggestions is our friends' working set of documents, which increases the odds of finding compatible data.

The set of devised mechanisms will be exposed in an already existing system. Bolt [32] is that system, which provides a working set dissemination service. Therefore, Bolt will be extended to accommodate the conceived mechanisms in order to give rise to the SocialSeer.

With this dissertation we intend to prove our approach is valid and is relevant to the current IT scenario. Furthermore, we expect that relevant information can be presented and at the same time provide a platform to share our lives and daily experiences with our friends. Ultimately providing the user with useful data and a seamless social experience fully integrated into the working set of documents.

1.3 Dissertation outline

The rest of this dissertation is structured as follows: Chapter 2 presents an overview of work; Chapter 3 presents the concepts devised as well as the proposed architecture of the system and the prototype description; the prototype evaluation and the main results are detailed in Chapter 4. Finally, Chapter 5 concludes this work.

Chapter 2

Related work

In this chapter we overview some concepts that are key to the current research. We address the management of metadata to describe general sorts of documents, existing trends in personnal Recommendation Systems and the concepts involved in Social Networks. While these concepts separatelly are well known and their adoption widespread, only very recently their integration has started being implemented. We review some of these initiatives.

2.1 Metadata

2.1.1 Metadata in the Web

Metadata, literally *data about data* [34], has been growing in importance since the last decade. It has been used in libraries and museums to speed up books finding. But, the Internet and the use of Web search engines to deal with the immense available information online, acted as a true catalyst for the great and growing importance of metadata in information retrieval [34]. Since a long time Web search engines make effective use of crawling and metadata indexing in order to retrieve efficient and accurate results. The indexing mechanism in Web search engines, like Google [30], is achieved by forward indexes and inverted indexes. A forward index [30] maps documents to terms appearing in that document. Conversely, an inverted index [30] maps terms to documents in which that term appears. Both forward and inverted indexes can be built from one another. However, webpage metadata indexing had some disadvantages. Before, Google introduced its *pagerank* algorithm, web search engines were grappling with websites' intentional metadata manipulation. For example, the title HTML tag can be a great source of information as it can provide a summary of the page contents. Nonetheless, if we change the title tag to match the terms of the top search list we could increase our website's visitors.

It was also online that the concept of social sharing of descriptive metadata was successfully introduced. One example is the popular online video sharing YouTube [23], which not only allows users to rate and comment on videos, embedding those attributes in the video format, but also to perform searches on metadata attributes such as tags that the user assigned when she uploaded the video.

2.1.2 Desktop metadata indexing

Due to the growing volume of information stored in our personal computers, similar approaches to the web have been brought to personal computers. For instance, stand-alone applications like PDF Adobe's Acrobat Reader that enabled comments in order to allow reviewing and collaborative work were created. Secondly, open standards appeared, like OpenMeta [11] to allow applications to take full advantage of the metadata attributes. Finally, complete desktop search engines that rely on file-contents and metadata are part of the most common operating systems (Windows, Mac OS X and KDE/Linux). These systems provide a file system-level search, which helps to organize, catalogue, find and access the information. These systems are:

- Spotlight Store [3,18] on Mac OS X;
- Windows Search [21,38] on Windows;
- Nepomuk [9] on KDE/Linux.

All systems provide mechanisms to automatically extract metadata from popular file formats as well as allowing the users to add descriptive metadata to their documents. In addition, Mac OS X Leopard automatically adds the metadata attribute *wherefrom*, with the source website link, every time a file is downloaded from the web. There are not many differences between the three information retrieval systems since they all have followed similar approaches. Nonetheless, we will try to pinpoint their main features and differences:

- All rely on both document's metadata and document's content to provide the user with new ways of organizing and accessing information. All the data is stored in a storage service that holds all metadata attributes about a document and an index to its contents;
- The first time they are enabled the indexing process can take a while to complete, but then all the indexing is incremental;
- When performing a query the results can be filtered according to one or more metadata attributes;
- Nepomuk and Spotlight both are aware of when files are updated, created or deleted, thus the index is updated straightway. On Windows Search as files are updated, created or deleted their file paths are inserted in a queue (the Gatherer) and scheduled to update the index whenever is possible [38];
- Provide APIs for developers to create their own metadata importers as well as already implemented importers for most of the standard file formats (JPEG, RTF, PDF and MP3);

- Windows Search allows natural language query (e.g. mail from John).

In terms of architecture, Spotlight Store [18] is itself a file system-level database that is divided into a Metadata Store (a database specifically created to handle metadata), which has all file metadata attributes, and a Content Index (managed by Apple's Search Kit [14] and only used if the file is text based). The Metadata Store stores each file has an *MDItem object*. This object comprises a dictionary of all metadata attributes associated with that file. The keys of that dictionary are a subset of pre-defined Mac OS X unique keys (e.g. *kMDItemAuthors, kMDItemContentType, kMDItemNumberOfPages*, etc). Those unique keys are generic in order to suite the entire file formats, but Spotlight allows us to create our own personal metadata attributes Another example of these unique keys is the *kMDItemTextContent* (populated if the file has text content), which holds a reference to the Content Index. The Content Index (Search Kit managed) holds an inverted index of the terms that composes the document.

2.2 Personal recommendation systems

Due to the need to filter information, recommendation systems have long been an important research area [25]. In fact, the problem of recommending items to users can be reduced to the problem of estimating the rate one user would give to an unseen object. According to how recommendations are made, recommendation systems can be grouped in three main categories [25] [26]:

- Content-based recommendations: these kind of systems try to estimate the rating one user would give to an item, based on the previous records of rates the user gave to similar items. However, this approach raises some problems. The *new user* *problem*: at the beginning there is not enough information about the user preferences because the user history is too short to produce accurate recommendations. The *limited content analysis*: this approach is based on the content of item i.e. this approach works well with text based items but with not multimedia items because content extraction is very difficult. The last problem is the *over-specialization* meaning that the system will only recommend items similar to the ones the user has already rated.

- Collaborative filtering recommendations: these kind of systems try to estimate the rating one user would give to an item, based on the rating other users with similar profiles, gave to that item. This approach also raises some problems. One of them is also the *new user problem.* Other one, the *new item problem* is similar to the *new user problem* because when a new item is introduced in the system it first has to be rated by a significant number of users. The last problem is *sparsity*: this problem comes from items which are rarely rated (meaning they will be rarely recommended) and from people whose preferences are very different from the majority resulting in inaccurate recommendations.

- Hybrid methods: these kind of systems combine *collaborative filtering* with *content-based* to overcome the problems of both methods. [25] proposes a classification to the different combinations as follows:

  – implementing collaborative and content-based methods separately and combining their predictions;

  – incorporating some content-based characteristics into a collaborative approach;

  – incorporating some collaborative characteristics into a content-based approach;

– constructing a general unifying model that incorporates both content-based and collaborative characteristics.

There are a few examples of popular recommendation systems. These systems are best known for its appliance in the industry of electronic commerce websites like Amazon.com [37]. Amazon has an item-to-item collaborative filtering system (i.e. finds items people tend to buy together), which makes suggestions about books, CDs, etc. There also other examples: MovieLens [27] that is a movie recommending website; bX [28] a recommendation system for scientific papers. One of the downsides of these web based recommendations systems is that they do not rely on personal data, so profiling becomes limited to the actions of the user in the website. More recently, recommendations systems based on the data we have stored in our desktops appeared: AudioScrobbler [1], recommends music using a *collaborative filtering* algorithm based on the metadata of music we listen to. This system has plug-in for iTunes [4], winamp [20] and xmms [22], etc. that extracts the metadata tags from the music we are listening to and sends it to remote server of LastFM; CRESDUP [31] harvests personal data like schedules, favourite websites and emails, in order to build a profile based on the collected data. Then the profile is sent to their server to produce recommendations based on the built profile.

2.3 Social networking

Social Networks are incredibly popular. They expose a fair amount of our personal lives. One of the most valuable information available on Social Networks is the social graph [33]. From it we can learn how people are related to each other (friends, co-workers, family, etc.). The OpenSocial [12] API is already implemented by almost every Social Network and allows for social applications to take advantage of features like the social graph. As we will be seeing later in this work, the social graph plays an important role in order to achieve the goals proposed.

The abstract concept of making suggestions has already been exploited by Social Networks. Facebook [29], the world leader social network in terms of number of users, provides a mechanism of suggestions. Namely, *apps you may like* and *people you may know*. Nonetheless, these suggestions are not centred on the user's interests (as the one presented in this work), but they are made based on what her friends or other users with similar profiles enjoyed the most. On the one hand, *apps you may like* inspects both her social graph and similar profiles, to be aware of which were the applications they gave highest rating and suggest them. On the other hand, *people you may know* explores both social graph transitivity and similar profiles in order to make the suggestions. Beyond Facebook, other social networks have recommendation systems: Youtube has a metadata attribute *Related Videos* embedded in its video format. This attribute may be regarded as suggestions on the same theme, since it is built taking into account videos with identical tags; LastFM [5] is other social network offering a recommendation system called AudioScrobbler. As mentioned earlier this system makes recommendations of music based on the music we listen to and what people with similar tastes enjoyed listening to.

