Personal Information Dashboard: Me, at a glance

João Pedro Mendes Aires

Dissertation for the Master’s degree in
Information Systems and Computer Engineering

Jury
Chairman: Prof. Mário Rui Gomes
Adviser: Prof. Daniel Jorge Viegas Gonçalves
Members: Prof. Carlos Duarte

November 2011
Acknowledgements

I would like to take this opportunity to thanks all of those who, in some way, contributed to the success of this dissertation. Without your help, this journey would not have been possible.

Many thanks to my parents for being so supportive and always being there for me. A special thanks to my girlfriend, for always being patient and for all the strength and motivation she gave me. To my brother, for his support and constructive comments. To all my relatives that always supported and believed in me. Thank you all for all your love.

Thanks a lot to all my friends, specially Tiago Garcia who always was there to help me when I need it.

Thanks to all of you that participated on the evaluation process of this dissertation, without you my work would be incomplete and lot poor.

Last but not least, I would like to thank my adviser, professor Daniel Gonçalves, for his continuous and restless dedication. His knowledge and experience played a vital role in the journey that now comes to an end.
Resumo

A nossa utilização e interacção com os computadores cria diariamente uma grande quantidade de informação sobre as nossas actividades, as nossas interacções com os outros e, no final de contas, sobre nós mesmos. No entanto, esta informação encontra-se muitas vezes espalhada pelos mais diversos locais o que torna difícil controlar e visualizar essa mesma informação. Existem já algumas aplicações que tentam ajudar os utilizadores a gerir toda esta informação, no entanto, estas abordagens são, tipicamente, muito pouco sofisticadas. No fundo, estas abordagens não vão muito mais longe do que reunir, num só sitio, informação proveniente de vários pontos da rede. Isto significa que a informação não chega a ser processada nem cruzada com informação de outras fontes. Para além disso, tipicamente estas aplicações mantêm o formato original da informação o que significa que não é acrescentado nenhum valor à mesma.

Neste documento nós apresentamos Personal Information Dashboard, uma aplicação web que propõe uma nova abordagem para este problema permitindo aos utilizadores verem, de relance, várias facets das suas vidas num determinado ponto. A ideia é revelar os padrões de comportamento dos utilizadores através da combinação de várias fontes de informação pessoal de uma forma mais natural e não textual. Imagine acordar de manhã e, apenas por olhar para o computado perceber imediatamente, não apenas quantos emails tem para ler, mas também quais aqueles que poderão ser-lhe mais interessantes. Outro cenário interessante seria descobrir quais os amigos com quem tem falado menos no último mês para que lhes possa enviar uma mensagem para reatar o contacto. Isto são apenas algumas ideias daquilo que é possível fazer. A imaginação é o limite.

Para testar a nossa solução levámos a cabo Testes de Usabilidade e alguns Casos de Estudo com um grupo representativo de potenciais utilizadores do nosso sistema. Durante os testes de usabilidade verificámos que os utilizadores não tiveram grandes dificuldades a usar o sistema e a executar as tarefas propostas. Durante os Casos de Estudo verificámos que os utilizadores foram capazes de encontrar coisas relevantes e que desconheciam. Em ambos os casos, os utilizadores foram capazes de perceber a informação que estava a ser transmitida e, no fim, obtivemos um pontuação SUS próxima de 75, o que é um resultado bastante bom em termos de usabilidade.

Palavras-Chave

Visualização de Informação, Informação Pessoal, Painel, Padrões, Interação Homem-Máquina.
Abstract

Our use and interaction with computers creates a lot of information about our activities, interactions with others and, ultimately, about ourselves. However, this information is often spread over many different places making it difficult to visualize and control. Some applications already tried to help users manage all this information however, their approaches are typically quite unsophisticated doing no much more than simply gather, in one single place, information from several sources. This means that the information is usually not processed nor cross-referenced with information from other sources. Moreover, typically the format of the information is similar to the one present on the original source, what means that these applications add no value to the information.

We present Personal Information Dashboard, a web-application that proposes a new approach to this problem allowing users to see, at a glance, various facets of their lives at some given point through this messy and scattered information we produce is a daily basis. The idea is to show people’s life patterns using and combining several sources of personal information in a more natural, non textual based way. Imagine waking up in the morning and, just by looking at the computer, immediately know, not only how many emails you have on your inbox, but also which ones are of most interest to you based on several aspects of yourself and the email itself. Another interesting scenarios would be finding out which are the friends with whom you’ve talked the less in the last month so you can text them. These and many other features should be possible to get through Personal Information Dashboard. Imagination is the limit.

To test our solution we’ve performed Usability Tests and Case Studies with a representative group of potential users. During the Usability Tests we observed that users had no difficulties in using the system and performing the proposed tasks. During the Case Studies, users were able to find relevant and interesting things that they didn’t know. In both cases, users were able to understand the information that was being transmitted and in the end, we got a SUS score next to 75, which is a pretty good result in terms of usability.

Keywords

Information Visualization, Personal Information, Awareness, Dashboard, Patterns, Human-Computer Interaction.
3 Proposed Solution 28

3.1 Requirements .................................................. 28
3.2 Solution Overview .............................................. 29
3.3 Architecture ..................................................... 30
  3.3.1 External Sources and Browser ................................. 31
  3.3.2 Themes and External Libs .................................. 31
  3.3.3 Plugins and Services ...................................... 32
3.4 Technology ....................................................... 33
3.5 Personal Information Sources ................................. 34
  3.5.1 Supported Sources ......................................... 34
  3.5.2 The Source Manager ....................................... 35
  3.5.3 The Source Piece Processor ................................ 35
  3.5.4 Keywords’ Detection Method ................................. 36
3.6 Contacts and Groups of Contacts ............................... 37
  3.6.1 Contacts’ Management ..................................... 37
  3.6.2 Group Management .......................................... 38
  3.6.3 Contact Merging ........................................... 38
    3.6.3.1 Automatic Merge ...................................... 39
    3.6.3.2 Manual Merge .......................................... 40
    3.6.3.3 The Base Merger ........................................ 41
3.7 Visualizations .................................................. 41
  3.7.1 The Plugin ................................................ 42
  3.7.2 Implemented Plugins ...................................... 42
  3.7.3 Details-on-demand ........................................ 50
  3.7.4 Plugin Intercommunication ................................. 51
3.8 Graphical User Interface ........................................ 53
  3.8.1 Dashboard .................................................. 53
4 Evaluation

4.1 Usability Tests
4.1.1 Methodology
4.1.2 Participants
4.1.3 Results
4.1.3.1 Configuration Tasks
4.1.3.2 Details-on-demand Tasks
4.1.3.3 Visualization Tasks
4.1.3.4 Plugins Configuration Tasks
4.1.3.5 Plugin Intercommunication Tasks
4.2 Case Studies
4.2.1 Methodology
4.2.2 Participants
4.2.3 Results
4.3 Users' Satisfaction
4.4 Users Feedback
4.5 Discussion

5 Conclusion and Future Work

5.1 Future Work

Appendix

A Extensibility
A1 Supporting a new source of personal information
A2 Supporting new contact and group sources
A3 Implementing a new plugin
A4 Implementing a new merger
A5 Implementing a new processor
A6 Implementing a new intercommunication feed
B  Usability Tests - Test Guide ................................................. 90
C  Case Studies - Test Guide .................................................. 94
D  Satisfaction Questionnaire .................................................. 97

Bibliography                                            100
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A <em>Conversation Clock</em></td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>The <em>PostHistory</em> interface</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>The <em>Social Network Fragments</em> interface</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>Themail interface</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>An email thread example</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>An exemple of a message tree with timeline</td>
<td>8</td>
</tr>
<tr>
<td>2.7</td>
<td>A reduced-resolution document overview</td>
<td>8</td>
</tr>
<tr>
<td>2.8</td>
<td>The <em>Newsgroup Crowds</em> interface</td>
<td>9</td>
</tr>
<tr>
<td>2.9</td>
<td>The <em>AuthorLines</em> interface</td>
<td>9</td>
</tr>
<tr>
<td>2.10</td>
<td><em>ChatCircles</em> interface. At the top we can observe a group where the user is participating.</td>
<td>10</td>
</tr>
<tr>
<td>2.11</td>
<td><em>Conversation Landscape</em> interface to visualize the conversational archive of Chat Circles.</td>
<td>10</td>
</tr>
<tr>
<td>2.12</td>
<td><em>Loom</em> showing message patterns.</td>
<td>11</td>
</tr>
<tr>
<td>2.13</td>
<td><em>Loom</em> showing content patterns.</td>
<td>11</td>
</tr>
<tr>
<td>2.14</td>
<td>The <em>CrystalChat</em> 3d structure at two different levels.</td>
<td>12</td>
</tr>
<tr>
<td>2.15</td>
<td>A <em>SpiraClock</em> over time. As time passes, we can see that the minute hand is getting closer to the next event (e.g. the central figure, missing only five minutes for this event).</td>
<td>13</td>
</tr>
<tr>
<td>2.16</td>
<td>The <em>Info-Lotus</em> interface</td>
<td>14</td>
</tr>
<tr>
<td>2.17</td>
<td><em>Blomm</em> interface. One can see the inbox view on the right side and the plant view on the left side.</td>
<td>15</td>
</tr>
<tr>
<td>2.19</td>
<td>An <em>InfoCanvas</em> representation</td>
<td>16</td>
</tr>
<tr>
<td>2.20</td>
<td>Two designs of screen savers on mobile phones.</td>
<td>17</td>
</tr>
<tr>
<td>2.21</td>
<td>The <em>CellMailGraph</em></td>
<td>18</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Feature's summary table for the Patterns and Personal Information Visualizations 23
2.2 Feature's summary table for the Glanceable Information Awareness Applications. 23
2.3 Feature’s summary table for the Frameworks and Modular Information Awareness Applications. 24
2.4 Summary Table. 25

4.1 Usability Tests results 64
4.2 Table showing the SUS average, standard deviation, maximum and minimum score value. 71
Chapter 1

Introduction

Nowadays the use of computers (in its many forms) has become a commonplace in our lives for many different reasons: to work, for pleasure, to communicate, etc. As a result, our interaction with these devices and often, by extension, with others creates an enormous amount of information about us and about what we do: the files we create, the emails we send, etc. These artifacts are part of a kind of information widely known as personal information: information from or to someone not always owned or controlled by the subject. This information is part of us, a piece of our history that can help us understand who we are and what we do. It’s our “reflection” in this digital river we all live in.

However, and thanks to the proliferation of things like the Internet and social networks, information about us can be virtually anywhere: in our brand new laptop, in our email archive, in that old newsgroups that we no longer attend, in that forgotten social network we thought extinct, etc. This high level of dispersion makes it difficult, and sometimes impossible, to keep track of all that information and see everything as a whole. For instance, the things I say and discuss via email might be related to the ones I write via instant messenger, but that information is often broken into two different sets, not reflecting this. Moreover, these sets might complement each other, making more sense when analyzed together. Finally, the level of dispersion tends to increase directly with the number of platforms, application or services we use: online social networks are a striking example of such fact. Through it all, managing all our personal information might just become a terrible nightmare and, therefore, we tend to ignore the management and value of such information, until the day we either need it or lose it (or both).

To deal with such problems and limitations, we developed a dashboard-like application we called Personal Information Dashboard. The main objective of our work is to, using and combining several sources of our personal information, show interesting facets and patterns of our lives giving an idea, at-a-glance, of what’s important at the moment. The idea is having all our information is just one place and treated as just one set, and not like a set of sets. This is a new way of looking at personal information, a cohesive way of dealing with the information we produce in a daily basis.

Personal Information Dashboard is a web application that follows a plugin based approach. Each plugin manages a visualization and the visualizations are the entities that ultimately show the user’s personal information. The idea is that the visualizations should display the information in a more graphical and glanceable way, providing a new way of looking at the information. The number of possible visualizations is immense and one of the first challenges was trying to understanding which ones were the most interesting to start with. During our
work, we have implemented a total of 11 plugins for Personal Information Dashboard and these plugins are just a small sample of what can be done: it was not our intend to fully explore all the possibilities on this regard. Based on information retrieved by several sources of personal information (emails, instant messaging, visited pages, documents, etc.) it should be possible to create several visualizations that show statistics and interesting patterns about our lives at any given point. The system’s architecture is prepared to easily support either new plugins as well as new sources of personal information.

Imagine waking up in the morning and, just by looking at the computer, immediately know what’s happening. Does anyone sent you an email while you’re sleeping? What are the last news on Facebook? Do the emails you receive have something to do with the tweets of your friends? These are just some of the things that a well organized plugin can answer in a very glanceable way.

Now imagine a visualization that uses your Facebook posts, your tweets and emails to infer the most discussed topic for the last few days. Just by looking into it you can recall your week’s concerns. Yet more interesting is to use information from the month before and remember what was happening at the time. Are you discussing the same topics? Are you discussing it with the same people? What changed? Again, a graphical visualization can help the users understand and answer most (if not all) of these questions with a low effort.

At first sight it can be argued that some applications already do something like this. Some of those applications can probably be found in section 2. In fact, some applications were able to join information from several sources and some others presented new forms for the user to observe and relate with his/her own personal information but none of them was able to combine these two ideas as Personal Information Dashboard promises to. There’s no cohesion, the information from one source is seen isolated from everything else. The personal information is not treated as a whole nowadays, changing this is the goal of Personal Information Dashboard.

1.1 Contributions

Personal Information Dashboard is a dashboard-like application focused on showing our personal information in a more cohesive and interesting way. Therefore, the main contributions of our work can be decomposed on the following key points:

- A fetching mechanism for online personal information.
- An extensible architecture that allows the easy integration of new behaviors.
- A dashboard-like interface with advanced mechanisms.
- An innovator mechanism for filtering information spread over different visualizations.
- A different way of looking at the information in order to reveal hidden patterns and interesting aspects.
- A more cohesive and centralized way of looking at personal information.
1.2 Document Structure

To support our work, we’ve first conducted a research and analysis over all the existing solutions that, somehow, can contribute or are related to our work. This survey of the state-of-the-art can be found in chapter 2. This is a crucial step as it allowed us to identify the strengths and weaknesses of those solutions so we can start building a solid foundation to our work. In chapter 3 we’ll describe our solution, showing how we implemented Personal Information Dashboard and all its underlying features. Next, on chapter 4 we’ll be presenting the methodologies we used to evaluate our prototype as well as the results of that evaluation. Finally, chapter 5 we’ll be reserved to discuss the major conclusions of our work and possible future work.
Chapter 2

Related Work

The visualization of our life’s patterns as well as the idea of being aware of transient information has been the focus of several studies and works over the years. The purpose of this chapter is to describe the most relevant (possibly the most “interesting”) of such works. More than briefly describe these works, this chapter will present the strengths and weaknesses of each of these solutions according to a set of aspects that were highlighted as being important in applications of this type. The highlighted aspects are almost as questions about the application itself and intend to draw a brief profile of the application. The selected aspects are:

- **Sources** - What are the sources of information? Is more than one source used? Emails, instant messaging and newsgroups are among the many possible sources.

- **Information** - How much useful information is presented to the user? Is this information enough to clearly show behavioral patterns?

- **Expressiveness** - How successful is the solution to present the data properly? Is the information understood almost immediately or it takes time to understand?

- **Interactivity** - Is the application static or interactive? Can one navigate the application in order to get *details-on-demand*?

- **Usability** - How easy is to use the application regarding the allowed level of information and interactivity?

- **Attractiveness** - How visually appealing is the application? How important is this aspect to the final result?

- **Extensibility** - Is it possible to add new behaviors to the application? Can and how is easy is for the user to create or find third-party addons?

Obviously, it’s possible that some aspects are more important than others depending on the type of application being studied. The related articles and applications were divided into the following groups/sections: Patterns and Personal Information Visualization, Glanceable Information Awareness Applications and Frameworks and Modular Information Awareness Applications. The rationale behind this decision has to do with the fact that such works are consistent with the objectives and requirements of our research: notice that, at the end, we want to show people’s patterns in a glaceable way through a modular framework.
2.1 Patterns and Personal Information Visualization

The view of our patterns, us and our lives has been the subject of many proposals and solutions over the years. Some of these solutions will be described in this section. Typically these applications have the user’s personal information as input and, as output, a graphical representation of such information. The objective of such metamorphosis is to transform an often disorganized and dense information in a visual form much more attractive, organized and glanceable.

2.1.1 Conversation Clock

The main goal of this work is to “augment live interaction by providing a social mirror and visual history to highlight social cues and signals for conversing individuals.”[1]. The nature of co-located group interaction was explored by augmenting aural conversation using what the authors called Conversation Clocks: this clocks displays individual contribution to the conversation providing a shared social mirror corresponding to those contributions over the course of interaction. In short, a social mirror records the conversation detecting each time a participant speaks and then represents this contribution.