2.4 Undergoing projects

For the time being, there are three projects in their initial stages, which have identical goals to this dissertation: Nepomuk [10], Social Desktop [16] and Social Desktop – Microsoft Research [17]. We overview them next.

2.4.1 Nepomuk

As a matter of fact, the Nepomuk project started in 2006 in the scope of a European funded project. Its goals were to build an open-source framework that defined a set of API's and standards to build what they call the *Social Semantic Desktop.* This Social Semantic Desktop shall aggregate: search for resources, profiling, data analysis, social interaction and desktop sharing. By the end of the European project in 2008, one practical outcome arose: Nepomuk-Java [8]. The Nepomuk-Java is a proof-of-concept of Nepomuk project. It is a stand-alone Java application with a front-end called PSEW. It integrates all the concepts derived from Nepomuk project . Next we will describe the Nepomuk-Java.

Firstly, this project follows a peer-to-peer approach where there is no client-server or centralized coordination. Both client and server run in our desktop. Nepomuk is divided in several modules, there is one module that crawls all the data from the working set of documents: files, contacts, activities, etc. The resultant leveraged metadata is indexed and stored in RDF [13] format. It provides a mechanism to tag documents and automatically suggest suitable tags. Nepomuk also comes with a module to allow search based on content and metadata attributes. This search is both local and distributed. Meaning that, in a search procedure the desktops of users who subscribed the same group are also searched for matching documents. The social features comprise joining a group of interest or share documents within a group. We can create or join a group of people who are interested in the same topic. It is provided as well, backup features: we can backup photos from our flickr [2] account and backup a document, through a web form, to another computer.

As it is stated before, Nepomuk-Java is only a proof-of-concept to prove the validity of the concepts proposed. Therefore, another project appeared Nepomuk-KDE [9]. The main goal of this project is to blend Nepomuk in the desktop. A first version has been integrated in the recently released KDE 4. It is yet a work in progress and only provides a partial implementation of Nepomuk project. Therefore, in KDE 4 we only have access to a metadata information retrieval system, and a mechanism to associate metadata attributes with documents. In KDE 4 the search and crawling are very heavy and consumes too many resources. Although the Nepomuk project and this dissertation share similar goals the approach taken is very different.

2.4.2 Social desktop

There is another undergoing project that is studying ways to connect our desktop to our peers. With this project they intend to ease and to incite sharing and exchanging of knowledge between communities. Having this in mind, they have implemented a *plasmoid* (i.e. a desktop widget in KDE), which will be a first implementation of the concept in KDE 4.3 version. Although their main goal is to interconnect users' desktops, they mainly focus on exchange of information between open-source communities. Moreover, this project wants to incite and support the growth of the open-source software. Considering this, the widget will provide a news channel for open source information; provide Linux live support; support groups and friend relationships; see what are the activities that our friends are in and an events feed.

2.4.3 Social desktop – Microsoft Research

Recently, Microsoft Research made public its intentions of researching on the Social Desktop subject. A visit to their website helps us to understand what are the project intentions in the future. First, they make it very clear the project will be a proof-of-concept and it is in the initial stages. However, they give us some hints on the architecture and the approach they will follow.

Social desktop – Microsoft Research, wants to free data from the limits imposed by the file-system concepts. In order to do so, they intend to attribute a unique web URL to each data on our working set. These URLs identifiers not only provide access to the file, but also add a preview of the file as well as comments, tags and related items. They call this set a social preview, which will be accessible through a Silverlight [6] web page. With that social link our friends will be able to directly download data and add comments or tags that will be directly available in our Windows desktop. In what comes to architecture, the social previews and documents will be stored in their cloud service Windows Azure [7]. Social Desktop application will be a local service running on our desktops. The local data will be mapped into a .NET bus service, which will enable access through firewalls. They also intend to provide a web service so RSS event streams can be possible. As mentioned previously, this project is taking its first steps and intends to research on how to connect the desktop to the web.

Chapter 3

SocialSeer

With the increasing amount of information stored on our computers, comes the need to organize, catalogue, correlate, filter and share information. Towards this necessity, we devised a platform integrated with the user's desktop that eases the organization and correlation of documents and, at the same time, promotes the exchange of information through the social recommendation.

SocialSeer offers its functionality based on metadata found on the user's working set of documents. Locally, at the user's machine, the system collects metadata from the working set of documents. Some metadata attributes are automatically assigned to each document while others are explicitly assigned by the user, e.g. rates and arbitrary tags.

All metadata collected locally is replicated remotely into SocialSeer's server. The server indexes all the available metadata to infer user profiles and to feed an *automatic document recommendation* service. To the networked users the output of SocialSeer's server is twofold: an enriched characterization and valorization of their documents and the recommendation of new documents of interest from the users community.

The current prototype of SocialSeer builds on the operating system metadata management for the local features and on our own file synchronization system *Bolt*. *Bolt* is a *working set dissemination service* fully integrated into the desktop. It aims to synchronize a user's working set of documents across several machines. In Section 3.2 we describe its basic features and the extensions made to support SocialSeer.

3.1 Architecture

SocialSeer uses a client/server architecture. The client module runs on the user machine, seamlessly integrated with operating system's user interface and offering the following functional modules:

Metadata Manager module is responsible for two main actions:

- collecting metadata information whenever a document is created or modified.
- managing auxiliary metadata files, one per document, containing the documents' metadata attributes and unique ids.
- updating auxiliary metadata files with incoming server data.

Get Friends module looks for friend's contacts in local applications, such as address books and social networking clients.

Tag and Rate module is embedded in the operative system and allows the user to assign a arbitrary tags and rates to any document in the file system.

On the server side, six main functional modules are available:

Index Manager module aggregates a set of indexes (both forward and inverted indexes). There is a set of indexes for all metadata attributes that map a key to a value.

Metadata Parser module parses all the incoming auxiliary metadata files so the document's metadata attributes can be indexed.

Profiling module elaborates the profile of every user of the system.

Document Rate Sharing module provides the document rate sharing feature.

Open Social module makes use of the Open Social API to interact directly with the web social networks, in order to retrieve and send relevant information.

Document Recommendations module provides documents recommendations. In conjunction with the profiling module, supports *automatic document recommendations* and the *user-driven recommendations feature.*

3.2 Prototype

In the following we describe the implementation of SocialSeer's prototype. The whole system relies heavily on the Bolt [32] working set dissemination service. Both the Bolt client and server software has been extended to support SocialSeer. The current prototype client software has been developed to integrate with Mac OS X.

We start by an overview of Bolt and how it has been extended in this work. Then we define the set of considered metadata and its management within SocialSeer. The establishement of social relationships, user profiling and their combination is detailed next.

A discussion of some implementation aspects related to the system's overall performance conclude the section.

3.2.1 Bolt and SocialSeer

The Bolt system is a working set dissemination service completely integrated into the desktop. It monitors a given folder for content updates and synchronises them across all of the user's machines.

Bolt provides a basic platform with a set of mechanisms easing the implementation of SocialSeer concepts. One of such mechanisms, is the *file system watcher* that provides the monitoring of file system. Another mechanism, is the transparent synchronization of updates with a remote server.

However, as an ordinary working set dissemination service, Bolt isolates each user's set of documents from the remaining users, to preserve their privacy, as depicted in Figure 3.1.

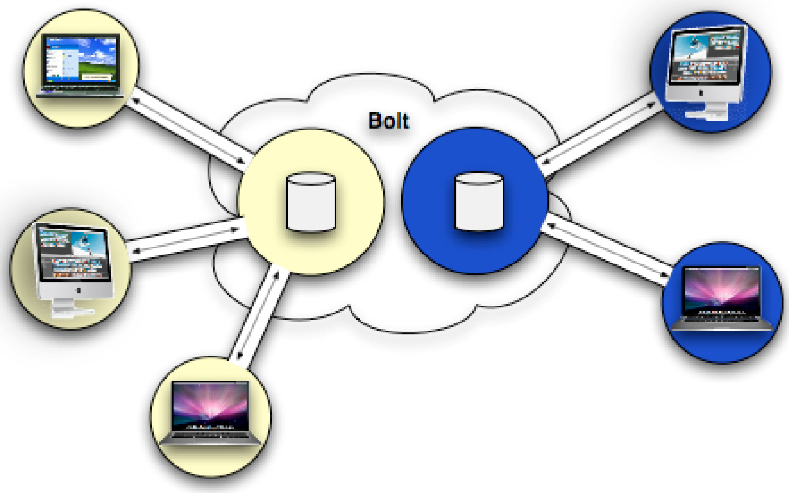


Figure 3.1: Bolt's isolation level

With SocialSeer we try to expand Bolt by reducing its level of isolation, in order to enable the sharing of information within communities of users. Nevertheless, we still take into account users' privacy: only the metadata and documents, the users determine to be public or that are publicly available in the Internet are shared. This balance we reached between private and public, enables the concepts presented in this work. Figure 3.1 and 3.2 depicts Bolt and SocialSeer different approaches.

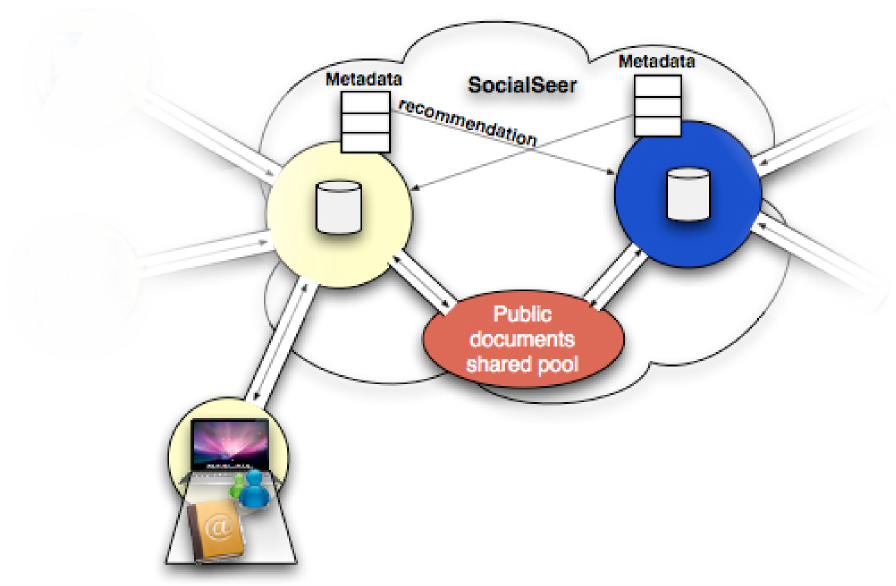


Figure 3.2: SocialSeer's isolation level

3.2.2 Metadata: gathering and manipulation

Metadata plays a central role in this dissertation. From metadata, we can improve our understanding of the documents' contents, resulting in better organization and search accuracy, inferring user preferences, recommend documents, etc.

Bolt system does not provide any means to deal with the working set of document's metadata. As a result, in order to support the presented concepts we have added some new modules to Bolt's client application. In fact, as showed in Figure 3.3 there were introduced four new modules: *Get Metadata*, *Set Metadata*, *Get Friends* and *Star Rating Finder plugin*. The purpose and the behaviour of these modules will be held next.

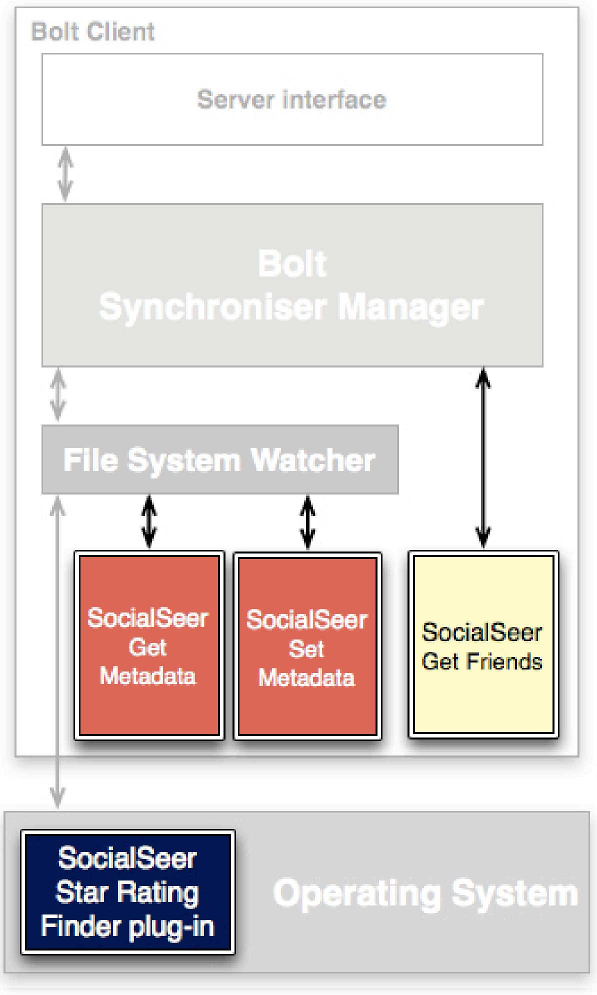


Figure 3.3: Introduced modules in Bolt's client application

Metadata subset

The Mac OS X operating system provides us a set of pre-defined metadata attributes that are common to a lot of file formats. In fact, this set is quite big, offering over one hundred of metadata attributes. We chose a subset of these attributes that we believe are needed to support the goals proposed for this work. This subset comprise attributes, which are common to the majority of file formats:

- *kMDItemKind* depicts the type of a document according to the file type. For instance, if a *PDF* document this attribute would contain: *PDF (Portable Document Format)*;

- *kMDItemAuthors* contains the set of one or more elements of the authors of the document. For example, if a music file it may hold the name of the band;
- *kMDItemTitle* depicts the title of the document. For instance, if it is a music file it may hold the song title.

and some descriptive attributes:

- *kMDItemStarRating* contains the score, zero to five, assigned to a document;
- *kMDItemFinderComment* used to hold an arbitrary set of tags assigned to the document.
- *kMDItemWhereFroms* if the document was downloaded from the web, the download link is assigned to this attribute;
- *kMDItemFriendStarRating* attribute is not present in Mac OS X and was created in the scope of this dissertation. This attribute may contain the score that our friends have given to the document;

Finally, this subset also consists of metadata attributes that are specific to file formats:

- *kMDItemKeywords* holds the keywords defined in a *PDF* document;
- *kMDItemDescription* holds a brief summary of *PDF* document;
- *kMDItemAlbum* holds the name of the album of a music song file;
- *kMDItemComposer* holds the name of the composer of a music song file.

Metadata editing

Most of metadata is extracted from the documents. However, there are some attributes that can be directly edited by the user. In this subset we are interested in those attributes assigned by the user like tags and ratings. There are not mechanisms embedded in the operating systems to easily edit these metadata attributes. For instance, in order to edit the rating of a document in Mac OS X we would have to use a stand-alone application like Tagit [19]. Moreover, there is no mechanism to visualize the rating assigned to a document. Therefore, we implemented a *Finder plugin* 3.3 fully integrated within the operating system. This plugin allows the user to select one document and with a context menu, to rate the document from none to five stars. This action will set the *kMDItemStarRating* attribute to the chosen *star rating.* Furthermore, the plugin also provides a new way to simply visualize the rating of a document: when we assign a rating to the document its icon changes; the plug-in overlaps a star icon, corresponding to the star rating value, with the icon of the document.

With regard to document tagging, even though there is no direct method to simply add tags to documents, we decided to use the *kMDItemFinderComment* as the tag holder. This attribute can be edited by opening the file information dialog. It is a free form text input, so in order to be used as a tag editor, the user is advised to follow the syntax: “keyword, keyword…”. The inserted text is then parsed and transformed into tags.

Both the *kMDItemStarRating* and the *kMDItemFinderComment* are indexed by the local search engine. Therefore, performing look-ups on these attributes is possible. In addition, the Mac OS X *smart folders* mechanism allows a new way of organizing documents. They are dynamically populated according to some criteria. With the mechanisms purposed we can create a *smart folder* that matches, for instance, the following criteria: all image documents, which are tagged as *funny* and with a rating equal to five stars, thus aggregating documents not by their location in the file system but by a common attribute.

Metadata collecting

Mac OS X has a set of metadata importers to the most common file types. These importers harvest our working set of documents. The extracted metadata is then indexed by *Spotlight* and made available for searching. *Spotlight* provides an API to retrieve metadata associated with a document. When we move or copy a document within the monitored folder, the *Get Metadata* 3.3 module prompts Spotlight for the defined subset of metadata attributes, associated with that document. If the document does not possess all the attributes only the corresponding smaller part of the subset is returned. *Get Metadata* also computes a hash key (*SHA-1*) of the document in hexadecimal form.

Both the *SHA-1* key and the retrieved metadata attributes are written into an invisible auxiliary file with name: *.metadata-filename*. The auxiliary metadata file is put in the same folder as the *main* document and is synchronized to the server like all the others. In addition, whenever a document is created, deleted or modified the corresponding *.metadata* file is updated accordingly. The main document and auxiliary file are treated as a single entity, thus when we share or synchronize a document the auxiliary file is sent along with the real document. This auxiliary file is key to the system proposed.

Descriptive metadata synchronization

Descriptive metadata attributes cannot be obtained from the document contents. Therefore, when a document is synchronized or shared between two machines those attributes are lost in the process. The *kMDItemStarRating*, *kMDItemFriendStarRating*, *kMDItemWhere-*

Froms and *kMDItemFinderComment* attributes are stored as *extended attributes* [3] in the file system. The same happens to the custom icon from a rated document, which is stored in the file's resource fork.

In order to maintain those attributes when we share or synchronize documents we take advantage of the auxiliary metadata file . As mentioned before, the auxiliary file is always sent along with the main document and holds all the metadata belonging to it. As a result, all incoming auxiliary metadata files are parsed by the *Set Metadata* module 3.3 and the descriptive metadata attributes assigned to the respective main documents.

3.2.3 Personal metadata indexing

To enable a fast access to a document's metadata from all the features presented in this work, we needed to index the metadata attributes. First of all, each user has its own set of indexes, thus isolated from the other users.