Figure 2.1 presents the final look and feel of a Conversation Clock. The participant’s contribution to the conversation is represented by several concentric bars painted with an unique color for each one of the participants. The lengths of these bars indicate the level of participation. The participation level of an individual was not only measured by the number of times that he/she spoke or the duration of such statements, but also by the volume of his/her aural input. This latter information is quite interesting to keep since that, for instance, the louder volumes can indicate confidence, desire to be heard, etc, which makes it useful information when the aim is to highlight social cues. Also note that, sometimes, bars with different colors can get overlapped: this indicates that, at that very moment, the individuals who’s those colors represent were talking at the same time.

To test the concept, it was used a round table and four individuals were challenged to sit and have a conversation. The conversation was, obviously, recorded by individual microphones (i.e. one per individual) placed on the table and the corresponding Conversation Clock projected on the table surface so that everyone could watch the evolution of the conversation.

Strengths: The way the visual feedback is given to individuals is quite adequate in the sense that it displays the information in a clear and, not least, interesting way.

Weakness: The amount of information displayed could be higher. Also, the Conversation Clock continues to distract some individuals, diverting their attention from the conversation. Since this effect seems almost inevitable, it might be interesting to study how representative is that influence in the behavior of individuals.
2.1.2 PostHistory and Social Network Fragments

Throughout our lives, we humans tend to associate objects to events and experiences thus assigning a personal and often inestimable value to those objects. The fact that our life is increasingly digitized does not alter this need, creates however new challenges to get such social artifacts. *PostHistory* and *Social Network Fragments* are visualization tools that show the users social digital artifacts through the visualization of high level patterns of their email habits: *PostHistory* focuses on direct interactions between users with each of their email contacts; *Social Network Fragments* allows users to visualize the social network patterns that emerge due to email exchange. Somehow like photos, these visualizations are social artifacts for remembering and storytelling providing a tangible link to digital interactions letting users reflect on their patterns.

![Fig. 2.2: The PostHistory interface](image1)

![Fig. 2.3: The Social Network Fragments interface](image2)

As shown in figure 2.2, the PostHistory interface is divided into two panels: on the left, the calendar panel and, on the right, the contacts panel. The first one shows the intensity of email exchanges over time and the second one, the names of the people with whom the user has exchanged email. On the calendar panel, each square represents a single day and its size it’s proportional to the quantity of email received on that day. Also, the brighter the color of a given square, the more directed to the user were the emails of that day (for instance, an email sent only to the user is “highly directed” and an email sent to a mailing lists is “not directed at all”). Regarding the contacts, there are three possible representations: vertical, circular and alphabetically. The user can also click on a person’s name on the contact panel to see yellow squares displayed on top of each day that the person has sent an email, or click on a specific day on the calendar to get the people names highlighted. There’s also a animation that simulates the passage of time.

*Social Network Fragments* is an interactive animation showing data evolving over time. Its interface is also divided into two panels (see figure 2.3): the primary social network panel and a history panel. The history panel describes each time slice with two squares. The outer square indicates the number of connections that occurred during the period of time and inner square indicates the number of knowledge ties. During the animation, the current time slice is highlighted. The user can click on a time square to force the application to continue the animation from that point. The network panel shows the network, but only people actively
communicating with the user during a given time slice are shown in this view. Names with larger font represent individuals strongly tied to others and to the user. The user can click on an individual’s name to see just that person’s frequency of connections over time slices. It’s also possible to do zoom-in to certain regions of the network.

**Strengths:** Interactive and, experiments show, quite effective and exciting to use.

**Weakness:** Too high level: users may be interested in patterns, but they would appreciate to see what actions (in this case emails) led to these patterns. The network panel is quite messy.

### 2.1.3 Themail

The authors of this article[26] propose *Themail* as an application to allow the visualization of the contents of a person’s email archive. The main goal of this application is to allow a visualization that portrays relationships among individuals using the exchanged messages preserved in email archives. What the authors claim is that with the increasing number of interactions occurring via email, this information has become a “valuable records of people’s relationships”. Unlike other email visualizations that only look at header information, *Themail* relies on the content of the messages to build a visual display of interactions between individuals over time providing clues about the topics individuals discuss. With such goal in mind, the application uses an algorithm to calculate the topic words which are then arranged in columns (a column corresponding to a month or year) along a timeline. Large faint words on the background represent the most used terms by the user with a friend over an year and yellow words on the foreground over a month (see figure 2.4). The selection and size of the words to be displayed are based on the frequency and on how distinctive the word is to this specific relationship when compared to the rest of the email archive. The colored circles represent email messages exchanged: the circle size stands for the length of the message and the color for the direction (i.e. incoming or outgoing). Figure 2.4 illustrates how someone can get the messages associated with a particular word just by clicking on that word (on this case, the word “performance”). The email messages that contains such word will then appear in an information box (figure 2.4, on the left)

---

**Fig. 2.4:** Themail interface
This tool gives us the ability to follow our email messages history and understand, following the timeline, which topics were most discussed throughout this time and watch the underlying trend. The authors also asserted that this application released the storyteller that’s inside each of us and that, in many cases, users recall events and interactions of which they simply no longer remembered. “It is like looking through a photo album”, some users said. This paper also reveals that two modes of using this system (which the authors called the haystack and the needle mode of interaction) had a substantial number of usage patterns emerge.

**Strengths:** The possibility of reviewing our past through our email archives reveals to be quite interesting and the way this story is shown is truly faultless in my opinion. Apart from anything else, the fact that we can see the email that led to the presented results makes the application quite interactive since we can dig a lot of our past with a few clicks.

**Weakness:** The calculation of the terms to display is not perfect. For instance, not all messages are created equal, and some are more important than others. This difference, as well as many other aspects related to the processing of email content, is not (well) supported by Themail.

### 2.1.4 Email Visualizations to Aid Communications

Although our use of email has evolved, email clients have not been able to keep the pace and are pretty much similar to the ones we had thirty years ago, claim the paper’s authors[19]. Thinking about reinventing the email client, the authors have focused on three features that may be visualized: message threads, time and the document’s content.

For the message threads, it was used the tree metaphor to represent messages, replies to that messages, replies to those replies, and so on (see figure 2.5). Each message is a node and its color reveals the sender-recipient relationship (yellow for the recipient’s messages, purple for someone outside of the recipient’s company, red for the recipient’s manager, etc).

The approach for the next feature, time, was to combine the message trees with a timeline (see figure 2.6). The vertical yellow lines are day boundaries and the space for each day is allocated depending on messages received on that same day. The text in the middle band is the thread’s subject.

Finally, for the last attribute, which refers to the content of the document, the idea is to convey a reduced-resolution overview of the email’s content: dates are listed at the top and message bodies’ overviews are shown in the column below. The authors think that reduced-resolution overviews can help people find the documents they want.
**Strengths**: The reduced-resolution overview is a quite interesting idea.

**Weakness**: Low interactivity. It would be interesting to have all the views linked so that we can navigate them. Also notice that a long thread might induce chaos.

### 2.1.5 Newsgroup Crowds and AuthorLines

This paper [27] presents two different tools, or *visualizations*, to help the users of social cyberspaces (such as Usenet) to get, at a glance, a better understanding of their online peers. The motivation behind these two tools relates to the fact that these spaces are, often, saturated with low quality information which hinders navigation. What this article suggests are tools that can help users *separate the wheat from the chaff* when using spaces like this.

![Fig. 2.8: The Newsgroup Crowds interface](image1.png)  
![Fig. 2.9: The AuthorLines interface](image2.png)

*Newsgroup Crowds*, the first tool, graphically represents the authors participating in a particular newsgroup (see figure 2.8). Each author is represented as a circle placed along two axes: the vertical axis represents the number of days an author has been active in any given month and the horizontal axis the author’s average number of posts per thread. The color of the circle is another dimension that reflects how recently this person has posted: the brighter the fresher. It’s also possible to get more information using the author information box (semi transparent box next to the yellow circle). Using this visualization, the authors were able to distinguish four different patterns of behaviors for authors.

*AuthorLines*, the second visualization, made the obvious next step allowing to focus on a single author and get a much deeper understanding of this person’s posting pattern. This visualization works similarly to a histogram showing the intensity of posting of a particular author over time (a year-round). On figure 2.9 we can a see a horizontal timeline of one year with vertical white lines dividing the year into months. Circles are lineup for each week each one representing a conversation thread to each the author contributed. The size of the circle is proportional to the number of posts on that conversation. Finally, orange circles above the timeline refer to threads initiated by the user while the yellow circles refer to all the other threads to which the author only contributed. Although the authors did not explicitly explain this decision, it can be argued that authors knew, from the start, that this separation would be important to help distinguish posting patterns. Is then possible to determine the periods in which the author has posted more or less intensively, the threads in which the author was more active, etc.
Strengths: The emergence of patterns and its easy recognition are a reflection of the effectiveness and efficiency of the proposed solution.

Weakness: A major flaw is not being able to actually get the content of the messages. Although this failure may not compromise the determination of posting patterns, the truth is that ultimately limits the perception that users have of each other since they can’t perceive the quality of others posts. Another desirable feature would be the possibility of making side-by-side comparisons of various authors.

2.1.6 Visualizing Conversation

The main challenge of this work[4] is how to design graphical representations of persistent conversations (such as newsgroups, mailing list, chat, etc). To do that, the authors present two different projects with different objectives in mind.

One of them is Chat Circles, a graphical interface for chat conversation that is able to create visual records of conversational patterns. On Chat Circles each participant has a circle with an unique color, as an avatar. For a participant to communicate with another, he/she must move his/her circle along the screen close enough to the circle of the target participant. This means that participants that are not sufficiently close to each other, will not be able to read what each other says and, therefore, will not be able to communicate. Using this tool is possible, for instance, to see the appearance and disappearance of conversational groups. The authors have also developed Conversation Landscape: an interface to visualize the conversational archive of Chat Circles. Here, vertical lines show the activity of one participant and the horizontal lines correspond to messages. The figure 2.10 and 2.11 show both graphical interfaces.

![Chat Circles interface](image1.png)

**Fig. 2.10:** ChatCircles interface. At the top we can observe a group where the user is participating.

![Conversation Landscape interface](image2.png)

**Fig. 2.11:** Conversation Landscape interface to visualize the conversational archive of Chat Circles.

The other project is Loom, a visualization tool for Usenet groups that creates visualizations of the participants and interactions on that space. The two main goal of this project are: provide a visual interface for browsing the newsgroups archives so the viewer can perceive the social patterns; and develop visualizations that would let the viewer quickly understand the atmosphere of each group. To do so, participants are listed along one axis while time is the other axis and dots are placed on the grid representing participants’ messages. It’s also possible to trace connections between sequential posts in a thread (see figure 2.12). Messages can be
classified into categories (angry, information, etc), reflecting the user’s state of mind, which are then displayed as colored dots (see figure 2.13).

![Figure 2.12: Loom showing message patterns.](image1)

![Figure 2.13: Loom showing content patterns.](image2)

**Strengths**: The approach taken in *Chat Circles* is very interesting and seems to actually meet targets. The *Loom* presents a very interesting new way of looking into a group.

**Weakness**: In terms of visual beauty, none of the tools is exactly spectacular. In the case of *Loom*, categorization reveals itself in a very early stage of development, it’s a reasonable and interesting starting point but there’s much work to be done.

### 2.1.7 CrystalChat

While other social visualizations often focus on social networks, “CrystalChat focuses on the social interaction centered around one person, supporting a self exploration of one’s own chat history” [23] revealing the patterns associated with an individual.

This goal is achieved by incorporation the text messages in a 3D structure that integrates an individual’s chat interaction pattern with a temporal trace of history, creating an interactive visualization. The *CrystalChat* structure (figure 2.14(a)) consists of several facets (figure 2.14(b)). Facets contain all the conversations, messages, etc, between the user and a given friend. Each circle corresponds to an exchanged message and its color indicates who sent the message (gray for the user, other color for the friend). Each row of circles is a conversation and the bottom conversation is the oldest one. Using this structure is possible to visualize the several conversations we already had and find out, at a glance, very interesting aspects of our pattern of conversation: how many conversations we initiated, what are the people with whom we talk the most, etc.
Strengths: The information provided, although it is just about chat messages, is quite complete and is organized in a very interesting and intelligent way providing a fairly large palette of information about user’s conversation pattern.

Weakness: The possibility of the 3D structure of accommodating information is directly proportional to the complexity of navigating and manipulating it. Therefore, as it was expected, the use of the 3D structure can become quite challenging and/or frustrating.

2.2 Glanceable Information Awareness Applications

This section describes some works or applications that fall within the scope of information awareness. The reader will notice that there are several ways of keeping the user aware of information and also various sources of information that the user would like to be aware of.

2.2.1 SpiraClock

In today’s world, our routine is full of appointments and events of all kinds: meetings, medical appointments, pick up children at school, etc. When it comes to managing all these events through a software application (the software organizers), these programs usually use one or both of the following approaches: a static calendar and/or pop-up reminders. However, both approaches have obvious disadvantages: the static calendar is useful for organizing and consulting these events, but accessing them has a cost since they can’t be continuously displayed on the screen; on the other hand, pop-up reminders are intrusive, sometimes annoying and it’s just not a natural way to anticipate events, according to the author of this work.

By displaying the nearby events inside a spiral analog clock, SpiraClock allows the user to anticipate upcoming events in a more natural way (see figure 2.15). The idea is pretty simple: events are represented by colored sectors inside the clock and more upcoming events are those closest to the minute hand and farther from the center. As time moves, the spiral unwinds and
events move in a radial way. Closer events are highlighted fading out when they cross the minute hand. It’s also possible to show past or distant future events by dragging the clock’s hands as well as adjusting the degree of visibility of future events by dragging the spiral towards its center or towards its border.

**Strengths:** Simple, very glanceable, easy to use and understand. Not only allows you to have knowledge of upcoming events in a natural way as its small size means it could be used as a regular computer clock.

**Weakness:** Although one can observe past and future events, the *SpiraClock* doesn’t appear very appropriate for finding remote events or having a birds-eye view of our schedule.

### 2.2.2 Info-Lotus

Nowadays, everyone receives massive amounts of electronic messages on a regular basis. As a result, to help us deal with all this whirlwind of digital information, we often use notification systems to stay aware of incoming emails while performing other tasks. *Info-Lotus*[^31] is different from many existing notification systems in two key aspects: first, *Info-Lotus* provides an appropriate notification for the relevancy of the email; and second, *Info-Lotus* gives the user an aesthetically pleasing visualization of his/her incoming box. All emails are filtered using the basic email filter based on a set of pre-determined keyword phrase(s) and *Info-Lotus* only displays the filtered unread email information. The idea behind *Info-Lotus* is quite interesting. Here is the typical scenario the authors used to explain the way *Info-Lotus* works:

“... a user received an email from his/her manager who asked the group to take group photos. Some team members replied to the email but regarding different topics, e.g. someone replied about providing digital camera and some replied revising the picture. Then, the user started tracking that series emails by customizing the basic keyword phrase – the first email subject from the manager was “Group photo” and the rest secondary topics in replied emails were related with “Photo revision”, “Digital camera”, etc.”
The above scenario would produce something like what you observe on figure 2.16. Basic categories (red group) are placed on the right bottom corner of the screen and sub-groups (yellow or blue groups) will gradually grow up from the bottom group grouping related emails as they arrive. Notice that every filtered unread email is first stored in the basic group but, if at least three users reply to the same keyword phrase, a sub-group will eventually appear. The size of the sub-group, in flower numbers, reflects the number of times a keyword phrase was already replied: one flower represent five emails. Also, if any subgroup grows beyond the top of the screen, this group is simply not shown. *Info-Lotus* provides two more visual clues: sub-groups with already read emails are displayed as blue flowers while unread ones as yellow flowers; a flying bee above a sub-group means that an important sender (for instance, a manager) has replied that keyword phrase. Finally, the user can use the button placed near the basic group (see figure 2.16) in order to switch between two different detail-level visualizations: an obscure but glanceable one or a bigger, clear and attentive one.

**Strengths**: This application is a quite new way of allowing users be aware of what’s happening in their mailbox. Very attractive and glanceable visualization.

**Weakness**: *Info-Lotus* requires much more screen space than other similar applications. The usage of metaphors has always some risks involved: people may not understand or relate to the metaphor, people may not like the metaphor, people may not want to learn the metaphor, people may not like the use of ANY metaphor, etc.