Every synchronized document has its auxiliary metadata parsed by the *Metadata Parser* module in the SocialSeer's server. By parsing we mean, reading the auxiliary file from the beginning to the end in order to retrieve the metadata attributes. Then, according to the kind of the attribute, the corresponding index is updated. Some attributes have their own index while others are indexed combined. Figure 3.4 illustrates the indexing process. We use two kinds of indexes: *forward* and *inverted* indexes.

Forward indexes

A *Forward index* maps documents (key) to terms (value) appearing in that document. In our case, key refers to the *filename* or the *SHA-1* key and value refers to the metadata attribute. This kind of indexes are very useful when we need to find out the metadata

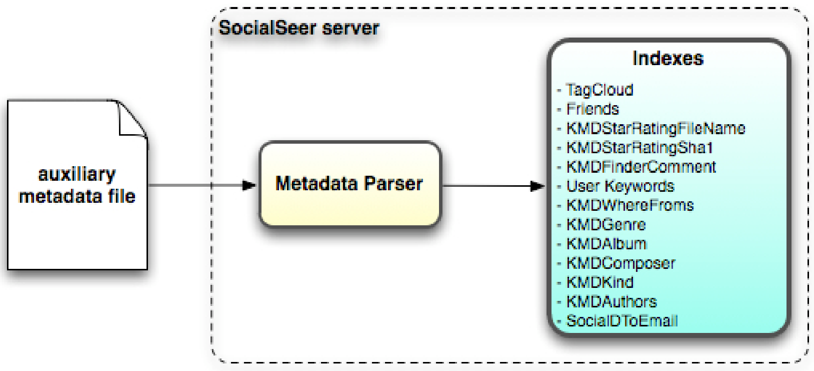


Figure 3.4: Indexing process in SocialSeer's server

attributes related to a specific document. Because descriptive metadata can not be extracted from the contents of the document, these attributes are indexed in forward indexes:

- *KMDWhereFroms* index the metadata attribute *kMDItemWhereFroms* in relation to the *filename* key.
- *KMDStarRatingFilename* and *KMDStarRatingSha1* index the star rating of a document in relation to the *filename* and *SHA-1* key.
- *KMDFinderComment* index the metadata attribute *kMDItemFinderComment*, which key is the *filename*.

Inverted indexes

Conversely, an *Inverted index* maps terms (key) to documents (value), in which those terms appear. In this work, key refers to the metadata attribute and value refers to the set of documents containing the attribute.

When thinking of providing search based on terms, this sort of indexes are a solution. As a result, we defined a few inverted indexes, which we will be presented next. For a better understanding we have grouped indexes with similar characteristics:

- ***KMDGenre, KMDAlbum, KMDComposer and KMDAuthors***

  As the names suggest, these indexes are associated with the *kMDItemGenre, kMDItemAlbum, kMDItemComposer* and *kMDItemAuthors* metadata attributes, respectively. Each maps the corresponding attribute to the set of documents containing the attribute.

- ***KMDKind***

  This index is related to the *kMDItemKind* metadata attribute. However, it differs a little from the previous indexes in that, the mapping between the attribute and the set of documents is not direct. Due to the multitude of document types and the *document rate sharing problem*, we decided to limit the spectrum to four main document types: audio, video, image and other. As a result, all the *kMDItemKind* attributes are mapped into one of these four categories.

  The main document types form the set of possible keys in the index, but the values are yet another index i.e a *sub-index*. The *sub-index* maps the *filename without extension* (key) to a set of documents with the same names but with different extensions (value). Figure 3.5 depicts this index.

- ***User Keywords***

  Every text-based metadata attributes i.e. all except for the *kMDItemStarRating*, are also mapped by this index. The *User Keywords* index groups all attributes into a single index, thus

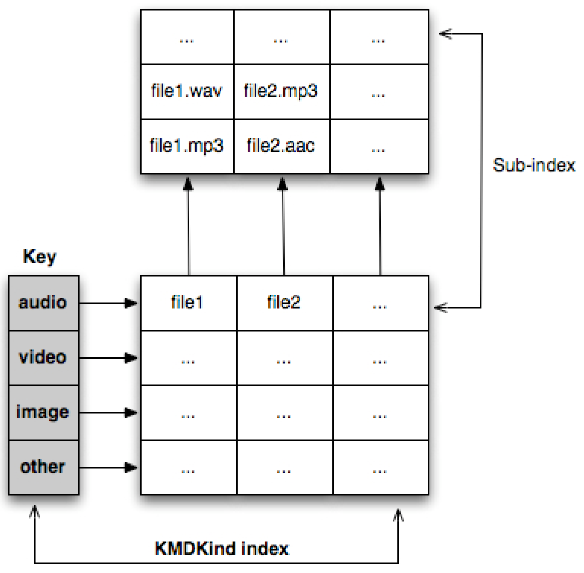


Figure 3.5: KMDKind index

aims to provide search based on terms independently of the metadata attribute.

In this index it is introduced an intermediate building stage, which is assured by the *Stemmer* [15] module. Briefly, for every attribute being parsed, before being handed to the index, is passed through the *Stemmer*, which will reduce the term to its morphological root. Then, the *stemmed* attribute is inserted into the index (key), along with the *filename* (value), in which it appears.

Auxiliary structures

In addition to forward and inverted indexes, there are other useful auxiliary structures:

- *Tag Cloud* is an index (can be also thought as a weight list) that maps terms into the total number of occurences they appear in our data. It takes into account all the text-based metadata attributes and it shares the same building stages as the

User Keywords index;

- *SocialIDToEmail* index maps the users' *socialID*[1] to their email.
- *Friends* holds all friends contacts related to a user.

3.2.4 Dynamic user profile

There are two types of user profiles: the long-term profile and the short-term one. The long-term profile covers the set of subjects we are more interested in. For instance, everyday we usually listen to the songs we like the most, we surf some of the same web sites, because these are our personal preferences. However, there is a set of subjects we are interested, which may vary from day to day. This is the short-term profile and we call it *daily user profile*. For example, when we need to buy a new phone, when we listen to a new band or when we heard about a subject in the news and we want to know more about it. In this section, we refer to a *global user profile* and a *daily user profile*. Both will be described in the next paragraphs but for better prior understanding the *global user profile* is a merge of daily user profiles.

Generally, user profiling is very important in the context of *automatic document recommendations*. Given the heterogeneity of the subjects we are concerned over the days, we wanted the *automatic document recommendations* to be made on a daily basis. Everyday we compute a *daily user profile*, which follows a *Tag Cloud* approach. In other words, it is a weight list based on the metadata term-frequency from the documents we created or modified during the day. In terms of implementation, it is built using the same parser code as the *User Keywords* index. And for each parsed keyword or term: if the weight list already contains that keyword the respective counter is incremented; instead if the keyword does not exist in the weight list,

[1]The *socialID* is an unique personal identification number attributed by the social networks to every user.

then the keyword is appended to the list and its counter set to one occurrence. The *Tag cloud* index holds the *daily user profiles*.

However, for a more accurate profiling, which could also cover the long-term profiling, we decided to introduce a new factor. We call it the *aging factor*. Basically, for each user we store their five newest *daily user profiles*, from them we compute the *global user profile*. In order to build it, we assign a weight to every daily user profile. The newest *daily user profile*, corresponding to the current day, is assigned the highest weight. Accordingly, to oldest one is assigned the lowest weight. The distribution of weights is the following: five to the newest *daily user profile*; four to the second newest; three to the third newest; two to the fourth newest and one to the oldest. Next, each frequency of the keyword is multiplied by the assigned weight. In the next step, we merge all the five daily user profiles into one, the *global user profile*. Finally, the *global user profile* is sorted by descending frequency. This behaviour is depicted in Figure 3.6.

To sum up, the *global user profile* holds some interesting features. Although it gives more importance to the contents we saw in the current day, if we were interested in the same subject in the previous days that subject would still be in the top of the list.

3.2.5 Circle of friends

Although not a regular social network, the system heavily depends on social relationships and interactions. The system, itself, does not provide the means to establish and maintain these relationships. Instead, the social relationships are obtained in an indirect way. There are two main sources from where we extract the social information: the local computer and the web social network account.

Our local computer provides one source of social information extraction. When we first register with the system, the running client - *Get Friends* module 3.3 - searches our personal computer for contacts. It

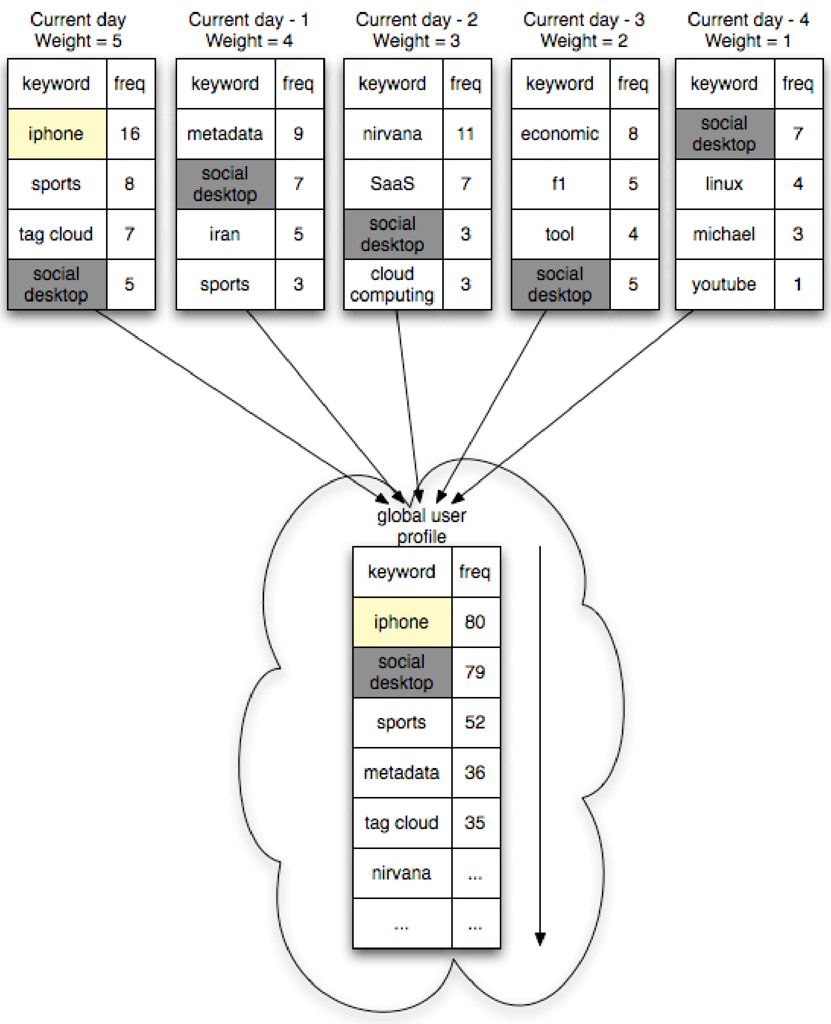


Figure 3.6: Dynamic user profile

looks in the local *Address Book* application for our contacts' emails. Moreover, it also retrieves contacts from instant messaging account such as Jabber, MSN, Yahoo, AIM and ICQ. Then, all the contacts are sent to the server to be indexed as people we are related to.

Besides this local information, we on the system also try to get the social graph from our social network account. When we first register with the system we are provided with a name of the *OpenSocial gadget* we need to install from our social network web site. These gadgets are XML applications, which grant access to our profile. We had to implement one of these *gadgets* that has to be certified by the social network in order to be publicly available in the applications area. Our *gadget* grants access to our profile and displays our *socialID*. With both the *gadget* and the *socialID Open Social* module retrieves our social graph, which does not hold our friends' emails but their *socialIDs*. Then, the retrieved *socialIDs* are handed to be indexed. The social graph retrieving process is depicted in Figure 3.7.

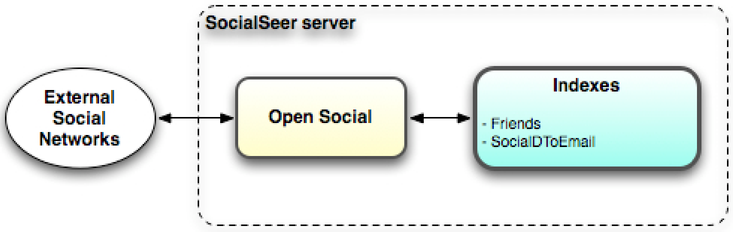


Figure 3.7: User's social graph retrieving process in SocialSeer's server

3.2.6 Sharing of document rates

This mechanism allows the sharing of the rating that we give to the data we have in common with our friends. Next, we describe in detail the algorithm we devised to allow this sort of behaviour.

As explained before, whenever a document is synchronised its auxiliary metadata file is sent along with it. Then, the *Metadata Parser* module parses the auxiliary file in order to update the user's indexes. On completion of the parsing process, the module verifies if the synchronised document has been rated by the user, if not it signals the *Document Rate Sharing* module to initiate the sharing process.

As a matter of fact, we wanted to distinguish between two major documents types that we call multimedia documents (i.e. video, audio and image) and the non-multimedia documents. Therefore, we devised two versions of the algorithm, which are applied according to the above categories.

For the non-multimedia documents, for every user's friend we get their *KMDStarRatingSha1* index and we search for an identical *SHA-1* key. If there is a match, the respective rating is brought to the *Document Rate Sharing* module.

However, for the multimedia documents we start by using the same approach, but if there is not a *SHA-1* key match, we get the user's friends *KMDKind* index, in order to check whether there is a document of the same type and with the same name, as the synchronised document. If there is a match the respective rating is brought to *Document Rate Sharing* module. An example of this behaviour is depicted in Figure 3.8.

The reason for this separation relates to the fact that the multimedia documents represent a special case at the file level. In other words, two users can, for example, have a video with the same *filename* but with different extensions i.e. encodings. However both documents represent the same video. As for text documents, it becomes less likely that two text documents with different extensions represent the same document.

From now on, both versions of the algorithm follow the same stages: after all the ratings have been collected from all the user's friends, it is made an arithmetic mean and the resulting rating is sent to the user.

On the client's side, the *Set Metadata* module is responsible for creating the *kMDItemFriendStarRating* metadata attribute with the received rating. Furthermore, to warn the user of the friend's rating we use the *Star Rating Finder plug-in* mechanism but with a slight difference: the overlap icon displays different colours to distinguish from the user assigned rating.

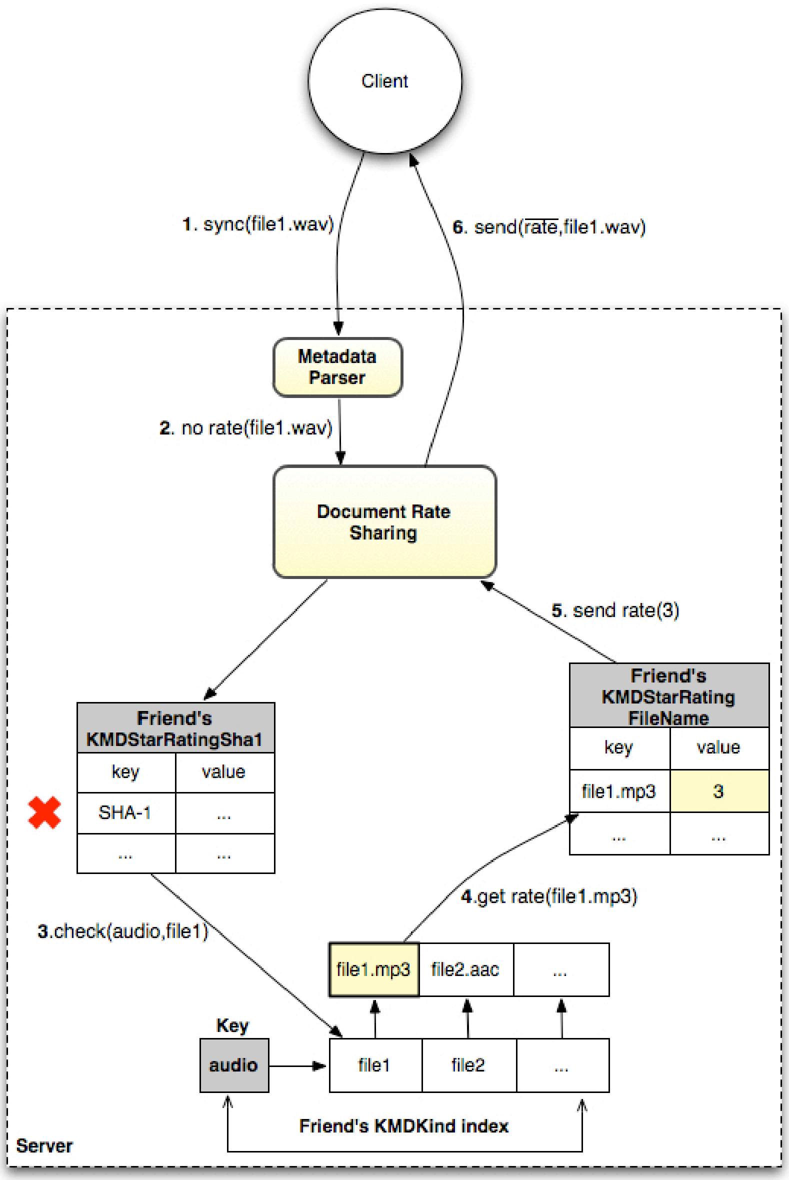


Figure 3.8: Example of a multimedia document rate sharing

3.2.7 Document recommendations

In this dissertation we propose two *document recommendations* mechanisms: *automatic document recommendations* and *user-driven recommendations*. In terms of implementation, both share the same principle: a folder called *inbox* is created inside the monitored folder and recommended documents are pushed by the system into that folder. Both mechanisms are described in detail in the following subsections.

Automatic document recommendations algorithm

The *automatic document recommendations* are made on a daily basis and are based on the global user profile. With this mechanism we had to take into account the user's privacy. So a fair balance between total privacy and no privacy had to be reached. As a result, only the documents we determine to be public will be pushed directly, as recommended documents, to our friends. However, our private documents, or better saying, non-public documents[2] are still considered when recommending documents. There is a slight difference though: instead of pushing a copy of the recommended document into the inbox folder, we send a link, extracted from the *wherefroms* metadata attribute of the document, where it can be downloaded.