### 2.2.3 Bloom

*Blomm* is a very recent proposal. We can think of it as being some kind of email-to-task converter that uses a desktop plant metaphor. Unlike other solutions, *Blomm* is integrated into the email system. The main idea is that, whenever a message on the inbox is starred as a to-do list item, the respective flower for that specific task will appear and grow on an external touchscreen. The color of each bloom is a reflection of its category (Money In/Out, School, Personal and Informational) which is determined manually. Yet, the authors consider that, in a near future, it will be possible to automatically generate categories by using natural language processing and filters. Users can read an email by simply touching a flower that will grow and spin showing the message’s text above it. To close the text window, users just have to re-touch the flower. Finally, to mark tasks as done, users just have to simulate the movement of plucking the flower off the plant.

**Strengths**: Simple, interesting and very user friendly set of ideas for managing a to-do list.

**Weakness**: Requiring an external touchscreen for managing a “simple” to-do list is, in my opinion and nowadays, asking too much. Little information.
2.2.4 Scope

Scope[24] is a radar-like application presenting a big circle divided into sectors, each one of them grouping a different kind of notifications. Also, by looking at figure 2.18 we can see that several small shapes can be found spread all over the main circle: these small shapes represent several kinds of notifications (for instance, a purple circle is a new appointment while overdue items have a yellow halo).

The more urgent a notification is considered, the more centrally it is placed making sure that, by looking at the center, the user immediately gets a sense of the most important notifications. The main goal of this application is to allow users to stay aware of information and notifications from multiples sources (namely: e-mail, IM, information alerts and appointments) minimizing distractions. As expected, the user can also interact with this tool to get, for example, more information about a notification or even to open the item in its native application. It’s also possible to manually modify the positions and groupings of the notifications, these actions allows the user to, for instance, change the urgency level of an item or even to delete it (by dragging the item of the Scope). The buttons on the periphery of the Scope act as filters for the items within the re-
spective sector: tasks can be filtered for overdue times. Sadly, only one filter can be used at a
time: clicking on a filter will turn off the current active filter.

Strengths: Very glanceable and extremely successful when it comes to showcasing the most
important notifications from a set of different types of information. The look and the possible
iterations with the application are two other aspects in favor of the application.

Weakness: The use of different shapes still implies a small, however unwelcome, learning/adap-
tation curve. Only one filter at a time. Little room for extension.

2.2.5 InfoCanvas

This work it’s quite different from all the others mentioned so far on this section, and that’s
exactly why it’s here. *InfoCanvas* it’s a highly personalized visualization that shows
user’s information in a abstract, elegant way that blends into the user’s natural environment.
The user can choose a theme and manipulate the meaning and behavior of each graphical item:
for instance, a sock on a clothes line can represent an unread email or the weather forecast for
today. Another important feature is that information is somehow “encoded” on every graphical
aspect, what means that it’s possible to show sensitive data to the user and someone simply
passing by would not be able to understand that information.

By simply looking at figure 2.19 the truth is
that we can only imagine the meaning of each
element. But, if we were the creators of such
representation, we would know that, for in-
stance, the color of the lady’s bathing suit en-
codes traffic (green is good, yellow is fair and
red is poor), and that the weather on the sky
encodes tomorrow’s forecast. More informa-
tion is encoded in the picture and users can
use more literal (weather) or more abstract
(traffic) representations. Due to it’s nature,
*InfoCanvas* was designed thinking about us-
ing the application on a secondary display de-
vice, and not integrated in the user’s computer
desktop. In fact, *InfoCanvas* can also work as
a painting on the wall. It’s a form of “Inform-
amtion Art”, claim the authors.

Strengths: Quite personal and attractive information representation with support for sensitive
information. Users stay aware of their information in a non intrusive way while performing
primary task. Multiple sources of information.

Weakness: No interactivity and no *details-on-demand* at all. Suitable only when a general
sense of the status is needed. Requires the use and presence of a secondary display device.
2.2.6 Mobile Phones as Tool to Increase Communication and Location Awareness of Users

Based on the fact that the mobile phone is nowadays a ubiquitous technology, the authors of this paper[21] revealed how it’s possible to use the mobile phone screen saver to let users aware of their information. This paper presents some examples of designs that provide peripheral awareness of a person’s communication patterns, some of those examples are shown and discussed in the following lines.

![Solar system design](image1)

(a) Solar system design

![Aquarium design](image2)

(b) Aquarium design

**Fig. 2.20:** Two designs of screen savers on mobile phones.

On figure 2.20(a) we can see the Solar system design case. Here, each planet is assigned (by the user) to one contact and the more messages were exchanged with that person, the bigger the planet is. The planets also moves around the sun and the velocity of such translational motion is a reflection of the amount of calls that took place with that contact. Other parameters (activity of background stars, rotation of planets, appearance of comets) can be configured to map more information. Another interesting design case is the one on figure 2.20(b). On this prototype, we can have several fish moving around the screen. Each fish represents a contact and the size of the fish maps the amount of money spent on communications with this person. Also, the faster the fish, less time has passed since the last call. The direction of the fish movement reflects the direction of the initiation of calls: a fish swimming from left to right means that you call this person more often than you are called. Aspect like bubbles from the fish’s mouth or plants in the water can convey further information. Notice that, pretty much like the work presented on 2.2.5, this designs are totally abstract which helps to protect the sensitive user’s information.

**Strengths:** Interesting and fun way of showing information. It may contains a lots of information.

**Weakness:** Limited to only one kind (source) of information.
2.2.7 CellMailGraph

*CellMailGraph*[6] is a cell phone application that’s worth mentioning thanks to its simplicity and room for improvement. *CellMailGraph* is a monitoring and alert system that represents unread email in an email inbox. On its first use, the user must determine his/her own settings: specify an email account, a list of contacts and/or a series of keywords. The current visualization (figure 2.21) shows dot in different colors and sizes placed along a series of concentric rings. Each dot is an unread email, it’s size its proportional to the number of keywords they match and the color identifies the sender (grey for unmonitored contacts). Also, the older they get the farther from the center and more transparent they become. The concentric rings comprise a period of 24 hours. Finally, the keywords found in any message are written on the screen in the color that identifies the sender.

**Strengths**: Very simple and quite glanceable visualization of a mailbox. Although it’s a cell phone application, it can be easily adapted to a desktop application/gadgets.

**Weakness**: Obvious limitations associated with a mobile handheld device. Users have to remember which color corresponds to each contact. Users must use other means to actually see the email’s content.

![Fig. 2.21: The CellMailGraph](http://onthespot.info/cellmailgraph)
2.3 Frameworks and Modular Information Awareness Applications

This last section presents a set of frameworks and modular applications. These applications are extensible in the sense that they allow the user to use plugins in order to be aware of certain sources of information. While the previous section described stand-alone, closed application, here we present open solutions that can be extended to obtain new behaviors and information.

2.3.1 Transient Life

Today, we not only produce large amounts of information about ourselves and our lives but that information can also be created and obtained from various sources: e-mail, IM and blogs are just some of the usual suspects. This huge amount of means by which our information can flow, coupled with our tendency to share all this information, turns out to require an extra effort to maintain and manage all this information. Transient Life\[^{22}\] tries to solve this problem by, not only collecting that information, but also by ultimately facilitating the publication of it. Transient Life is a modular sidebar located on the display’s periphery that allows the user to: update his/her current personal state (feelings, location, etc), manage a “to do” list as well as a list of already done activities, collecting interesting photos and URLs and write about things that had drawn his/her attention. Unlike other similar applications, Transient Life is not just about collecting, managing and visualizing information, but also about help sharing that information.

**Strengths**: The possibility of not only see but also modify and add new information about ourselves and then share (through various means) with the “rest of the world”, makes Transient Life a very complete and useful application.

**Weakness**: The aspect of the graphical user interface could certainly be improved to make it more appealing to new users: even a simple change of color scheme would do wonders for the appearance of the application. Another interesting and desirable feature, which the users pointed out, it would be the ability to synchronize data between multiple computers: if I use Transient Life at home and at work (on different computers) I would like to have the same information in both places. Finally, the application seems to still have some minor bugs.

2.3.2 Sideshow

Sideshow\[^{3}\] is a sidebar application designed to help users stay aware of important information. The Sideshow sidebar resides on the edge of the screen and can accommodate a large number of, what the authors called, “tickets” which display a small summary of information. On figure 2.23, we have an Outlook ticket (which remember the user that he/she has a meeting in about 23 minutes), and Contacts ticket (informing the user about the amount of online/offline/unavailable friends), a Stock ticket (giving current information on the stock market) and so on.
The application can display virtually any information (incoming e-mails, weather, traffic, etc) it’s only necessary to use the tools and the distribution processes the authors provided to create a ticket able to collect, process and present that information. To allow this extensibility, the authors have released a SideShow SDK and designed the application tickets such that they could be distributed as files.

In addition, the application was developed to provide all that information in the least intrusive and distracting way possible: the authors indicated three different ways of interrupting people with information (by pools, alerts and peripheral awareness)[3] and argued that each one has its uses and that the “ideal interface” must make the best use of the characteristics of each of these strategies. Another important design principles that we would like to highlight, are as follows: making the application always visible (thus making full use peripheral awareness), provide little information allowing details-on-demand (thus presenting the least possible information that can then be consulted in greater detail by the user) and making the application extensible (allowing the creation of new Sideshow tickets) and scalable (so that the sidebar can support a large number of tickets).

**Strengths**: Lots of information with details-on-demand, extensibility and scalability are some of the aspects that make Sideshow an interesting application.

**Weakness**: It’s just a prototype, what means that is still much work to do before it can became a “real application”. Aspect is now outdated.

### 2.3.3 Google Desktop (version 5.9)

*Google Desktop*[^2] is a desktop search software developed by Google which had its first version released on October 14, 2004. Since then, the application has had a large number of versions and is currently available not only for Microsoft Windows but also for Mac OS X and Linux. Although its primary job is to index user’s files so the users can latter on make a text search among those files, in this dissertation we are more interested in discussing another feature present in this software: the Sidebar (see figure 2.24). The Sidebar resides off to one side of the desktop and can hold several gadgets. Gadgets are interactive mini-applications (that can not only be docked in the Sidebar but also placed anywhere on the user’s desktop) that can show virtually any information: new emails from the user’s Gmail account, weather for a specified location, photos, personalized news and much more. Google not only offers a gallery of available gadgets for download, with a nice graphical interface for browse and download gadgets, but they also have an SDK and an official blog for anyone who wants to write his/her own gadgets. Information such as emails or news are displayed in summary forms (typically, the subject or title followed by the first words of the email or article) and the user can then click on this email or news to see the entire thing. Apart from the gadgets, the application works pretty much as expected. For instance, with the auto-hide mode turn on, the sidebar will only appear once the user moves the mouse cursor towards the site where the sidebar is placed overlapping all the other windows.

[^2]: http://desktop.google.com/
**Strengths:** A very professional, powerful and useful application with the Google’s signature on it. The great diversity of available gadgets and the possibility of creating our own are evidence of the application’s liveliness.

**Weakness:** “With great power comes great responsibility”: Google Desktop continues to be haunted by privacy and security issues and the usage of third-party gadgets remains a cause for concern for many uses.

### 2.3.4 Rainmeter (version 1.2)

Rainmeter\(^3\) is a desktop customization platform created to show information like CPU load, memory utilization, network traffic, email, RSS feeds and weather forecasts in a graphical way. All this information is displayed in compact applets that can float freely on the user’s desktop so the user can place all of them on the most appropriate place. On figure 2.25 we can observe the use of several applets, not only the one mentioned above, but also applets showing a calendar, today tasks or a to-do list. It’s also possible to modify the look and feel of the application by creating a new skin or, alternatively, by searching the web for one. The process might not be simple but it come from the great flexibility this application has.

![Rainmeter Applets](image)

**Fig. 2.25:** One of the possible skins for Rainmeter demonstrating how nice can look your desktop.

**Strengths:** If used properly, Rainmeter can lead to a very nice and organized desktop. Quite large customization possibilities.

**Weakness:** There are still some stability issues and users reporting some bugs and crashes. The personalization feature requires time that most users may not be willing to spend.

\(^3\)http://rainmeter.net/RainCMS/
2.3.5 iGoogle

iGoogle\(^4\) is a customizable starpage by Google launched in May 2005. In short, iGoogle is an homepage that the user can fill and customize with handy little gadgets like to-do lists, calendars, news feed, and much more. This means that, unlike other applications and services mentioned in this article, iGoogle is not a desktop application but rather a browser-based application.

To add a new gadget or choose a theme the user just needs to click on the “Add stuff” link and select the desired gadget or theme. Then, through drag-and-drop users can easily arrange and change the position of the gadgets. There is a very large range of available gadgets and the open API stimulates the emergence of new ones. Another interesting feature of iGoogle are the tabs: tabs allow users to organize the start page based on interests. The user can, for instance, add a tab named “sport” and iGoogle will automatically add gadgets related to sport on that new tab.

![iGoogle](image)

**Fig. 2.26:** An example of an iGoogle startpage.

**Strengths:** Easy to use. Lots of available gadgets and an open API. We can access our information through any computer connected to the Internet without having to install and/or configure.

**Weakness:** The service is decoupled from the desktop, what means that the user must actively open/check the browser window to see if there’s something new. Requires Internet access.

2.3.6 Discussion

Tables 2.1 to 2.3 summarize the features and characteristics of each work presented in the previous section. Such dimensions were chosen according to what was considered relevant and more important for the type of work under study.

In Table 2.4 we can find a brief summary of the comparative analysis in this section using the key aspects identified on section 2. For each pair work-aspect was assigned a score given on a 1-3 scale (using • , • • or • • •).

---

\(^4\)http://www.google.pt/ig
Tab. 2.1: Feature’s summary table for the Patterns and Personal Information Visualizations

<table>
<thead>
<tr>
<th>Application</th>
<th>Organization</th>
<th>External Sources</th>
<th>User Info</th>
<th>Others Info</th>
<th>Details-on-demand</th>
<th>Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversational Clock</td>
<td>Temporal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PostHistory and Social Network Fragments</td>
<td>Temporal and Spatial</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Themail</td>
<td>Temporal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Email Visualization to aid communications</td>
<td>Temporal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Newsgroup Crowds and AuthorLines</td>
<td>Spatial</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ChatCircles</td>
<td>Temporal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Loom</td>
<td>Spatial</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CrystalChat</td>
<td>Temporal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Tab. 2.2: Feature’s summary table for the Glanceable Information Awareness Applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Email</th>
<th>Calendar</th>
<th>Alerts</th>
<th>Tasks</th>
<th>Weather</th>
<th>Traffic</th>
<th>Conversations</th>
<th>Details-on-demand</th>
<th>Metaphor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpiraClock</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Spiral</td>
</tr>
<tr>
<td>Info-Lotus</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Flower</td>
</tr>
<tr>
<td>Bloom</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Flower</td>
</tr>
<tr>
<td>Scope</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>InfoCanvas (see 2.2.6 section)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Several</td>
</tr>
<tr>
<td>CellMailGraph</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>
Tab. 2.3: Feature’s summary table for the Frameworks and Modular Information Awareness Applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Type</th>
<th>Layout</th>
<th>Info Gathering</th>
<th>Info. Publishing</th>
<th>Details-on-demand</th>
<th>Plug-ins</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transient Life</td>
<td>Desktop</td>
<td>Sidebar</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>Section</td>
<td>✗</td>
</tr>
<tr>
<td>Sideshow</td>
<td>Desktop</td>
<td>Sidebar</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>Ticket</td>
<td>✗</td>
</tr>
<tr>
<td>Google Desktop</td>
<td>Desktop</td>
<td>Sidebar</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>Gadget</td>
<td>✔</td>
</tr>
<tr>
<td>Rainmeter</td>
<td>Desktop</td>
<td>Full/Panel</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>Applet</td>
<td>✗</td>
</tr>
<tr>
<td>iGoogle</td>
<td>Browser</td>
<td>Full/Panel</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>Gadget</td>
<td>✔</td>
</tr>
</tbody>
</table>
### Tab. 2.4: Summary Table.

<table>
<thead>
<tr>
<th>Application</th>
<th>Sources</th>
<th>Information</th>
<th>Expressiveness</th>
<th>Interactivity</th>
<th>Usability</th>
<th>Attractiveness</th>
<th>Extensibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversational Clock</td>
<td>Audio record</td>
<td>●</td>
<td>●</td>
<td>–</td>
<td>–</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>PostHistory and Social Network Fragments</td>
<td>Email</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Themail</td>
<td>Email</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Email Visualization to aid communications</td>
<td>Email</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Newsgroup Crowds and AuthorLines</td>
<td>Newsgroups</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>ChatCircles</td>
<td>–</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Loom</td>
<td>Newsgroup</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>CrystalChat</td>
<td>IM</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>SpiraClock</td>
<td>–</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Bloom</td>
<td>Email</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Info-Lotus</td>
<td>Email</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Scope</td>
<td>Email, etc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>InfoCanvas</td>
<td>(see 2.2.6 section)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>CellMailGraph</td>
<td>Email</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Transient Life</td>
<td>(see 2.2.6 section)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Sideshow</td>
<td>(see 2.2.6 section)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Google Desktop</td>
<td>#1</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>–</td>
</tr>
<tr>
<td>Rainmeter</td>
<td>#2</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>iGoogle</td>
<td>#2</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Legend:

1 - Multiple online information: email, traffic, stock market, etc
2 - Multiple online (see 1) and local information (cpu usage, etc)
3 - May vary, depending on the used skin.
As we saw in the previous section, the various papers and applications we presented addressed the patterns and personal information visualizations in quite distinct manners. For instance, works like *Conversation Clock* and *ChatCircles* are themselves their own source of information while others, as is the case of *Themail* or *CrystalChat*, obtaining information from an external source (email archive, instant messaging logs, etc). Another important thing about the sources of information is the type of source chosen by each one of the applications. Although the text information is still the most common type of information being used, applications such as *Conversation Clock* decide to innovate and find new ways to observe our behavior patterns using audio input. Regarding the use of textual information, there are applications such as *Loom* linking moods to the messages exchanged between users, giving an overview of the mood of the interactions between individuals. Although all works provide information about the user himself, only few of them are capable of showing information about other users. Details-on-demand and filtering are two other important aspects in the analysis since they allow filter uninteresting items and select an item to get more details, respectively[18]. Since we are talking about patterns, it’s interesting to note that, typically, applications either have both aspects or none.

On table 2.2 we can see that email is the primary concern of this kind of applications, maybe a reflection of the dependence we cherish by current email services. Since we are talking about glanceable applications it’s important to talk about details-on-demand: the glanceable applications, by nature, show the information in a very direct and summary way so the user can understand the essential part of the information in an almost automatic way. It’s therefore important allow users to obtain more detailed information when they want it. Works such as *SpiraClock* and *CellMailGraph* do not have this feature, but such failure is excused by the simplistic nature of the work. On the other hand, this feature would be important in works like *InfoCanvas*: we have many sources of information, but little depth in each one of these sources. It’s also interesting to notice that almost every application uses a metaphor to show the information. Some of the metaphors (*InfoCanvas* and the mobile phones screen savers on section 2.2.6) are even configurable allowing the protection of user’s sensitive information through not so obvious mappings. Works like *Info-Lotus* and *Bloom* might seem, at first, very similar but they are indeed quite different: *Info-Lotus* uses the flower metaphor to keep users aware of their email inbox while *Bloom* is actually some kind of task manager that simply uses the flower metaphor and the emails as source. Although they use the same metaphor and source, their intentions are different.

About the frameworks and modular information awareness applications, we have two basic types of layouts: the sidebar and the Full or Panel layout. Both have advantages and disadvantages: the extra space a panel can use to display more information can lead people to rely less on this type of application, since they have to actively change the window in order to get the information. In application such as *Google Desktop* and *Rainmeter* it’s possible to drag-and-drop the information items (gadgets, applets, etc) to any part of the screen which is a great advantage. The gathering of information and the details-on-demand features are present in all systems but only one system (*Transient Life*) allows the publication of new information. Catalogs of plug-ins are only available in two of the frameworks, maybe those two that are more commercial and widespread: is it a cause or a consequence? We have also presented desktop applications and one browser based application. With a browser based applications users just have to configure it once and then access it everywhere but such kind of approach can still be a little unnatural for some users. Finally, it’s interesting to notice the various nomenclatures used to name the information items (tickets, applets, gadgets, etc).
Generally speaking, the level of detail and depth of information that each application provides is a key aspect to consider. While works such as *Conversation Clock* show smaller amounts of information (in this specific case, only the flow of conversation and the underlying pattern) other works like *CrystalChat* or *Scope* strive to provide mechanisms that reveal a great set of information. A pattern that was detected, and somehow expected, is that the more information an application provides, the more difficult it becomes to use that application. One case where this is made clear is the *CrystalChat* that using a 3D structure eventually make its use more difficult than other applications such as the *ChatCircles/Loom* or even *Themail*. The sources of information vary but email is probably the most popular one. However, works like the mobile phones screensaver on section 2.2.6 or *Conversation Clock* use things like phone calls and aural input. All the frameworks have multiple source of information, as expected due to its nature.

Another aspect related to the amount of information available, is the expressiveness with which it is displayed to the user. Here it’s just about displaying the information in an efficiently way to the user: if the users takes too long to understand the information (because it is poorly framed or organized), the application fails because it loses the notion of information *at-a-glance*. Again, the applications with more information were those that more suffered with the expressiveness problem, was is not totally unexpected. *Themail* is an example of an application that was able to show all its information in a expressive way. High levels of expressiveness were observed in works like *SpiraClock* and *Bloom* and such aspect was very well addressed on the glanceable applications in general.

While applications such as *CrystalChat* or even *Newsgroup Crowds/AuthorLines* show some degree of interactivity, other applications simply don’t have it. One of those applications is *Conversation Clock* that only shows information, not allowing the users to interact with the application to manipulate data, view it from another perspective, etc.

Another important aspect when creating an application of this type is the *look and feel*. Although this aspect may seem rather secondary and even superficial, truth is that ends up being a very important aspect since it’s often the first thing users see about the application. And, truth is, we don’t get a second chance to make a good first impression. Following this line of thought, we can say that for (many) users the graphical user interface is the system. In relation to the presented works, we can notice a clear concern of building graphical interfaces with quite acceptable levels of attractiveness. *Conversation Clock*, *Themail* and *Info-Lotus* are just some examples of graphical user interfaces that will most certainly whet the users appetite.

Finally, the extensibility of some works (*Transient Life* and *Sideshow*) was considered low because, although it’s told in the articles that you can create new plug-ins, is not really that easy/possible to find ways of learning it. Other works such as *Google Desktop* and *iGoogle* are actually much more alive and open to extension, they also have documentation and ways of learning how to create new plug-ins.
Chapter 3

Proposed Solution

On the previous chapter of this document we’ve performed a state-of-the-art survey and analyzed a set of already existing applications that were considered relevant to our work. This analysis helped us understand, not only what the user is expecting, but also how can we surpass those expectations in order to created a whole new way of looking into one’s personal information.

On this chapter we’ll be presenting our solution, Personal Information Dashboard. We’ll start by enumerating the top requirements that were identified by us and that guided all the development process. Next we show a brief overview of our solution followed by a quick discussion about Personal Information Dashboard’s architecture. These two points will be important to help the reader understand the rest of the document. On this chapter we’ll also present all the currently implemented plugins, as well as many other details of our implementation.

3.1 Requirements

After having analyzed the several works in the previous section, and taking into consideration our objectives for this work, we’ve elaborated a list of top level requirements that guided the development of our prototype. We must reemphasize that these are top level requirements, almost like guidelines for the development process. Each one of these requirements can be decomposed into several low level requirements. The list of top requirements is as follow:

R1 Several sources of personal information should be used and combined
    This is the starting idea for our solution, the factor that distinguishes it from many other works.

R2 Mechanisms should be developed and available for accessing information
    To ease the access to the information from those sources.

R3 There must be a local cache for performance and information reuse reasons
    To locally store the fetched information, for performance reasons.

R4 The system should display multiple visualizations in order to allow the cross-referencing information
    Our solutions want to make it possible to see the user’s personal information as a whole and from distinct perspectives.
R5 It should be available a mechanism for information reuse among the several visualizations

It’s fundamental to guarantee that the fetched information is shared and reused among the several visualizations so that the same information is not fetched twice.

R6 The visualizations must be as expressive as possible and easy to understand at a glance

This is a major concern already studied in the previous chapter. If the user does not understand the visualization, the system fails.

R7 The visualizations should be configurable

Our work is all about personal information. Therefore, it’s important to have configurable visualization that can adapt to different users with different habits.

R8 Details-on-demand and filtering mechanisms should be available whenever possible/desirable

These two mechanism are two other aspects analyzed on our survey of state-of-art and they definitively play an important role on this kind of applications.

R9 The system must be easily extensible ie, it should be easy to create and add new visualizations, sources of personal information, etc

We want our solution to evolve with time in order to support new sources of personal information and present new, inspiring ways of showing user’s personal information.

R10 The system should provide some kind of catalog where the user can choose which visualizations to use

In order to manage all the available visualizations.

R11 The user interface must be attractive and simple to use

The user is expecting something attractive yet easy to use. This is an important matter that we already discussed and analyzed.

3.2 Solution Overview

With the requirements of section 3.1 in mind, we have developed a functional prototype that we’ll describe in this chapter. In this early section our idea is to give a very first concrete idea of our prototype, an overview of our solution that we’ll help the reader understand the rest of this chapter as a whole.

One of our first challenges was deciding what kind of application would Personal Information Dashboard be. After some considerations, we decided that it should be a web-application that would work, somehow, like iGoogle, the customizable startpage described in 2.3.5. The possibility of having our project being access any time, any where by any terminal with a browser and Internet access was a powerful player on this decision. Moreover, nowadays it’s already possible to create attractive and very functional graphical user interfaces for web-applications with a relatively low effort.
The main idea for our work rests on the dashboard metaphor. On our interpretation of this metaphor, the dashboard is populated by several, possibly distinct, visualizations of the user’s personal information. The visualizations can be moved from place to place and even resized at user’s will. Each visualization is associated to a plugin instance, the entity responsible by processing personal information and generating the correspondent visualization. The plugins can be configurable in order to show different aspects of the personal information and the visualizations are connected as an action in one visualization can affect others visualization.

The information used on the plugins is fetched from several sources of personal information. We currently support a small set of all the possible sources of personal information but the architecture is prepared to allow an easy addition of new sources of personal information. An important aspect of our personal information is the people who we communicate to. On our solution we simply call them contacts. We support the fetching, from personal information sources, and management of such entities. Associated to the concept of contact is the concept of group. We also support this concept.

3.3 Architecture

Our architecture is based on a flexible structure composed by two main types of entities: the dynamic entities and the static entities. The dynamic entities are those that can be added and removed from the system in order to add or remove some behavior. These are the entities that allow and support the addition of new sources of personal information, new plugins, etc. They are, in short, the extensible part of the application playing a fundamental role in the achievement of one of our majors requirements: R9 (The system must be easily extensible ie, it should be easy to create and add new visualizations, sources of personal information, etc). The static entities are those whose job is to manage all the flow of the system. They work just like glue between the dynamic entities or as common interface for the dynamic entities.

On figure 3.1 we present Personal Information Dashboard’s final architecture. As one can observe on figure 3.1, the architectural elements (rectangular boxes that represent the system’s entities) are divided into several groups (slashed boxes with elements inside and, in some cases, a name on the top left corner). Elements are them connected to other elements with arrowhead lines indicating the flow of information. Blue elements represent placeholders or examples of concrete dynamic elements. Black elements are the static elements that control the flow of information: some static elements were omitted as they were considered not fundamental to understand the architecture. Finally, gray elements are elements outside Personal Information Dashboard application included just to give some context to the architecture itself.

As depicted before, the user uses an browser to interact with the application. The browser will then communicate to an entity that rests in the server: the theme entity. This entity is the final responsible by constructing everything the user sees, with the help of several libraries located on the external libs group. In order to actually show something, this entity communicates to the plugins manager entity, the entity that manages all the information produced by the plugins. This last entity is responsible, among other things, by collecting all the visualizations produced by each plugin. The plugin entity is a core entity from our architecture responsible by processing the personal information and generating the respective visualizations. Notice that the plugins don’t directly access the sources of personal information (the external sources group), they use
the shared source entity. This entity manages all the information serving as interface between
the plugins (the entities that use the information) and each source (the entities than fetch the
information). The plugins can also use of of the many services available in the services group.

On the next subsections we’ll describe these elements and groups. More information about some
of the entities they represent will be presented further in this document.

![Diagram of Personal Information Dashboard architecture]

Fig. 3.1: Personal Information Dashboard architecture.

3.3.1 External Sources and Browser

The “external sources” group and “Browser” are external elements to the project that were
included in order to give some context. The “external sources” are the sources where the
information will be retrieved from and virtually any source can be used for that. Currently our
system fully supports 4 sources of personal information: Gmail, Facebook, Twitter and Flickr
(3.5 for more information). Many other sources can be easily added to the current version of
our prototype, for more information of how actually do that, see A1.

3.3.2 Themes and External Libs

The "theme" element is the responsible for generating the entire user interface (presented in
3.8). This information is returned to the “Browser” entity in the form of a string that a browser
can render. In order to do that, it can use “external libs” that will help in the job of creating a
more engaging user experience: as an example, we may have Protovis (the visualization toolkit
used by our project) or even jQuery UI. However notice that many others can be added and used. In fact, “external libs” contains not only the visual related libraries but virtually any third party lib that was not considered a “service” (a service is something that a plugin can use). A more monolithic approach (with these two elements together in one single element) would be possible but we consider that this division allows a much more flexible way of presenting the information to the users: different users can even see things in different ways.

The “theme” entity also has a connection to the “plugins manager” where the user-to-plugin and plugin-to-user information travels. The “plugins manager” transmits a string containing the visualization’s code of each plugin. This string may consist of html, javascript or anything else a browser can understand and work with. The “theme” entity will then place everything in place according to its own rule to generate the graphical user interface. On the other hand, the “theme” entity uses this connection to transmit any configuration the user performed regarding the plugins realm.

3.3.3 Plugins and Services

On the “plugins” group we can find the “pluginManager”, the entity that manages all the installed and added plugins. This entity manages the timings for the collection of information ensuring that all the plugin’s visualization are as synchronized as possible, so that everything can make sense as a whole. The plugins are the entities responsible for processing the information and returning the processed information in the form of a graphical visualization to the “plugins manager”. That information flows in a form of a string containing the code for generating that visualization.

In order to get the necessary information, the plugins do not directly access the sources of personal information, they instead use the “shared source” entity. This entity is the interface between the “plugin” and each “source” entity. The use of this entity has three main advantages:

- **Easier to implement plugins** Since the plugins use this entity as a black box to get information, the developer doesn’t have to worry about accessing the sources: the information is (or will be) there, just use it.

- **Information reuse** As all the information is gathered in just one place, the information may be reused which means less access to the sources and information made available quickly (since the information might be already available locally). This advantage meets requirement R5 (should be available a mechanism for information reuse among the several visualizations).

- **Centralized fetching code** Having all the fetching code in a small subset of entities means that, if anything changes in the source’s access method, the code changes are minimized. This is important since many APIs change with some frequency: new methods for new functionalities, deprecated methods, etc.

- **Less accesses to the sources** Since we already have part of the information, only new information is fetched. This is not only important for performance reasons but also because some sources impose some limits: number of request per second, amount of data transferred for a period of time, etc.

Regarding the information that flows in the connection between a “plugin” entity and the “shared source” entity, this information can take a lot of different forms according to the specificity of
this information. Typically this information will be python objects (lists, tuples, dictionaries, etc) containing items of information (emails, posts, etc) which in turn are more python objects, possibly with even more python objects, and so on. This means that the plugin must know how each type of information will get to it, as no standard form is defined. Although this may seem like a disadvantage, it’s a necessary evil to guarantee that the system will support different kinds of information with multiple forms and organization, according to requirement R1 (Several sources of personal information should be used and combined). The statements on this paragraph are also applicable to the connection between the “shared source” entity and the “source”’s entity, since that the “shared source” entity works just like an interface between the plugins and the “sources”.

Inside the group “service”, we also have the “sources”’s group. A “source” is an entity with all the logic necessary to access, fetch, store and retrieve the personal information from one source of personal information (Gmail, Facebook, etc). The nature and format of information flowing from the “external sources” to the “source” entities depends on the actual protocol and formats each source uses and supports. As an instance, we use IMAP to fetch information from Gmail and the JSON format for the Flickr objects.

Each “plugin” is also free of using the “services” group in order to do its job. These general services may consist in basically anything that could help the “plugin” in some way: a geolocalization modules, a math utils lib, etc. Some services are already available and more might be easily added in order to facilitate the development of plugins and the centralization of similar code in a “service” entity. Again, the actual format of the information flowing from these two kind of entities depends on the “service” entity. For instance, a geolocation service would return a python dictionary with one entry for latitude, another for longitude, etc, a service for detecting keywords would return a list of strings, and so on. This design choice provides flexibility for the introduction of new services with different needs and type of information.