Bearing in mind the classification of recommendation algorithms presented in Chapter 2, the proposed *automatic document recommendation* algorithm can be categorized as a hybrid algorithm, because it combines a *content-based* approach with a *collaborative filtering* approach. In other words, documents from users with similar preferences, the circle of friends, are chosen(*collaborative filtering*) based on what the user liked in the short-term past (*content-based*). Next we will present a detailed view of the algorithm.

The first step of the algorithm is to obtain the *global user profile*, which is a list of keywords sorted by descending frequency. The next step takes into account the relative frequency of each keyword to determine how many document recommendations of that keyword are to be made. More formally, this step follows the function:

[2]Documents that were not manually determined to be public, but by its metadata attributes there are web links to download it.

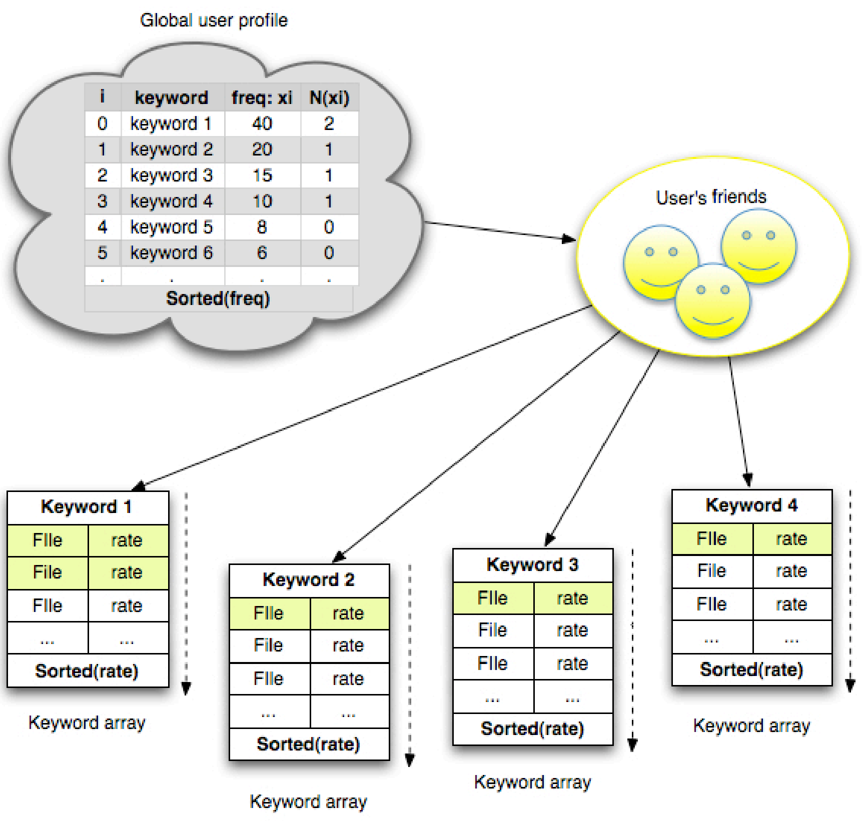
$$N(x_i) = \lfloor (\frac{x_i}{\sum_{i=0}^{S_{total}-1} x_i}) \times S_{total} \rceil \tag{3.1}$$

Where $N(x_i)$ is the number of recommendations for a given x_i. x_i is in turn the frequency of a keyword in the position i from the *global user profile* set. S_{total} is the total number of document recommendations to make. Moreover, S_{total} is a static parameter defined in the system, however, in the future, we intend to give the user the possibility to determine the total number of recommended documents she wants to receive.

The first factor, gives the relative frequency i.e. the approximated probability, of a keyword appearing in the first S_{total} positions of the set. There are two reasons why we only take the sum of the S_{total} first positions: the first one is to cover the worst-case scenario where the frequency of all keywords is equal. For instance, take an ordered set of thirty elements with a frequency of one to all keywords and S_{total} of ten. The sum of all absolute frequencies would be thirty. Consequently, the resultant number of recommendations to each keyword would be near to zero; the second reason comes from our observations that led us to conclude the distribution of frequencies usually follows a common pattern. This pattern has a lot of occurrences of some keywords but low occurrences of a lot of keywords resulting in similar behaviour to the worst-case scenario. The adaptation of the formula leads to better results and mitigates the worst-case problem. Taking the example above, if we take only the first ten positions of the set, the sum of all absolute frequencies would be ten. Therefore, in the worst-case scenario we would issue one recommendation for each of the first ten keywords. In fact, since $N(x_i)$ is rounded to the nearest integer number, the maximum possible number of recommendations is $S_{total} + 1$.

In the third step, we create as many arrays as the number of different keywords that were calculated to be recommended - the *keyword* *arrays*. Then, for every one of our friends we search for each keyword in their *User Keywords* index. If there is a match, the document is added to the respective *keyword array*. Next, the resultant *keyword arrays* are ordered by the document's rating.

The final step, takes all ordered keyword arrays and for each it is picked the $N(x_i)$ first positions of the array (i.e. the $N(x_i)$ most rated documents) and handed to the user as a document recommendation. However, if our friends have no documents matching some keyword then no recommendation is made for that keyword. An example of the algorithm behaviour can be seen in Figure 3.9.



i	keyword	freq: xi	N(xi)
0	keyword 1	40	2
1	keyword 2	20	1
2	keyword 3	15	1
3	keyword 4	10	1
4	keyword 5	8	0
5	keyword 6	6	0
.	.	.	.
Sorted(freq)			

Figure 3.9: Automatic recommendations algorithm example, where $S_{total} = 5$

User-driven recommendations

Every day we send emails to our friends in order to collectively share some data. This data can be text documents, images or videos. Having in mind this behaviour, we decided to mimic it by allowing *user-driven recommendations*. *User-driven recommendations* is a mechanism that allow us to explicitly share data with our peers. Next, we will describe the flow of such operation. First, we mark a document inside the monitored folder as a *to share* document. Then, the running client application becomes aware of that flag and in the next synchronization period the server is told to share that document. From now on, the procedure is similar to the rate sharing algorithm: for each of our friends the document to be shared is put in the *updates* list; in the next synchronization of our friends the document is brought to a folder inside their inbox folder; this folder is called *username recommendations*, where username can either be our email or our name (if the email is available in the *Address Book*).

One of the problems we had to think about when devising this feature was how to signal the *user-driven recommendation*. We came up with a simple system of a dialog box over the icon of the application alerting us to a new recommendation. The alert message would follow the syntax: *username* has recommended you *name of the document*. Nonetheless, the implementation of this signalling feature will be left for future work.

3.2.8 In-memory index holder

The modules described along this chapter, imply an increased amount of I/O operations. For instance, every time a document is synchronized its auxiliary metadata file must be parsed and the indexes must be updated, which leads to two I/O operations per index: one read operation that brings the object from the storage to the server; then the index is updated accordingly; finally it is wrote back to the storage. As a result, in order to reduce the number of I/O operations, which could lead to poor performance, we introduced a new module: *In-memory Index Holder*. In a synchronisation period, we can have a lot of documents to be synchronised and thus the respective indexes need to be updated. Therefore, during a synchronisation period, the needed to be updated indexes are brought from the storage into memory and left in memory i.e in the *In-memory Index Holder*. After the synchronisation is over, all the updated indexes are flushed to the storage.

With this module we can save up to one read I/O operation i.e. the stage of bringing an index from the storage into the server. Taking the example above, for each index update, the index is read from the *In-memory Index Holder* (if it is there), then it is updated accordingly and by the end of the synchronisation period, is wrote back to the storage. In Figure 3.10 it is depicted the role of the *In-memory Index Holder*.

3.2.9 Photo album synchronization

This was an idea that emerged at a later stage of this work. And has emerged has a proof-of-concept to demonstrate the possibilities of integrating the desktop with the web. As a result, taking advantage of the recently released version 0.9 of the OpenSocial API that allows to fetch and upload media items, we decided to create a *photo album* synchronization mechanism. A folder called *photo album* is created inside the monitored folder. Then, all images (and only images documents) are synchronized to our photo album in the social network. In addition, the contents of the folder are synchronized as all the others. Besides, along with the images is also sent the descriptive metadata associated with each document (i.e. rating and tags). As a result, in the photo album our friends can see the photos as well as the tags or ratings we gave to them. This mechanism provides full synchronization, which means that if we modify (delete, create or

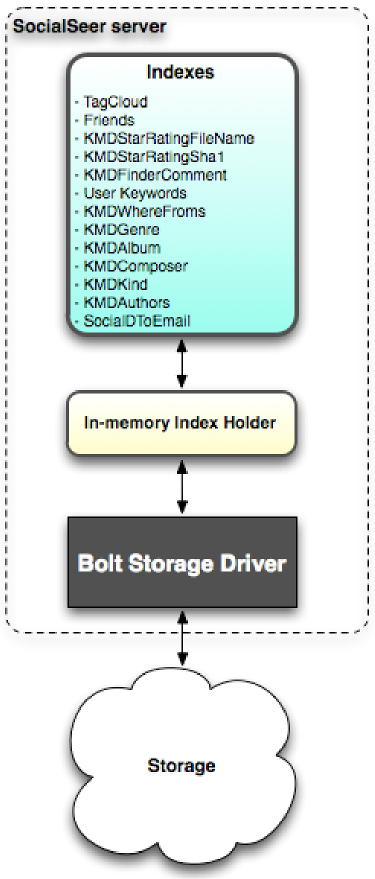


Figure 3.10: *In-memory Index Holder* and how it relates to other modules

change) a picture locally, the picture is also modified in the remote *photo album*. Moreover, the mechanism works in both ways i.e. if we had a picture via web browser the picture will also end up in the local *photo album* along with the tags and ratings.