Imagine that a plugin wants to display the main themes of email conversations for the last week. With this objective in mind, the plugin can use the “shared source” entity to request the list of emails from the last week. After that, the plugin would then use some specialized “service” that would be able to return keywords from a set of words contained on the retrieved emails. Next, the plugin would organize that information in some fashion constructing the respective visualization. Finally, that information would be collected by the plugins manager and delivered to the “theme” entity. This is a possible scenario for the mechanisms described so far.

3.4 Technology

The base of our project was developed using the Python Programming Language\(^1\), a powerful scripting language ideal for fast prototyping. As we were developing a web-application we needed to implement our http server application. For that, we used CherryPy\(^2\) a very fast and stable HTTP framework that allowed us to development our http server in an very attractive object-oriented fashion.

For the plugins’ visualization we opted by Protovis\(^3\), a visualization toolkit that uses JavaScript and SVG for web-native visualizations (ie, no need for plugins, just any modern browser is

\(^1\)The official Python website can be found at http://www.python.org/
\(^2\)For more information visit http://www.cherrypy.org/
\(^3\)Protovis can be found at http://mbostock.github.com/protovis/
required). Moreover, Protovis is quite simple to work with and capable of producing highly appealing visualization. Nevertheless, any plugin is free of using any resource to create its visualization.

For the creation of the web interface we used html, css, javascript, jQuery as well as some jQuery frameworks/plugins to facilitate the implementation of such features. The jQuery UI framework\(^4\) for instance, was used to implement the dashboard’s layout mechanism and the plugins’ resize feature, etc.

3.5 Personal Information Sources

One of the most fundamental parts of our work was meeting requirement R2 \((Mechanisms should be developed and available for accessing external information)\) as such mechanisms are fundamental to actually produce personal information visualizations able to meet requirement R1 \((Several sources of personal information should be used and combined)\). Ideally, Personal Information Dashboard should be able to use and combine informations from several sources all across the web in order to achieve its intents. During the development of our prototype, we’ve added support for some personal information sources (described in 3.5.1) but, more important than that, we’ve created mechanisms (described in 3.5.2) to facilitate the addition of support to others sources of personal information. Using this simple mechanism, we can make Personal Information Dashboard use, virtually, any source of personal information.

3.5.1 Supported Sources

Currently, Personal Information Dashboard fully supports four well known sources of personal information: Gmail\(^5\), Facebook\(^6\), Twitter\(^7\) and Flickr\(^8\). In addition to that, we also partially support Panoramio\(^9\): we don’t support the authentication process, only the fetching of public information. We found that, for our purposes, the authentication was not mandatory as many information is publicly available. Nevertheless, it can be easily implemented in the future if necessary. The choice of the sources to support on our prototype was mostly based on three key aspects:

**Popularity** We wanted sources that most of the users already use, sources with a large base of users in order to maximize the number of potential users of our system.

**Information** The quantity, quality and type of information that we could extract is important to actually find interesting information and produce meaningful visualizations.

**Open API** This is a critical requirement as an open API is fundamental to retrieve information from the original source. In addition, a simple and powerful API is always welcome.

\(^4\)The jQuery UI site is hosted at http://jqueryui.com/
\(^5\)Gmail site: https://mail.google.com/
\(^6\)Facebook site: http://www.facebook.com/
\(^7\)Twitter site: http://twitter.com/
\(^8\)Flickr site: http://www.flickr.com/
\(^9\)Panoramio site: http://www.panoramio.com/
Email, for instance, is one of the most ubiquitous communication technologies we use nowadays and a source of a lot of potential interesting information\textsuperscript{10}. It’s not a secret that social networks also have a great popularity among users: Facebook has more than 750 million active users\textsuperscript{11} and Twitter more the 145 million registered users\textsuperscript{12}. Sources like Flickr and Panoramio are also very interesting by the kind of information they contain, namely the photos and all the associated metadata that can be used to create interesting visualizations.

Regarding the authentication process, we opted by the OAuth standard\textsuperscript{13}. Using OAuth process, the users can share their private information (or even parts of it) without exposing their credentials. This means an increase of security and a better experience for the users.

### 3.5.2 The Source Manager

Requirement **R1** (*Several sources of personal information should be used and combined*) was one of our top priorities as it was the baseline of our project. However, supporting several sources of personal information is not trivial has each source has its own technical particularities: some are pieces of hypertext organized into labels, others are complex graph-like networks, etc. To overcome this challenge we’ve created the *Source Manager* entity. This entity is the base of the mechanism that allows the easy support for new sources of personal information. Technically, is a super-class that defines the interface between the sources of personal information and the rest of the system that will use such information. This class is represented on figure 3.1 by the entities named “source 1” and “source n” in the “sources” group.

Subclasses of this class should have one or more methods to fetch information and should also implement a cache mechanism for the respective source. Moreover, the subclasses should also implement an information refresh mechanism. This mechanism should refresh the information from time to time as long as the information is needed: this means that if some information was used once but is no longer required by any plugin, that information should not be refreshed. This is a important detail that must be kept into account in order to avoid unnecessary data transfer and store, saving bandwidth and space for information that is really necessary.

To support a new source of personal information it’s necessary to create a subclass of this class and implement the specific logic for that source (see A1 for more information).

### 3.5.3 The Source Piece Processor

During the development of our prototype we’ve came across this very specific need: sometimes, during the fetching of information from a source, we felt the need for intercepting all the incoming information to do something or extract part of the information. It might make sense, for instance, to detect all the words fetched by the application, or count the number of email received from a contact or from a set of contacts, etc. With such need in mind we’ve created the *Source Piece Processor* entity, the base class for all the objects that will be able to process such information as soon as it’s available. This entity defines a common interface to the process of pieces of information from a source. A piece is a general name for any atomic element of a source: emails,

\textsuperscript{10}Internet in numbers at http://royal.pingdom.com/2011/01/12/internet-2010-in-numbers/
\textsuperscript{11}According to https://www.facebook.com/press/info.php?statistics
\textsuperscript{12}According to http://blog.twitter.com/2010/09/evolving-ecosystem.html
\textsuperscript{13}http://en.wikipedia.org/wiki/OAuth
posts, tweets, etc. To actually do something to a source’s piece, a suitable piece processor should be implemented. For more technical information about this entity, see section A5.

3.5.4 Keywords’ Detection Method

If its true that nowadays we produce a lot of information about ourselves, it’s also true that most of that information is in the form of text. The email it’s just one obvious example of such evidence but we can think of many others. Moreover, even the sources based on other formats (like Flickr and its photos) are most likely to use some textual information: summary descriptions and tags are just some of the usual suspects. Such fact makes text and, at the end words, one of the most ubiquitous elements in our personal information.

With such conclusion in mind, we realized that we needed a way of extracting relevant words from a source’s piece of information. We were aiming for meaningful words that could reveal topics of conversation and not just any word (words like “the” or “is” for instance carry no interesting meaning, since they can occur in many different contexts). Using such words we would be able to summarize pieces of information and even link different pieces of information, possibly from different sources, based on their key words. With this mechanism, instead of showing the whole email to the user we can, for instance, only show the relevant words from that email, saving space and getting the user’s attention to the really important things. Therefore, we’ve implemented a key words detection algorithm. The workflow for this algorithm is in figure 3.2.

![Keywords’ Detection workflow](image)

**Fig. 3.2: Keywords’ Detection workflow**
As depicted in figure 3.2, our algorithm has three main phases. Typically, the output of one phase is the input for the next one:

**Tokenize phase** This is the first step. The input for this phase is a piece of information: an email, a tweet, a photo, etc. After extracting the text from that piece, the text is tokenize it into a set of words.

**Filtering phase** Based on a list of stopwords, the words from the last phase are filtered in order to eliminate common irrelevant words. The idea behind this phase is to eliminate words that are too common to actually mean something by their own.

**TF-IDF phase** At the end, the relative importance of each word obtained from the last phase is estimated using the Salton’s TF-IDF Algorithm\[20\], an algorithm that scores words based on their relative frequency in one document out of a collection. On this case, the “document” is the piece of information and the “collection” the set of all the pieces of information. The result is a set of pairs (word, importance) being importance the estimated importance for the word word in the original piece of information.

With such information at hand, we can infer the most important words of a piece of information, the words that summarize the whole text. If used properly (by choosing a sub set of really important words), they can be seen as some kind of tags for that information. The results are reasonably good and, best of all, the process is fully automatic.

**3.6 Contacts and Groups of Contacts**

Contacts are another very important dimension of one’s personal information: much of the information we produce is, one way or another, related to the persons we know and the persons we talk to. Therefore, supporting this dimension became a fundamental goal to our project. On this section we’ll briefly describe how these contacts can be managed in our system.

**3.6.1 Contacts’ Management**

Many personal information sources have support for the concept of “contact” (although the actual name varies from source to source). On our prototype, we have a list of all the contacts from all personal information sources configured by the user. This list is automatically populated by the system and the process is triggered as soon as the user ends configuring the respective source of personal information. The user can scroll that list, ask for more detailed information from a contact and contacts can me merged together (see 3.6.3). Also, each contact has a color associated to it: this color is automatically assigned (currently, using a random algorithm) by the system but the user can manually change it anytime. The purpose of the color is not be a universal identifier of a contact, since this would be impractical even for a relatively small list of contacts. On the other hand, it can work as a clue, a hint to help the user to identify someone or understand some pattern. Consider figure 3.3 for instance. On this visualization, each circle represents an email received and the color of the circle reflects the sender of it. In the worst case, just by looking at it, the user can automatically see that he/she has emails from three different people. In the best case, the user may even be able to identify who exactly sent him/her each one of the emails. In the average case, the user will eventually memorize the colors of the most important people (people with whom he/shes exchanges more email).
3.6.2 Group Management

Another concept present in some sources of personal information is the concept of “group”. This concept is useful to group contacts with something in common and deal with related people as they were only one entity. As in the case of the contacts, our work also has a list of groups automatically imported from the configured sources.

Currently, *Personal Information Dashboard* imports the groups from Gmail, Facebook and Flickr. Additionally, it’s also possible to create groups and add or remove contacts from a group directly from *Personal Information Dashboard*. This can be particularly useful for visualization purposes. A group has also a color that works just like the color of a contact. These two last features can be combined in order to create very glanceable visualizations. Imagine, for instance, that on figure 3.3 the darker blue circles are emails from family, lighter blue ones from close friends and orange ones from colleagues from work. Just by looking at it, the user can imagine the kind of emails he/she has on the mailbox: maybe, if I’m on vacation, the emails from work can wait longer. Or maybe not.

3.6.3 Contact Merging

The contacts’ merge it’s a very specific problem of our solution as it arises from the requirement **R1** (*Several sources of personal information should be used and combined*). Many personal information sources have their own set of user’s contacts organized in some fashion. Combining several sources of personal information implies combining those contacts’ list but, the fact is, those lists are most likely to have a lot of interceptions: many times we have the same person as a contact in more than one of those lists. The person under *john.doe@email.com* on Facebook may correspond to the person using *johndoe80* on Twitter, for instance. This relationship might be obvious to the user, but it’s not for the system.

A simple concatenation of the lists was not an option as it would result in a very poor and valueless contacts’ list with multiple contacts instances for the same person. Again, we would have for instance, a *john.doe@email.com*, a *johndoe80*, a *john.doe*, etc, that in fact are one and the same person. Such situation would demolish, or at least weaken, the idea of seeing the information as a whole. At least, at the contacts’ level. Conceptually speaking, the solution...
is obvious: the system should be able to merge two or more contacts that belong to the same person. In the next two subsections we’ll describe how do we solved this problem by exploring two distinct alternatives that complement each other.

3.6.3.1 Automatic Merge

In order to mitigate the problem presented in the last section, we have developed an automatic merge mechanism. With this mechanism, the merges are all done automatically in the background. Nevertheless, if done wrong, it can create a lot of frustration for the user. There’s no chance for error here: we cannot afford merging two contacts that don’t correspond to the same person. With that said, this means that we needed to find a totally secure way of merging contacts from different sources that correspond to the same physical person.

After some considerations and inspections of several source of personal information, we found that there’s in fact a link between most of them: the email address. When a user registers himself/herself on some Internet service, social network, etc, an email address is, typically, required to fulfill and confirm the registration process. Moreover, unlike other required informations like name, username, etc, email addresses are unique across all the Internet.

We used this last fact to actually perform automatic merges between contacts from distinct sources of personal information. The idea is pretty simple: if two contacts are registered with the same email address, they belong to the same person and can be safely merged. Obviously, a person may have several email addresses but this is best we can do given the situation. This process runs whenever a new list of contacts is fetched from a source and the user is totally unaware of this process as he/she only sees the final result: the list of merged contacts. Again, imagine that John is registered under Facebook with his email address. With this solution we can effectively bridge the gap between a Facebook user and an email user.

Technically, this automatic process is based on the implementation of a Base Merger (see section 3.6.3.3) for each pair of sources. Such option ensures some flexibility as each source has it’s own way of actually implement our base merge idea. Actually, we had some technical difficulties implementing our automatic merge idea to some of the sources we support. In the case of Facebook, for instance, the user can see his/her friends’ email addresses through the graphical interface but doing that by the Facebook’s API is not possible for privacy reasons. The idea is to prevent Spam from happening and, at the end of the day, we must agree with such decision. We had the same problem with Twitter and Flickr, only Gmail openly exposes our contacts’ email addresses (for obvious reasons).

Such technical details relaid our work and we eventually looked for alternatives for the merging criteria. We found no suitable alternative and get back to the initial idea. After some time, we found that both Facebook and Flickr supported a service for looking for a contact by its email address. Using these services, and the algorithm in 3.4, we were finally able to implement our automatic merge for the Facebook and Flickr. Unfortunately, we could not do the same for Twitter as no similar service is available through its public API.

This algorithm results in a much slower merging process than if we had direct access to the email addresses. However, is the possible solution given the limitations we have. At the end, this process works correctly but is not guarantee to perform all possible merges: sources that, like Twitter, provide no access to email address; contacts with two emails, etc.
contactsList = the current list of contacts
newContactsList = the list of new contacts from a source (Facebook or Flickr)

for contact in contactsList
    emailAddress = contact.emailAddress
    newContact = searchByEmail(emailAddress)
    if newContact in newContactsList
        merge(contact, newContact)
    else
        contactsList.append(newContact)

Fig. 3.4: Merge algorithm’s pseudo-code

3.6.3.2 Manual Merge

To handle the situations that the automatic merge cannot solve, we have developed a manual approach. Our main concern was to make this process as easy as possible without, however, compromising the functionality. After some iterations we managed to reach an effective solution. The manual merge functionality is embedded in the contact list and can be accessed through the floating panel (the Merge Panel) located on the right side of the list (see figure 3.5). One click on that panel’s button starts the process. Thereafter, the user only needs to click on the contacts that should be merged together. After having selected all the intended contacts, a click on the “End Merge” button of the Merge Panel concludes the process.

Fig. 3.5: Contacts’ List. Notice the “Merge Panel” on the right side that handles the manual merge of contacts.

For the system, this is the safest solution as it’s user’s responsibility to merge the contacts. However, doing this for a large list of contacts is not an option for most of the users. Therefore, this functionality should be seen as a complement to the automatic process, only to be used in cases where the latter doesn’t work.
3.6.3.3 The Base Merger

The Base Merger is the base class that allows the automatic merging of contacts from distinct sources of personal information. Technically, it’s a class that defines a common interface and the necessary work-flow with the rest of the system. The sub-classes should implement a method that receives two lists of contacts (the list of current contacts, and the list of new contacts to add) and, based on some source specific criteria, return the ids of the contacts to merge. To support the automatic merge of contacts from, at least, two different sources of personal information a subclass of Base Merger should be developed as depicted in A4.

3.7 Visualizations

The visualizations are a core concept of our solution responsible for displaying information to the user. In order to support this concept we have created the plugin entity. The plugin is the entity responsible by processing the user’s personal information and generating a valid visualization of such information. In this section we’ll describe the plugins we’ve already implement as well as the respective visualizations. Image 3.6 shows how a dashboard can look like, after adding some plugins to it.
3.7.1 The Plugin

As depicted in 3.3.3, the plugin is the entity that processes the user’s personal information and generates the respective graphical visualization of such information. It’s also the plugin that will manage all the visualization’s configuration. Also notice that it’s possible to have more than one entity of the same plugin. This can be useful to allow many different comparisons: the user can have each instance configured to a different period of time, to a different source of personal information, etc.

The Plugin is the base class of all the plugins. Its purpose it to define the necessary interface with the rest of the system and also some general functions that can dramatically help the development of new plugins. For more information on how to implement a new plugin, see A3.

3.7.2 Implemented Plugins

During the development of our prototype we’ve implemented 11 plugins. These plugins should be interpreted as a proof of concept, a demonstration of our idea and of our prototype’s potential. It was not our objective trying to explore all the potential: the implemented plugins are not an end, but a mean of showing what our solution is capable of. Many more plugins can be implemented and, once again, the imagination is the limit. Figures 3.7 to 3.17 display the implemented plugins’ general aspect. In the next pages we’ll describe each one of the plugins we’ve implemented explaining how, each one of them, can be used to learn something.

FriendsMap

Nowadays the people we know can be anywhere in the world and come from the more unexpected places. FriendsMap is a simple and fun way of exploring that. Using Protovis and the Google Maps API\textsuperscript{14} we’ve created a map like visualization that marks places based on Gmail’s and Facebook’s information. From Facebook, we can get the location and hometown of each one of user’s friends. From the received emails, using the “received” field from the email’s header and a geolocation service\textsuperscript{15}, we can infer the location of the contact when he/she sent an email. With this information, we can then mark those places with distinct markers (blue for location, green for hometown, yellow for location and hometown and red for email sent). Holding the mouse over a marker will display who is that marker about (notice that a marker can represent several friends that share that local). Users can then find out where are their friends from, where have they been lately and where are they right now. This visualization can also lead to interesting and unexpected findings: a friend who recently moved to another country, a friend who was born in a distant city and that never mentioned that fact, a colleague who is temporarily working elsewhere in the globe, etc.

As an example, look at figures 3.7(a) and 3.7(b). The first one shows a bird’s eye view at the level of a small country (Portugal) and, as you can see, the distribution of friends is not uniform. In fact, all this user’s friends are located next to the Lisbon’s area, although the fact that some of this contacts have their hometown in more distant zones of the country (as in the case of the contacts from Algarve). Figure 3.7(b) is a zoom into the Lisbon’s area revealing the exact

\textsuperscript{14}Google Maps is a well known web mapping service application and technology provided by Google. Its API can be found at http://code.google.com/apis/maps/index.html

\textsuperscript{15}Freegeoip is a free IP geolocation web service that can be found at http://freegeoip.net
distribution of the friends in this area. Moreover, the visualization can be configure to only show a subset of the possible markers: the user might just be interested in knowing his/her friends hometown, for instance.

**Keywords Cloud**

Tag Clouds are a great way of showing a top view of one or more text based documents. On this visual representation, the more important words are placed larger and more centrally relatively to less important words. This format is useful for quickly perceiving the most prominent terms of a set of documents and, therefore, have a better understanding of the documents itself. On Keywords Cloud we applied this idea to summarize what the user, and his/her friends, are writing about in multiple sources (Gmail, Facebook and Twitter). The user can select what sources to use as well as the time period to consider allowing a very flexible visualizations with adjustable degrees of detail. The selection of the words to display uses the methods explained in 3.5.4 and the visualization is based on an adaptation of the jQCloud\textsuperscript{16} by Luca Ongaro. Figures 3.8(a) and 3.8(b) are two examples of what we can observe with this plugin.

Figure 3.8(a) is configured to only use information from the user’s Gmail account and 3.8(b) to only use information from the user’s Facebook account. The differences are evident. For instance, “personal”, “information” and “dashboard” are the more prominent words on email conversations but they are not even mentioned on Facebook: this indicates that the user is exchanging a lot of emails about this work but that prefers to use Facebook for discussing other subjects.

\textsuperscript{16}JQcloud is an open-source jQuery plugin. The source code an documentation can be found at https://github.com/DukeLeNoir/jQCloud
My E-Emotions

The things we say carry emotions. The things we write also. And the things we write on the web are not an exception. But are we conveying the intended emotion? Are we being too aggressive? Are we sounding too sad? Fact is: expressing and understanding emotions on a text-based dialog is much more difficult than in a sound-based one. With this plugin we tried to address this problem. In order to do that, we use the Regressive Imagery Dictionary[8, 9, 10, 12, 13, 17, 28, 29, 30, 11] coding scheme (hereafter referred to by RID). RID is used for text analysis to measure “primordial” and conceptual content. Conceptual thought is abstract, logical, reality oriented and focused on problem solving. Primordial thought is associative, concrete with little account of reality. RID contains a few thousand words grouped into categories that are themselves classified as primary, secondary, and emotional. A piece of text is then classified by what percentage of its words fall into each one of the categories. Consider the following example of a small email exchanged between friends:

Hi mate,
I’m sorry but I’m not going tonight.
Hope to see you soon.

Using this piece of text as input, RID would detect the words “mate”, “sorry” as words carrying emotion: affective and sadness emotions, respectively. This simple method, although not 100% accurate, ends up generating quite satisfactory and interesting results that meet what we want without compromising the overall system performance. On My E-Emotions we use RID to analyze the pieces of text the user produces in the web (emails, tweets, etc) and infer the emotions.
For the visualization we’ve chosen a simple pie chart, useful to understand the weight of a specific emotion among all the detected emotions. A problem with the use of pie charts has to do with the fact that is more difficult for comparisons to be made between the size of items in a chart when area is used instead of length. This is particularly true when there’s small differences between item’s area. However, this is not a problem here: the method to identify emotion is not totally accurate and the use of pie charts can help “hide” it from the end user.

On figure 3.9 we can observe two instances of this plugin configured for two distinct periods of time. It’s easy to see the difference between this periods: affection (pink area) was clearly the most prominent emotion on the first period but that is not true for the second period as there’s 4 emotions with similar areas. It can also be interesting to configure different instances to use distinct sources (an instance for Gmail, an instance for Facebook, etc) and see the differences within the same period of time but in different contexts. Interpreting this differences is part of the fun provided by this plugin.

My Feeds

Keeping user’s alert of what’s import at the moment is one of our main objectives. With multiple places to look at and little time to do so, many times we end up loosing the memento of an event: a funny comment made by a friend, an import email sent by a college, etc. My Feeds is the right plugin to handle this challenge. The visualization is based on a circle packing layout provided by Protovis. Each outer circle represents the contact associated with that circle’s border color and each inner circle an “event” from that contact: email (red), facebook post (facebook’s blue) or tweet (twitter’s blue). This way, all the events from a contact are located inside the same outer circle making it easy to follow a contact’s activity. As events become older and older, they shrink. After a configurable amount of time, they completely disappear from the visualization. This way, the user is always aware of what’s happening at the moment and its attention is always directed to the most recent events. Holding the mouse pointer over an event pops out a tip with information about that event and a link to open it on the original source.

Figure 3.10 is an example of how this plugin can look like. Notice that many contacts have “done something” within the configured time range. Most of them, have only performed an event. Others, on the other hand, are quite active, as in the case of the contact that have been made a lot of tweets lately. Also notice that all contacts are only using a service: no contact is sending, for instance, an email and posting on Facebook.
Our Favorite Stuff

Who your friends are tells a lot of what you are and, what your friends like tells a lot of what they are. The conclusion is obvious: algebraically, the things your friends like tells a lot of what you are. Knowing what our friends like is not only a self-knowledge experience but also a funny way of finding out things you might just like. Facebook tells you what each one of your friends like but no general view is available. This plugin is that forgotten general view. The visualization is presented as a mosaic with the images of the things user’s friends like ordered (from left to right and from top to bottom) by the number of likes. Holding the mouse over an image will display a tip with information about that item and a link to its Facebook page.

The aspect of this plugin’s visualization is shown in figure 3.11. The use of this tool may provide quite interesting moments as we can just discover things that we didn’t know about but that are just right for us. Notice that is possible to configure the amount of things to show, making it possible to see a large number of things that you friends like. This plugin has another interesting feature: by clicking on one thing, all the things that have no contacts in common with the first thing, will dramatically fade away. As an example, imagine that the user clicks on the Sporting football team image. The Benfica image will, most likely, fade away, since these two are rival teams and nobody did liked them both. This is an obvious scenario, but the idea is to discover unexpected situations.

Photo Search

Photo Search is a simple, nice and attractive photo slider like visualization. The photos are extracted from Flickr based on a set of words that the user can configure: the user can, for instance, select the words “portuguese food” to see a set of popular photos that fit that criteria. The user can enjoy the slide-show or click on the next button (located on the bottom right area) to see the next picture.

This visualization can be useful if the user wants to be aware of something. Imagine that the user just arrived from a popular tournament in which he/she participated and is expecting photos from that event to appear in Flickr. By entering the right key words the user can be aware of what is being upload to Flickr without having to manually search it from time to time.
Spark Stats

When you intensively use tools as email you end up totally losing track of the amount of information you receive and produce on such context. Spark Stats is a simple visualization of such information that uses the concept of Sparkline\textsuperscript{17}. A sparkline is a type of information graphic characterized by its small size while presenting trends and variations associated with some measurement over a period of time. This plugin uses this lines to show measures such as emails sent and received, posts made, etc. Sparklines are the perfect way of revealing such measure’s evolution without requiring too much space.

![Spark Stats Example](image)

\textbf{Fig. 3.13:} Two instance of the Spark Stats plugin.

Figure 3.13 shows two instances of this plugin with slightly different configurations. The left most instance simply shows the measures and its evolution for the last month. The right most instance is configured to highlight the two weeks in the middle of that month, showing the values of the measures for that period. This feature can be useful to analyze and query the dynamic of such measures: why have I been receiving more emails on the last week? Is the dynamic of received emails similar to the dynamic of sent emails? When was the last time I posted on Facebook? Etc.

Stacked Memories

Nowadays, many of our photos are in the digital format and not in physical one. Stacked Memories is just like a call from the past as it organizes your photos in scattered piles of nostalgic Polaroid-like photos. The interaction with the photos resembles the old days: the user can move the photos from place to place, dig through out the piles of photos and click on some of then to see it larger. For this plugin we used the photos, from Facebook, where the user is tagged. We opted by the photos where the user is tagged in order to maximize the chances of that photos actually mean something to the user: the user was there, he/she will probably associate that photo to an event, resulting in a nostalgic moment, what is exactly what we’re looking for. Also note that, this includes not only the photo from the user itself, but also from its friends what is even more

\textsuperscript{17}See, for instance, more information about sparklines at http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0001OR&topic_id=1
interesting: we can find photos that the user did not expected making the findings even more nostalgic.

Figure 3.14 shows how Stacked Memories can look like but keep in mind that most of this plugin’s charm lies in the type of interaction it allows. Also notice that most of this user’s photos seem to be taken outdoor, what might indicate that the user is someone that like outdoor activities.

**The Surroundings**

The Surroundings is a plugin that can be pretty useful for users that travel from time to time. The idea behind this plugin is to show users photos of possibly interesting nearby places. In order to achieve that, this plugin first uses a geolocation service\(^\text{18}\) to find user’s current position. After that, we use the Panoramio site to search for popular photos of (probably) popular places nearby. We opted by Panoramio, and not Flickr for instance, as Panoramio was considered by us the most appropriate service to retrieve postcard like photos. The visualization is based on a slider that performs a never ending slide show.

Imagine that you arrived the day before in another country for a business meeting, but now you are already done. As such, you still have some free time before going to the airport. If you add this plugin to your dashboard you can, in just a few seconds, see photos of nearby place that might be of interest to you. Of course, you could just search for interesting places in the web, but that would then require a filtering by nearby places, since your flight leave in not too long. This plugin can just save you that work.

Imagine that you are a person who travels a lot (not only to other countries but also within your home country), with this plugin in your dashboard you can always be aware of places nearby that might be of interest to you.

**Who&How**

Keeping track of your activity with each one of your contacts can be difficult when the amount of data is just too much to handle. Who&How is a solution for such problem. This plugin’s visualization is based on several concentric areas arranged on the edges of a central circle. Each one of these areas is associated to one contact. Then, inside that areas, we can find a bar for each kind of source activity: one bar for the amount of emails sent from that contact, one bar for the amount of posts on Facebook made by that contact and one bar for amount of Tweets. To better understand this plugin, observe figure 3.16. During the development of this plugin, we faced with a problem that compromised its visualization: a contact with an abnormally high activity could distort the visualization in the sense that all other bars would appear too small for any kind of analysis. To circumvent this problem, we decided to use a logarithmic scale (rather than a linear one) for the size of the bars. This scale may not be as natural as a linear one but it’s definitively better to show arbitrarily different lengths while maintaining the ability

\(^{18}\)Again, we use Freegeoip. For more details go to http://freegeoip.net
to compare the sizes of the bars and, therefore, the activity among contacts.

With this visualization, the user can observe several patterns: am I receiving many emails from many contacts, or are the emails from a restrict group of contacts? Are the people who sent me emails the same ones who post on Facebook? Is there someone far more active than the rest of my contacts? Etc. Note that this plugin can be configured to show different periods of time and different sources of information, allowing analyzing things over time and taking different sources into account.

Fig. 3.16: Who&How

You’ve Got Bubbles

The idea behind You’ve Got Bubbles is having a clear, more colorful and funny view of our email box based on a bubble-based visualization: each bubble is an email. The plugin is configurable from several perspectives: users can choose what email box to observe, the state (read or unread) of the emails, the meaning of the size and the color of the bubble, etc.

Figure 3.17 show the default configuration of this plugin. On this configuration, only the unread emails are displayed, the color of the bubble indicates the sender of the email and the size of the bubble the size of the email itself in number of words. With this configuration the user can quickly understand what is happening in his/her mailbox: how many emails do I have? Are
they from the different people, or all from the same person? If they are from the same person I better check if it’s something important; I have little time now, I want to save big emails for later, etc. Many configurations can be applied: it can, for instance, make sense having the bubble’s color associated with the state (read or unread) of the email, to automatically detect the unread emails.

### 3.7.3 Details-on-demand

One important requirement extracted from chapter 2 is R8 (*Details-on-demand and filtering mechanisms should be available whenever possible/desirable*). Such mechanisms can help users better understand their own information and have a more enjoyable experience while exploring each plugin. The implementation of these mechanisms is, ultimately, a responsibility of each plugin as each one has it’s own visualization and data structure. Nevertheless, we’ve tried to unify those mechanism as much as possible.

First, we’ve defined a consistency rule for triggering these mechanisms: details-on-demand should be triggered by a mouse-hover event and filtering by a mouse-click event. We see this as a critical rule: without this rule, each plugin would trigger each mechanism in its own way, confusing the user that would need to remember how each one of the plugin triggers each one of the mechanisms. Second, we’ve created a set of resources to help the implementation of the details-on-demand mechanism. These resources facilitate the implementation of this mechanism using balloon-like tool-tips like the ones on figure 3.18. Note that the existence of such helping resources is not innocent: if the programmer has a shortcut to implement details-on-demand, he/she will, most likely, use it instead of having the work of implementing a totally new way of doing it. This fact helps improving the consistency between all the plugins. Third, the filtering mechanism can be easily implemented using the Plugin Intercommunication mechanism we developed and describe in 3.7.4.

![Details-on-demand are available via contextual tooltips.](image)

(a) The tooltip on the FriendsMap plugin reveals the contacts associated with the markers

(b) The tooltip on the My Feeds plugin shows some basic information about the original item

**Fig. 3.18:** Details-on-demand are available via contextual tooltips.
3.7.4 Plugin Intercommunication

As depicted before, one of our system’s requirements was the existence of a filtering mechanism on the plugins’ visualizations. Later, on the development phase of our work, we found that such mechanism would make sense, not only for filtering the data in one visualization, but also for filtering data among the several visualizations on the user’s dashboard. A typical usage scenario for this mechanism would be:

After examining one plugin on his dashboard, the user notices that the word “Thesis” is occurring a lot lately. By clicking on that word, all the visualizations in the dashboard change in order to highlight anything related to that word: emails containing that word, contacts that used such word, related photos, etc.

The main idea is having the plugins connected in the sense that the filtering state is all the same for all the plugins, they are all filtered in the same way making it easier for tracking down the information. Moreover, this idea is suppose to be applied not only to words but virtually any element in the visualizations: imagine, for instance, clicking on one contact and having that contact’s emails, photos, posts, etc, been highlighted in all the visualizations. Finally, this idea also has the great advantage of improving the traceability and understandability of the information as the visualization can compensate and complement each others.

The actual implementation of the idea however, required some more deliberation about the way that we would highlight the selected things. We had two main hypothesis:

Use an highlight color. On this hypothesis we would have a pre-selected, reserved color for the highlight things. This way, all the important filtered things would have the same color while unimportant things would keep their original color.

Fade-out unselected things. By dramatically fade-out unselected things, we can highlight the important ones by contrast while keeping the context and any possible meaning associated with the original colors (as only the opacity of the color changes).