3.2.10 Implementation considerations

In order to show the feasibility and more easily evaluate the concepts proposed, we implemented a prototype of the system. With regard to the programming languages decision, we chose Objective-C for the *Mac OS X's* client and the server was written in Java.

The reason we chose the Mac OS X environment for the prototype's client, relates to several factors. Firstly, it provides a search engine based on metadata (Spotlight) with a simple API to easily access document's metadata. Secondly, it defines a comprehensive set of metadata attributes to which we can add new attributes. In addition, Mac OS X also provides an easy integration with built-in applications such as *Address Book*. Finally, *Bolt* client is fully integrated into this operating system providing monitoring of file system events. Consequently, Objective-C was the logical choice for the client's programming language, as it is the operating system's native language.

With regard to the programming language of the server, we chose Java because, apart from being the language used by *Bolt* server, it provides code portability as well as the necessary data structures.

Chapter 4

Experimental evaluation

Evaluation is always an important stage when developing new systems to prove the validity of the concepts devised. However, this work has components that can only be evaluated by qualitative tests, as a consequence it becomes difficult to evaluate. One example of such a component, is the recommendation system. As [36] states it is very hard to evaluate a recommendation system, because there is no agreement on what metrics to use in order to assess the *quality* of the recommendations. Nevertheless, during the development process we have performed some empirical tests to verify that the algorithm could produce relevant and accurate recommendations.

Therefore, we decided that the set of tests to be carried should be more quantifiable, thus the tests we conducted focus on the impact that the components added have in the performance of *Bolt* system. The experimental scenario description will be held in Section 5.1.

In Section 5.2 and 5.3, we present the results obtained and in addition we draw some conclusions about them.

4.1 Experimental scenario description

In order to verify the impact that the components added have in *Bolt* system performance, we conducted two different tests that establish a direct comparison between our system's behaviour and *Bolt's* system behaviour:

Load test

This test is mainly intended to verify the differences of performance between the two servers, with the gradual increase of the load. To this end, we used metrics as the percentage of CPU and the memory used by the servers' Java VM.

First, it should be noted that we used for each server i.e. our system's server and *Bolt's* server, exactly the same test conditions. Moreover, in order to better understand the behaviour of the systems with the increase of the load level, we decided to gradually add clients, thus every five minutes we introduced a new client into the system. Clients were configured to initiate a synchronisation period from ten to ten seconds, resulting in six connections per minute and per client, to the server.

We have implemented a Python [24] script that would run every five seconds, in order to create a text file or a folder with a probability of 90% and 10% respectively. These were then moved into the monitored folder. Both the contents and the size of the file were generated randomly and the maximum file size was 1MB.

Regarding the test environment, for the client's application we used a total of nine *iMac* machines, with 2 GB of RAM and a 2.0 GHz Intel Core 2 Duo CPU, which were connected to the server via LAN. In turn, the server was running on our personal laptop that has 4 GB of RAM and a 2.2 GHz Intel Core 2 Duo CPU. For the *storage* we used our local hard disk of 320GB.

Latency test

We have also decided to measure and compare the latency in both systems. By latency, we mean the time it takes to send a file from one user's machine until it gets to another machine belonging to the same user.

The existence of an auxiliary file metadata 3.2.2 per main document implies, that we increased to double, the number of files to upload[1]. Therefore, we have measured the latency in two different scenarios: for a single document being synchronised and for the case of multiple documents.

For the single document scenario, we used a *Adobe's photoshop* document of 75 MB of size. With regard to the multiple documents scenario, we have chosen a set of 287 documents totalling 75 MB. The set was comprised of *PDF*, text, music, video and image documents.

We wanted this test to be conducted in close to real conditions, as a consequence we have deployed both servers at one of our universities' servers and simulated the use of an ordinary external ADSL connection. In addition, we used two different computers, one for each client's machine, which lead us to a clock synchronisation problem. Therefore, we used the NTP[2] protocol to synchronise both machines' clocks and accurately measure the latency. There is yet other decision we made concerning the load on the servers: we opted to use just one client so we were measuring the overhead the added components have.

In what comes to the test environment, for the client's application we have used our personal laptops: the first one has 4 GB of RAM and a 2.2 GHz Intel Core 2 Duo CPU; the other has 4 GB of RAM and a 2.5 GHz Intel Core 2 Duo CPU. The servers were running

[1]However, the auxiliary file is a very small file, in average occupies 50 bytes.
[2]http://www.ntp.org/

on a two core 2.4 GHz AMD CPU with 4 GB of RAM. Regarding the ADSL connection, it had 997 kbps upload bandwidth and a download bandwidth of 12000 kbps.

4.2 Load test results

With this test, we wanted to verify what was the overhead our system introduces relatively to *Bolt* system. Having this in mind, we used the percentage of CPU usage and the memory used as metrics to assess both systems, with gradually increasing load.

We started off with just one client, which is equivalent to six connections per minute and we went as far as the ninety six connections per minute. We stopped introducing new clients at this level. because it was when the LAN's network reached its maximum throughput, thus both servers could not attend any more requests. In addition, during each test the volume of data synchronised reached the 40 GB.

Regarding the CPU usage, Figure 4.1 shows that both systems' behave similarly. Moreover, as the number of connections per minute increased, the percentage of CPU used also raised, and almost in a linear way. Comparing both systems, we can observe that our system consumes a little bit more of CPU, in average, one percent more than *Bolt* system.

Figure 4.2 depicts the total of memory used as the number of connections increases. Once more, the behaviour of both systems is quite similar. In fact, until the number of connections reaches the sixty six connections, the performance of the two systems is identical. However, from this point our system begins to consume more memory, but, in average, during our experiment our system consumed just 1 MB more than *Bolt* system.

To sum up, from the results of this experiment, we can infer that, both our system and *Bolt* system demand the same amount of re-

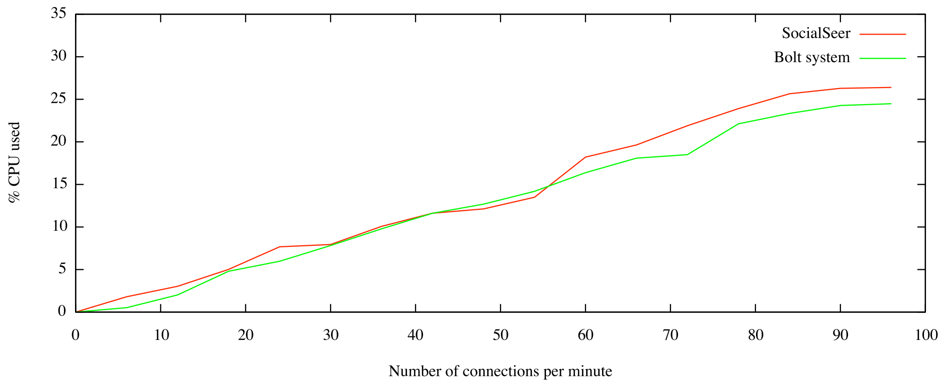


Figure 4.1: CPU usage

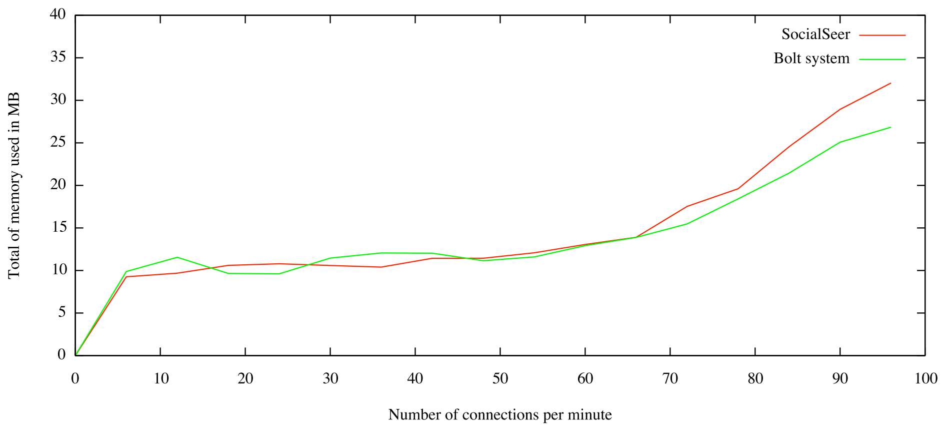


Figure 4.2: Total of memory used in MB

sources. Therefore, we conclude that, regarding the metrics used, the components introduced do not have a significant impact on performance, compared with *Bolt* system.

4.3 Latency test results

As it was mentioned before, this test aims to measure and compare the latency in both systems. To this end, we measured the time a file takes to be synchronised from one machine to another in two different scenarios: one single document and a set of multiple documents.