At the end, we opted by the second hypothesis. Using a highlighting color has some inconveniences: we would need to reserve a color, meaning that we would need to make sure that that color would not occur outside this context; moreover, we would have to make sure that similar colors don’t occur also; also, if few things are selected, there’s no really evident contrast of was is and is not selected; last but not least, if any meaning is associated to the original color, that meaning is lost. The application of our decisions lead us to the result shown in 3.19. Here we’ve added three plugins: FriendsMap, Keywords Cloud and You’ve Got Bubbles. Next we clicked on the darkest blue bubble on the You’ve Got Bubbles plugin (the right most plugin). In response to that, the system highlight two distinct aspects about the clicked email: the sender and the subject of the message. The sender information was used by FriendsMap (the left most plugin) to highlight the location of that contact. The subject of the email was used by Keywords Cloud (the middle plugin) to highlight words related to that subject. Finally, You’ve Got Bubbles itself used that the information to highlight all the emails from that sender and/or related to that subject.

Regarding the back-end implementation of this mechanism we opted by a bus-like implementation. This bus works just like figure 3.20 shows: 1) The user triggers a filtering event by clicking
Fig. 3.19: An example of what the plugin intercommunication might look like

1. Clicking on some element in the plugin A visualization;
2. The plugin A propagates this event to the intercommunication bus;
3. Each plugin in the dashboard is notified of the filtering event and has the chance to update its visualization accordingly.

**FILTERING EVENT**

1. Event propagated to the bus
2. Each plugin (A, B, C, D) is notified
3. Each plugin updates its visualization

**Fig. 3.20:** The Plugin Intercommunication bus

Regarding the filtering event, these events have the following structure:

- **Event** = (PluginID, Feeds)
- **Feeds** = (Feed1, Feed2, ...)
- **Feed** = (EventType, EventInformation)

An event object has two components, the **PluginID** and the **Feeds**. The **PluginID** holds the identification of the plugin that triggered the event. This can be useful by many reasons as, for instance, if the plugin wants to have a different behavior in the case of being the triggering plugin. The **Feeds** is a set of **Feed** elements, each one of them with two elements. The **EventType** represents a type associated with that event: “contact”, “word”, etc. The **EventInformation** has all the specific information for the current event i.e., information of what is selected. Conceptually speaking, this field can hold virtually anything, the **EventType** will specify how to deal with it. The use of several **Feeds** instead of just one, makes it possible to trigger event with multiple
types associated. Reusing the example above, a possible Event object for that scenario would be something like this:

```
Event = ('1', Feeds)
Feeds = (Feed1, Feed2)
Feed1 = ("contact", "Jon Doe")
Feed2 = ("word", ["chillis", "lunch"])
```

This means that the user selected the contact “Jon Doe” and the words “chillis” and “lunch”. It will then be up to each plugin to interpret this event and update its visualization accordingly.

With this simple yet expressive mechanism we can support any type of filtering event (filtering by word, contact, date, etc). In short, to support a new type of filtering event it’s only necessary to specify the protocol for that type of event ie, the corresponding EventType and the structure and semantic of EventInformation. From then on, each plugin that knows that protocol will act accordingly. For more technical details and information on this regard, see section A6.

### 3.8 Graphical User Interface

As mentioned before, our solution is a web-application that the user will access through a web-browser. Therefore, our interface is entirely based on HTML elements using CSS and a lot of javascript/jQuery to convey a more enjoyable experience for the user. The interface is divided into three main areas: the Dashboard, where the added plugins are located; the Catalog, where it’s possible to add new plugins; and the Configuration Area, where the user can configure its account and manage his/her contacts and groups. Throughout this section we will further describe each of these areas. The plugin’s layout is another important aspect of the look and feel of Personal Information Dashboard. On 3.8.4 we’ll describe how the intended behavior was achieved after some complications.

#### 3.8.1 Dashboard

On figure 3.21 we have the main page of Personal Information Dashboard that contains the dashboard itself were the visualizations are placed after adding a plugin. On this case, the dashboard is empty has no plugin was added yet. We opted for a clean and minimalist interface once we wanted the users to be focus on the visualization rather than in the interface that supports all the interactions. On the top right corner we have the menu for accessing the three main pages of Personal Information Dashboard: the dashboard, the catalog and the configure page. Also notice the blue balloon-shaped tip bellow the “add plugin”. Such tip is meant to help and guide the user on his/her first usage of the system: in this case, the system detected that the user has at least one account configured but no plugin added and suggests that the user should want to add a plugin as his/her next move.
Figure 3.21 shows a user’s dashboard already populated with some plugins’s visualizations. As depicted before, each plugin has an associated visualization that is materialized, on the dashboard, as a windows-like windows where the information is displayed. The user can manipulate the position of each visualization through the drag-and-drop mechanism described in 3.8.4. It's also possible to resize the visualizations at user’s will creating a more engaging and flexible experience: some visualization need more space than other, and given the user the opportunity to manipulate the size of each visualization is very important.

As new plugins are added to the dashboard, space becomes a critical factor. Trying to minimize this problem, we’ve implemented an auto-hide behavior for the dashboard’s header (the horizontal space that contains the menu). When the user has, at least one plugin in the dashboard, the header slides up making more space for the plugins (as displayed on figure 3.22). The header slides down again as soon as the cursor moves to the zone above the horizontal line that separates the header from the plugin’s area. This way, we can make more space for the dashboard, making the menu available again when the user needs it.
To meet requirement **R7** (*The visualizations should be configurable*) we decided that each plugin would have to implement an individual configuration menu were the user can configure the visualization to meet his/her needs. One user might want to only see data from GMail, and another user to see data from all possible sources. Also, the same user might want to configure two instances of the same plugin to use distinct time ranges. A proper use of this feature can led to interesting results.

These menus can be accessed by clicking on the top right icon of the plugin’s visualization and the general aspect of this menus is displayed on figure 3.23. The menus are divided into sections and all the menus have the “Basic” section. This section holds two configuration fields. The first one is a text-box with the name or title of the plugin’s instance. This field can be useful when the user has several instances of the same plugin and wants to identify each instance. Imagine that the user has two instances of the same plugin configured to use different sources of personal information, by assigning titles like “Keywords from GMail” and “Keywords from Facebook” the user can easily identify each instance. The second field is a set of check-boxes to pick the sources of information to use. The rest of the menu will depend on each plugin’s specific set of options. Nevertheless, we prepared all the structure to allow an easy and fast implementation of new menus for new plugins.

![Fig. 3.23: The plugin’s configure menu general look.](image)

Figure 3.24 is an example of how this feature can be used to find interesting things. Both images on this figure represent the most important words from an user email account. However they are configured to two different time periods: figure 3.24(a) is configure to July 2011 and figure 3.24(b) to July 2010. This means that each one will only show data from the month to each they are configured. The resulting differences are quite interesting. On figure 3.24(a) the more prominent words are “personal”, “information” and “dashboard”, ie words related to the work we are currently presenting: the user was working a lot on this thesis and as a consequence, he was constantly exchanging emails about this subject. On the other hand, one of the most prominent words in figure 3.24(b) is “louvre”: the user was planning a trip to Paris and discussing what to visit. These differences tell the story of the user in two different moments of his life and it is interesting to see how this all makes sense.
3.8.2 Catalog

The existence of a visualization’s catalog (or, better yet, a plugins catalog) is an important requirement as the user must be able to easily explore all the existing plugins to add new ones. On figure 3.25 we present Personal Information Dashboard’s catalog that can be accessed through the “add plugin” option found on the top right menu of the interface. The catalog is composed by a horizontal list of entries, one for each available plugin. Each entry has an image of the plugin, the name and author of the plugin and a small description of it. To add a new plugin, the users just have to click on the button below the corresponding plugin’s image.

![Fig. 3.25: The catalog where the user can see and add plugins to the dashboard. Notice that the Keyword Cloud plugin was just added to the dashboard.](image)
3.8.3 Configuration Area

The usage of several personal information sources imposed by requirement R1 (*Several sources of personal information should be used and combined*) implied some way of configuring those source’s accounts. Clicking on the “configure” option of the top right menu the user can have access to a configuration page like the one represented on figure 3.26.

![Personal Information Dashboard](image)

**Fig. 3.26:** The configuration area. Currently, no account is configured.

On this page, we have three different tabs meant to help the user configure three different aspects of his/her accounts. The first tab, the “sources” tab, allows the user to configure the accounts to use on Personal Information Dashboard. The user can configure up to 1 account per source. The configuration of the sources is simple and should begin by the user introducing the account name (the email address, the username, etc) on the respective textbox and then clicking on the “connect” button on the right side. The next phase is the authentication phase. This phase uses the OAuth protocol to authenticate the user and give Personal Information Dashboard permissions to access user’s personal information.

The second tab is the “contacts” tab. Here the user can manage his/her list of contacts. When the user configures an account, the application will automatically fetch the list of contacts (contact, friends, followers, etc) for that account if that’s the case. After that process, the user can manage the final list with actions that include changing the color associated to a contact, merging two contacts that belong to the same person, etc.

The last tab is the “groups” tab, where users can manage his/her list of groups. It’s possible, for instance, to create new groups what can specially interesting for visualizations’ purposes. As depicted before, imagine creating groups like “family”, “friends” and “co-workers” and compare your interaction with people from these three distinct groups.

3.8.4 Dashboard Layout

When the dashboard is populated by multiple visualizations the layout of such visualizations might not correspond to the user’s best expectations: a plugin misplaced, a plugin too big or too small, etc. It’s just sensate to think that the users would appreciate a way to control the layout of the added plugins at their will as different users might want information displayed in different ways: one user might want to organize the plugins by type, other by time, etc.
To meet those expectations we’ve introduced a drag-and-drop mechanism on ours’ prototype dashboard. Originally, this mechanism was totally based on the jQuery UI’s sortable plugin\textsuperscript{19} framework. However, this jQuery plugin revealed itself insufficient to meet all our needs on this regard: for plugins with different widths and/or heights, if the highest one wasn’t placed as left as possible, an undesirable vertical gap would appear due to the way that the CSS float property works. This problem can be observed on figure 3.27(a) and compared with the desired, and current, behavior on figure 3.27(b). Such problem can be solved by absolute positioning the plugins’ visualizations what, sadly, tends to dramatically conflict with the jQuery UI implementation of the drag-and-drop mechanism. We considered that developing and testing an absolute positioning drag-and-drop mechanism from the start would be an inglorious job, specially since jQuery UI provides a so complete, robust and tested one. Our best hope was trying to understand how to make both things work together. After some research we found a jQuery plugin that seemed to solve part of our problem: jQuery Masonry\textsuperscript{20}, a jQuery plugin specifically developed to solve this CSS gap feature. However, making both plugins work together was not easy as none of them was developed to be compatible with the other. We invested many hours trying to solve this problem and tried a lot of alternatives until we finally discovered a way of making both plugins correctly work together. The solution we found didn’t implied any modification in any of the jquery plugins but only a small but very specific and advanced customization of some of the options on both plugins. Conceptually, the jQuery UI drag-and-drop mechanism runs first, defining the general layout of the elements on the dashboard and, after that, the jQuery Masonry plugin proceeds to the final adjustments. The challenge was achieving the desired result without compromising anything.

\textbf{Fig. 3.27:} The vertical gap between plugins was our biggest problem regarding the plugin’s layout.

As depicted before, after having added a few plugins space becomes a critical factor. Another way of dealing with this problem consists on allowing the user to resize the visualizations: if the user feels the need for more space to add new plugins, he/she can resize some of the already added plugins to make more room. Moreover, is a practical and flexible way of minimizing the problem without compromising user’s will. This feature was implemented using the jQuery UI’s resizable plugin.

\textsuperscript{19}This plugin can be found at http://jqueryui.com/demos/sortable/
\textsuperscript{20}jQuery Masonry can be found at http://masonry.desandro.com/
Chapter 4

Evaluation

The main objective of Personal Information Dashboard is to, using and combining several sources of our personal information, show interesting facets and patterns of our lives giving an idea, at-a-glance, of the relevant aspects at the moment.

In order to evaluate our approach, and therefore verify the added value of our solution, we performed two different sets of tests: Usability Tests and Case Studies. Both tests consisted on a set of pre-selected tasks and the major difference between these two tests was the information used to perform each class of test.

For the Usability Tests, the users did not use their own personal information: it was used a pre-selected set of personal information items with real and representative information from a real user. The objective with these tests was to evaluate the intrinsic quality of the interface to transmit information unknown to the user. Moreover, since the data is the same for all users, it allows direct comparisons of objective measures such as time spent to perform a task, number of errors, etc. For the Case Studies, the users used their own personal information. These tests are much more difficult to assess from an objective point of view but are also essential to correctly evaluate our solution since it’s what Personal Information Dashboard is all about: personal information. Moreover, we were also interested in observing how the users would react to their own personal information: Can they understand the visualizations? Can they find interesting things? What stories might arise from those findings? Etc. Throughout this chapter, we’ll describe the methodologies used in each one of the tests as well as the main results drawn from each one. We’ll end this chapter with a final overview of the evaluation’s process and results.

All the tests were performed on the same computer: a laptop with a Pentium Dual-Core T4200 operating at 2.00 GHz, with 4 GB of RAM running Windows 7 and Python 2.7.1.

4.1 Usability Tests

Usability testing can be seen as an irreplaceable usability practice, since it gives direct input on how real users use the system[16]. The Usability Tests consisted on several sessions (one session per user) where each user had a set of pre-defined tasks to complete. The sessions had no time limit but the estimated total time was about 40 minutes per session. As depicted before, users used a foreign set of data, and not their own personal information. The idea was to evaluate the system with users that had little or no knowledge about the information they were supposed
to understand and, therefore, evaluate the effectiveness of our prototype of expressing such information without relying on the user’s memory. Moreover, having all subject the exact same set of personal information makes possible the comparison of user’s performance during each task allowing a direct comparison of measures such as, for instance, the time spent to perform a specific task. Such statement is not always true when using user’s personal information. The information used was real information from a potential user of the system. Moreover, this data was also considered representative of what is expected from a Personal Information Dashboard’s typical user.

This section will briefly presents the methodologies used during the Usability Tests sessions and the profile of user that participated on it. We close this section by revealing the conclusions drawn from these tests.

4.1.1 Methodology

Before the actual tests took place, the users were introduced to the nature and intentions of the project and had time to make a quick “test-drive” to our solution. This “test-drive” had no pre-defined time limit, nevertheless, no user took more than 5 minutes to explore the system. This way we ensured a minimal familiarity with the system before the actual tests.

Next, the users performed a set of tasks using an pre-selected set of data. The actual test guide with the list of the task can be found in appendix B. The tests were composed by two groups of tasks within different contexts. The first group, the “Configuration Tasks”, contained tasks related to general configuration and contact’s management aspects (merging two contacts, creating a group of contacts, etc). Each one of the tasks had a specific goal:

- **Task 1** - Evaluate the configuration of an account: where and how to do it.
- **Task 2** - Evaluate the merging of two contacts belonging to the same person.
- **Task 3** - Evaluate the process of assigning a color to a contact.
- **Task 4** - Evaluate the creation of a new group.
- **Task 5** - Evaluate the management of a group by adding a contact to it.
- **Task 6** - Evaluate the process of assigning a color to a group. In fact, the process is almost the same as the one on Task 3, what can help us understand how easy is to learn and assimilate it.

The second group, the “Visualization Tasks”, was focused in tasks involving the usage and understanding of one or more plugins. Our intend was twofold: first, test each one of the implemented plugins (although not exhaustively); and second, test the several plugins related mechanisms and features we implemented. The first objective is important to validate each one of the plugins and assess if they meet the user’s expectations. The second objective is fundamental to check if is everything in place and if the system works well as a whole. Yet again, each task had a goal:

- **Tasks 1, 9, 17** - Evaluate the understanding of the visualization and the details-on-demand mechanism described in 3.7.3.
- **Task 2, 5, 10** - Evaluate the plugin’s configuration process.

- **Tasks 3, 12, 13, 16** - Evaluate the understanding of the visualization.

- **Tasks 4, 6, 8, 11, 14, 15, 18** - Evaluate the plugin intercommunication mechanism described in 3.7.4.

- **Task 7** - Evaluate the usage of two instances of the same plugin.

Notice the high number of tasks related to the intercommunication mechanism. This fact is no coincidence as we see this feature as one of the most interesting ones on our work and we really wanted to pay some attention to it. During the tests, three measures were recorded by an observer as part of the evaluation process:

- **Time Spent** - Time spent, in seconds, performing the task.

- **Number of errors** - Number of errors performed during each task. We considered as error any action that didn’t helped solving the task: a click in the wrong place, an incorrect or incomplete configuration, etc.

- **Number of Assists** - Number of times the user asked for help in order to complete a task. The users were aware of this possibility but were encouraged to only use such help as a last resource.

Notice that we did not considered task completion: at the limit, the user may ask several times for help until the task is complete. Finally, the observer was also responsible of recording any interesting comments made by the user or about the user performance during each task.