In Table 4.1 is presented the latency measured for both systems and for both scenarios. First of all, regardless of the system, the table depicts a significant difference in the latency, between the two scenarios. The reason for this difference relates to the fact that, although both total 75 MB, the second scenario involves having to send many small files, which leads to more computational resources consumed in context switching. By context switching we mean that for a single file the file path is handled and it is only one file transfer, whereas for multiple files we have to initiate a transfer process for each file.

Regarding the first scenario, we can note that the latency difference between our system and *Bolt* system is small. Nevertheless, this difference can be explained by two reasons: our system needs some extra processing such as indexing; and our system involves sending two files because of the auxiliary metadata file.

The same happens in the second scenario. In the table we can observe that for multiple documents the difference between the two systems is greater than for the a single document. Once more, this difference can be explained by the same reasons as above: our system involves increasing to the double the number of files to send i.e in the second scenario means sending more 287 of auxiliary files and consequently indexing process.

The main conclusion we can draw from this test is that our system's added components do not imply a significant impact on the latency comparing to *Bolt* system.

Table 4.1: Latency table

System	Single document	Multiple documents
Bolt	15min 05s	15min 53s
SocialSeer	15min 11s	16min 14s

Chapter 5

Conclusion

In this closing chapter, we present the main conclusions we can draw from this work, as well as an overview of the major contributions made and some work to be done in the future in this context.

5.1 Conclusions

This work attempts to address the objectives set forth in Chapter 3, which focus on ways to help the user to deal with information that is subject to by allowing the organisation, cataloguing, correlation, filtering and sharing of information. We believe we have been able to accomplish these objectives by devising some mechanisms based on our working set of documents metadata.

As a proof of concept for the system proposed, we implemented a prototype that comprehends all these concepts and, in addition, provides a platform fully integrated into our working environment. As the main features of the prototype we can point out on the ability to collect data and metadata from our working set of documents, by the prior definition of a collectable set of metadata attributes. Furthermore, it provides a mechanism that enable us to know what is our friend's opinion about data we have in common by simply disseminating to our circle of friends, the rating we gave to those documents. Finally, the prototype delivers sharing and filtering mechanisms offered by the two document recommendations mechanisms. The first one, allow us to manually share documents with our friends i.e the *user-driven document recommendations.* While the second one, makes *automatic document recommendations* by picking documents from our friend's public documents, based on our inferred user profile.

There is yet another feature that does not make part of the prototype but that is intrinsically connected with it. It is the *star rating plug-in*, which offers a simple way of rating our documents within the operating system.

The devised system is composed of some features that can not be quantifiable. One of such features, is the *automatic document recommendations.* Objectively evaluating such a system is very hard, as a consequence our tests were mainly focused on the quantifiable aspects of our system.

Furthermore, the prototype was built on top of an already existing system called *Bolt*, thus we decided to evaluate the impact that the added components have on the performance of the *Bolt* system. Comparing the results obtained, we can conclude that the added components have little impact on performance, in comparison with Bolt.

Finally, we strongly believe that our approach is valid and that such systems are the way to go in the future in the sense of interconnecting user's desktops and to bring the web into the desktop, in order to help the user to better deal with the information available. The set of similar undergoing projects supports our opinion.

5.2 Contributions

There are some contributions that can be drawn from this dissertation, which can be aggregated in three main contributions that we will following describe.

Prototype

The prototype is in itself a contribution because, as it has been stated in Chapter 2, there is still no system providing the same kind of features and, so far we only have undergoing projects that share similar goals.

The developed prototype delivers a platform, seamlessly integrated into the desktop, based on four vertexes: organisation, correlation, filtering and sharing of information.

In order to accomplish these four keywords, the prototype offers some key features: metadata manipulation, document rate sharing, automatic document recommendations and user-driven document recommendations.

Set of the necessary metadata attributes

Metadata can be extracted from our documents and attempts to define a set of metadata attributes have already been made. However, these sets are in fact quite large and are intended for use in operating system's search engines, thus include attributes that are not needed for the scope of this work. As a result, taking into account the objectives set for this work, we picked those metadata attributes we believe are the necessary to fulfil the goals proposed, giving rise to the set of necessary attributes. In addition, we also defined a new attribute that we think is necessary to support the features presented in this work.

Star rating plug-in

In the context of this dissertation, we verified that there was no simple way of rating documents within the diverse operating systems. Consequently, we developed an operating system embedded plug-in, which introduces a new way of simply rate documents as well as a practical and user-friendly way of visualising it.

5.3 Future work

In this work, we presented our approach of how to integrate the Web with the desktop and at the same help the user to cope with the information overload. However, during this thesis, there were some ideas that could not be included in this work, but definitely are to be studied in the near future, such as those presented below.

When inferring the user interests and thus make data recommendations, we only take into account the documents we have within the monitored folder. However, we can think of other sources of information like the user most visited websites, the searches made in the local and the web search engine or take one step further in the *Open Social* to retrieve the profile of the users. Consequently, we would get a more accurate *user profile* and thus issue more accurate and relevant data recommendations. Another mechanism we believe would be interesting, was whenever we wanted to add some tags to a document, the system would automatically propose suitable tags based on the contents of the document and on our previous inserted tags.

Additionally, we would like to have a web interface, which allow us to access our working set of documents from any device with a running browser. Furthermore, the Web interface would make available a search interface that would mimic the local search engine behaviour. In fact, the prototype is already set for this add-on but due to time constraints it was not fully implemented.

Finally, due to the difficulty in assessing some components of the prototype at the qualitative level, we wanted to set up a group of available users, which could, for some time, try the prototype and by the end of the experimental period, they could provide us with a qualitative assessment about the relevance of the recommendations and the other proposed mechanisms.

Bibliography

[1] Audioscrobbler. http://www.audioscrobbler.net.

[2] Flickr. http://www.flickr.com/.

[3] Introduction to spotlight. http://developer.apple.com/mac/library/documentation/Carbon/Conceptual/MetadataIntro/MetadataIntro.html.

[4] itunes. http://www.apple.com/itunes/.

[5] last.fm. http://www.lastfm.com.

[6] Microsoft silverlight. http://silverlight.net/.

[7] Microsoft windows azure. http://www.microsoft.com/azure.

[8] Nepomuk-java. http://dev.nepomuk.semanticdesktop.org/.

[9] Nepomuk-kde. http://nepomuk.kde.org/.

[10] Nepomuk project. http://nepomuk.semanticdesktop.org.

[11] Openmeta project. http://code.google.com/p/openmeta/.

[12] Opensocial. http://code.google.com/apis/opensocial/.

[13] Resource description framework. http://www.w3.org/RDF/.

[14] Searchkit. http://developer.apple.com/documentation/UserExperience/Reference/SearchKit/Reference/reference.html.

[15] Snowball stemmer. http://snowball.tartarus.org/.

[16] Social desktop. http://www.socialdesktop.org/.

[17] Social desktop - microsoft research. http://research.microsoft.com/en-us/projects/SocialDesktop/.

[18] Spotlight overview. http://developer.apple.com/macosx/spotlight.html.

[19] Tagit. http://www.ironicsoftware.com/tagit/index.html.

[20] Winamp. http://www.winamp.com/.

[21] Windows search. http://msdn.microsoft.com/en-us/library/aa965362(VS.85).aspx.

[22] Xmms. http://www.xmms.org/.

[23] Youtube data api protocol:. http://code.google.com/intl/pt-PT/apis/youtube/2.0/developersguideprotocol.html.

[24] Python software foundation. python programming language. http://python.org, 1990-2009.

[25] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

[26] M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[27] B.Miller, I. Albert, S. Lam, J. Konstan, and J. Riedl. Movielens unplugged: Experiences with an occasionally connected recommender system. In *Proceedings of the International Conference on Intelligent User Interfaces*, Miami, Florida, 2003.

[28] J. Bollen and H. Van de Sompel. An architecture for the aggregation and analysis of scholarly usage data. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 298–307, New York, NY, USA, 2006. ACM.

[29] D. Borthakur and D. Zhou. Hadoop and hive at facebook data and applications. Hadoop Summit, 2009.

[30] S. Brin and L. Page. The anatomy of a large scale hypertextual web search engine. In *7th WWW*, 1998.

[31] T. Chen, W.-L. Han, Hai-Dong, Wang, Y.-X. Zhou, B. Xu, and B.-Y. Zang. Content recommendation system based on private dynamic user profile. In *Internl. Conference on Machine Learning and Cybernetics*, pages 2112–2118, 2007.

[32] F. Cruz, F. Maia, J. Paulo, J. Pereira, and R. Oliveira. Bolt: instant personal working set backup and synchronization (short presentation). *In INForum, Simpósio de Informática*, September 2009.

[33] B. Fitzpatrick and D. Recordon. Thoughts on the social graph. http://bradfitz.com/social-graph-problem/.

[34] T. Gil. *Introduction to Metadata*, volume Medata and the Web. Getty Research Institute, Los Angeles, 2008.

[35] A. J. Gilliland. *Introduction to Metadata*, volume Setting the Stage. Getty Research Institute, Los Angeles, 2008.

[36] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[37] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 2003.

[38] C. McConnel and S. Sechrest. *Good Citizenship When Developing Background Services That Run on Windows Vista.* Microsoft Corporation, 2007.

[39] N. Press. Understanding metadata. 2004.