After completing all the tasks the users were asked to answer a satisfaction questionnaire that we used to measure, in a more methodical way, the general satisfaction among users. This questionnaire was divided in two parts. The first one was based on the System Usability Scale (SUS)[2], a standard usability set of questions that is associated with a method that assigns the system a grade according to user’s answers. The use of this mechanism allows us to have a better idea, more or less objective, of how is or system in terms of usability. The second part consisted in three essay questions where the user could express his/her opinion: a question to enumerate the main advantages, another to enumerate the main disadvantages and a final one to write some comments that the user felt could help improve our system. The English version of this questionnaire can be found at the end of this document as appendix D.

Finally, before the test session ended, a small informal discussion usually took place. During this discussions our aim was to get a more natural reaction to our prototype. We tried to get as much comments and suggestions as possible and discussed some of the choices made that the users felt that could be improved.
4.1.2 Participants

For our tests we tried to gather a group of people that could represent the potential users of our system. This includes people of a wide range of ages, from both genres, with different academic and professional backgrounds that use one or more of the sources of personal information that our system currently supports (remembering: Gmail, Facebook, Twitter and Flickr).

The Usability Tests were performed by a set of 16 potential users of the system. From the 16 subjects, 12 were male and the average age was 24.81 with a standard deviation of 8.13. Regarding the academic background, about 63% of the subjects have an academic degree (bachelor or master degree) within areas that included engineering, business, mathematics and health. All participants reported using a computer on a daily basis and at least one of the personal information sources our system currently supports: 14 users had a Gmail account, 10 users a Facebook account, 3 users a Twitter account and another 3 users a Flickr account.

4.1.3 Results

As depicted in 4.1.1, during the Usability Tests an observer recorded three distinct measures: the time spent to complete the task, the number of errors during the execution of the task and the number of assists required by the user to complete the task. These measures are an important part of our prototype’s evaluation as they can help us understand and find potentially problematic aspects of our implementation. Table 4.1 summarizes the recorded values. For analysis purposes, we present the average, standard deviation as well as maximum and minimum value for each one of these measures.

Generally speaking, we can observe that the number of errors and requests for assist were pleasantly low, and also more common in the early tasks. We believe that this means that the early errors are related to the fact that the user is still exploring and learning how to use Personal Information Dashboard. These values also suggest that the average user had no great difficulties in performing the requested tasks.

4.1.3.1 Configuration Tasks

Regarding the Configuration Tasks we found a curious behavior that was somehow expected: task 3 and 6 are very much similar (the first one requests the user to change the color associated to a contact, and the second to change the color associated to a group of contacts) but the recorded values are dramatically different. On average, task 3 took users 5 times more time to complete. In order to verify whether or not this difference is statistically significant, we applied a Student’s t-test. We obtained a P-value of $4.29 \times 10^{-11}$ which ultimately confirmed that the difference is statistically significant (for $p<0.05$). This difference among tasks so similar is explained by a non obvious, however easy, way to complete the task. To complete this task the user must follow a series of steps that are not obvious at first but are, as the numbers on table 4.1 seem to indicate, easy and fast to remember and reproduce. At the beginning of this paragraph we admitted that this behavior was already expected because we had already identified this feature on our solution. That was the main reason why we decided to have two tasks that are so similar: so we can test and show that the task is not difficult to perform, just not that obvious.
The rest of the recorded values for these tasks are within our expectations with low number of errors and requests for help. The occurrence of errors and requests for help don’t seem associated with any particular task or set of tasks (all tasks but task 6 had, at least, an error and/or request for help) and we noticed that some of these mistakes were due to user’s distraction.

---

**Tab. 4.1: Usability Tests results**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Time</th>
<th></th>
<th></th>
<th>Errors</th>
<th></th>
<th></th>
<th></th>
<th>Assists</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(\bar{x})</td>
<td>(\sigma)</td>
<td>Max</td>
<td>Min</td>
<td>(\bar{x})</td>
<td>(\sigma)</td>
<td>Max</td>
<td>Min</td>
<td>(\bar{x})</td>
</tr>
<tr>
<td><strong>Configuration Tasks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1</td>
<td>43.81</td>
<td>13.71</td>
<td>68.00</td>
<td>14.00</td>
<td>0.25</td>
<td>0.45</td>
<td>1.00</td>
<td>0.00</td>
<td>0.19</td>
<td>0.40</td>
</tr>
<tr>
<td>Task 2</td>
<td>46.31</td>
<td>11.73</td>
<td>65.00</td>
<td>15.00</td>
<td>0.19</td>
<td>0.54</td>
<td>2.00</td>
<td>0.00</td>
<td>0.13</td>
<td>0.34</td>
</tr>
<tr>
<td>Task 3</td>
<td>50.13</td>
<td>11.21</td>
<td>60.00</td>
<td>24.00</td>
<td>0.75</td>
<td>0.77</td>
<td>2.00</td>
<td>0.00</td>
<td>0.44</td>
<td>0.51</td>
</tr>
<tr>
<td>Task 4</td>
<td>14.13</td>
<td>2.96</td>
<td>20.00</td>
<td>10.00</td>
<td>0.06</td>
<td>0.25</td>
<td>1.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.25</td>
</tr>
<tr>
<td>Task 5</td>
<td>9.19</td>
<td>2.40</td>
<td>12.00</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Visualization Tasks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1</td>
<td>35.25</td>
<td>12.64</td>
<td>60.00</td>
<td>10.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 2</td>
<td>34.81</td>
<td>11.87</td>
<td>52.00</td>
<td>14.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.13</td>
<td>0.34</td>
</tr>
<tr>
<td>Task 3</td>
<td>18.13</td>
<td>9.82</td>
<td>48.00</td>
<td>10.00</td>
<td>0.13</td>
<td>0.34</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 4</td>
<td>19.44</td>
<td>9.28</td>
<td>48.00</td>
<td>10.00</td>
<td>0.13</td>
<td>0.34</td>
<td>1.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.25</td>
</tr>
<tr>
<td>Task 5</td>
<td>18.13</td>
<td>3.98</td>
<td>24.00</td>
<td>10.00</td>
<td>0.13</td>
<td>0.34</td>
<td>1.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.25</td>
</tr>
<tr>
<td>Task 6</td>
<td>28.50</td>
<td>9.41</td>
<td>49.00</td>
<td>17.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.13</td>
<td>0.34</td>
</tr>
<tr>
<td>Task 7</td>
<td>20.50</td>
<td>3.39</td>
<td>28.00</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 8</td>
<td>26.31</td>
<td>8.11</td>
<td>44.00</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.13</td>
<td>0.34</td>
</tr>
<tr>
<td>Task 9</td>
<td>24.31</td>
<td>8.27</td>
<td>40.00</td>
<td>12.00</td>
<td>0.06</td>
<td>0.25</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 10</td>
<td>7.00</td>
<td>3.08</td>
<td>15.00</td>
<td>3.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 11</td>
<td>21.94</td>
<td>5.73</td>
<td>34.00</td>
<td>15.00</td>
<td>0.06</td>
<td>0.25</td>
<td>1.00</td>
<td>0.00</td>
<td>0.13</td>
<td>0.34</td>
</tr>
<tr>
<td>Task 12</td>
<td>21.63</td>
<td>4.96</td>
<td>34.00</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 13</td>
<td>24.38</td>
<td>10.33</td>
<td>50.00</td>
<td>12.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 14</td>
<td>64.38</td>
<td>45.28</td>
<td>136.00</td>
<td>15.00</td>
<td>0.19</td>
<td>0.75</td>
<td>3.00</td>
<td>0.00</td>
<td>0.19</td>
<td>0.71</td>
</tr>
<tr>
<td>Task 15</td>
<td>25.50</td>
<td>21.62</td>
<td>76.00</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 16</td>
<td>36.06</td>
<td>8.64</td>
<td>80.00</td>
<td>20.00</td>
<td>0.13</td>
<td>0.34</td>
<td>2.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.25</td>
</tr>
<tr>
<td>Task 17</td>
<td>32.56</td>
<td>10.47</td>
<td>50.00</td>
<td>10.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Task 18</td>
<td>45.38</td>
<td>37.39</td>
<td>150.00</td>
<td>12.00</td>
<td>0.13</td>
<td>0.34</td>
<td>2.00</td>
<td>0.00</td>
<td>0.19</td>
<td>0.71</td>
</tr>
</tbody>
</table>

---

Still regarding the Configuration Tasks, as you can see in figure 4.1, the highest values occurred for the first three tasks, which can be partially explained by the fact that the user is just starting to get used to performing tasks on the system. This statement is particularly true for the first task, a task with a very low level of difficulty but that requires the user to find the appropriate menu in the interface. Also notice that this task has the highest standard deviation of the Configuration Tasks. Our interpretation is that it has to do with the “test-drive” users had before the actual tests: some users made a very complete exploration of the system, while other explored little or nothing. As a consequence, the users that really explored the system had already explored the intended menu/area of the system what explains why some users spent so little time performing the task and other the opposite. Moreover, notice that the trendline in
figure 4.1 indicates a decline in the average time spent performing general configuration tasks. Therefore, given the fact that the final tasks are not considered easier than the first ones, led us to conclude that the users quickly become proficient performing this class of tasks.

![Figure 4.1: Time spent performing the Configuration Tasks.](image)

**4.1.3.2 Details-on-demand Tasks**

In order to evaluate our details-on-demand mechanism we’ve performed three distinct tasks (task 1, 9 and 17). Regarding this set of tasks, users had more problems on tasks 1 and 17. Task 1 was the first task of this groups, which implies that the user was not yet familiar with the details-on-demand mechanism, what is a good explanation for a higher value. On task 17 users felt, in most cases, the need to confirm how the information was organized, which caused an increase in the mean time. Yet, the difference is not significant (for p<0.05), which leads us to conclude that users were able to use the details-on-demand mechanisms without major complications or difficulties. Task 9 seemed to be clearly the most immediate one (moreover, this task’s visualization is quite similar to the one on task 1), and the results have confirmed that.

**4.1.3.3 Visualization Tasks**

Regarding the Visualization Tasks, tasks 3, 12, 13 and 16 were specifically designed to test how easy was for the users to understand a set of visualizations that might not be that obvious at first. Nevertheless, keep in mind that understanding the visualizations was always necessary to successfully complete any task. The first three tasks were quite obvious and only required some attention by the user in order to understand what was on the screen. As a consequence, the results for these three tests are quite similar. On the other hand, tasks 16 took considerably more time, errors and assists to be completed. This difference can be explained by the fact that task 16 required the user to interact and look for the information: the information was not immediately available, the user had to look for it.
In order to study this group of tasks, we performed an ANOVA test and found the existence of statistically significant differences (for $p < 0.05$) among the group. Further Tukey’s tests allowed us to conclude that this difference only exists thanks to task 16: this task has significant differences (for $p < 0.05$) when compared to any other of the tasks in this group. Figure 4.2 can also help us graphically understand this difference: each axis correspond to the time spent by each user performing a tasks and each line maps that amount of time to a specific task. Looking at the resulting shapes, we can see that the shape of task 16 (T16, the red line) is a lot different from all the others. This result finally led us to the conclusion that a visualization that requires interaction may have a cost associated. An interactive visualization can hold much more information than a static one (since not all the information has to be exposed at the same time) but that might affect the usability of that visualization in terms of time needed to perform a task. This is an important fact to take into consideration at the moment of choosing the visualization technique to use. Nevertheless further tests, with more tasks involved, may be necessary to prove this theory.

![Figure 4.2: Radar like chart showing the time each user spent performing a subset of tasks.](image)

4.1.3.4 Plugins Configuration Tasks

The results for the plugin’s configuration tasks (task 2, 5 and 10) are quite good and we can see that the mean time spent performing each task is decreasing from task to task. This might indicate that users become proficient very quickly in this type of tasks: this fact is no shock, this class of tasks is considered easy. Again, the time is considerably higher on the first task as the users is still learning how to configure a plugin. In the first task the user is still feeling a little lost, not knowing exactly how to proceed. However, after the first task, all other tasks are performed relatively quickly, almost mechanically. Moreover, our observations denote no problem whatsoever associated with these kind of tasks, as indicates figure 4.3.

![Figure 4.3: Time spent performing the Plugins Configuration Tasks.](image)
4.1.3.5 Plugin Intercommunication Tasks

The plugin intercommunication mechanism, as an important part of our work, was also target of several tasks (namely, tasks 4, 6, 8, 11, 14, 15 and 18). Here, we can find relatively high standard deviations on the time spent, for the last three tasks. Moreover, we’ve performed an ANOVA test and the results indicate that the differences, regarding time spent, among these tasks are statistically significant (for $p < 0.05$). We’ve also performed Tukey tests in the pursuit of some more information, trying to isolate one or two tasks as the origin of the significance. These later studies revealed that none of the task is, by its own, the origin of the difference. However, we found that Task 14 and 18 are the “most different” ones: these tasks has statistically different results with all other.

We can also notice that the Task 14 and 18 were the tasks that took users more time to perform. This fact can be explained by the observations we made: on the last three tasks, many users did not used the intercommunication mechanism to solve the task. As as example consider task 14 that says: “Who sent you more emails in the last 14 days? Can you find out where that person lives?”. Finding out “who” was pretty straight forward just by using one of the plugins but finding where that person lives implied finding that person in another plugin (the “FriendsMap” plugin). Some subjects manually lookup for the person in the map, taking a lot of time. On the other hand, others used the plugin intercommunication and clicked on that person’s name in the first plugin to immediately show that person in the map of the “FriendsMap” plugin, finishing the task in much less time. Why users did not used the intercommunication mechanism is harder to understand and justify. After some sessions we started to noticed this pattern and took the liberty of asking some of the new testers why. The most common response was “I forgot to use it”. However, we are not completely convinced with this response/conclusion: this would be acceptable and normal for the initial tasks, not for the final ones when the user has already used the mechanism several times before. We tried to look deeper, looking for a more satisfactory answer, but we had no success in finding another conclusion.

4.2 Case Studies

Usability Tests can show us how real users use the system. However, always using the same personal information to perform the tasks has an obvious drawback: the information is not personal to the tester, he/she cannot see itself on the information which is one of objectives of our work. How users use our system to examine their own personal information is the main objective of the Case Studies.

The Case Studies were performed by only a small group of the participants of the Usability Tests: the two test sessions together can take a long time, time that most of the participants would not be willing to spend. Also note that the Case Studies were always carried out after the Usability Tests and such fact was no coincidence: this way the user is already more comfortable with the system and thereby make better use of it to exploit his/her personal information.

4.2.1 Methodology

As mentioned before, after the Usability Tests some of the users also performed the Case Studies. This tests consisted on a set of tasks somehow similar to tasks performed on the Usability Tests (the actual list of tasks can be examine in appendix C). However two related differences emerge: for the Case Studies the testers used their own personal information and no measure was
recorded. The idea was to see what happens when a real user uses his/her real information, what kind of patterns emerge, what's the reaction of such discovers and how different it is when comparing to the usage of unknown information (what ends up being a comparison with the Usability Tests described in 4.1). The absence of measures is explained by the problem raised by the usage of user's personal information: the personal information of two users can be arbitrarily different. The intensity of usage of different sources of information shapes the personal information and some tasks might even be, at the limit, impossible to complete in certain sets of information. Also, as in the case of the Usability Tests, each one of the tasks performed had its specific goal, namely:

- **Tasks 1, 9, 17** - Evaluate the understanding of the visualization and the details-on-demand mechanism described in 3.7.3.
- **Task 2, 5, 10** - Evaluate the plugin's configuration process.
- **Tasks 3, 12, 13, 16** - Evaluate the understanding of the visualization.
- **Tasks 4, 6, 8, 11, 14, 15, 18** - Evaluate the plugin intercommunication mechanism described in 3.7.4.
- **Task 7** - Evaluate the usage of two instances of the same plugin.

Last but not least, note that the users that performed the Case Studies only answered the satisfaction questionnaire after completing these tests, and not immediately after completing the Usability Tests.

### 4.2.2 Participants

Only 5 of the original group of 16 users performed these tests. Such option was due to two distinct facts: first, the Case Studies required the user to have a proper set of personal information items, i.e. a considerable usage of several sources of personal information; second, performing this tests dramatically increases the time spent to complete the evaluation process with each user. Therefore, we are convinced that applying the Case Studies to only 5 users is a good compromise which provides a good amount of information without compromising the evaluation process.

### 4.2.3 Results

The results for the Case Studies are much more complicate to analyze from an objective point of view as no measures were recorded. This means that the results expressed on this section will be much more subjective and the lack of objective proofs will also be a constant. Nevertheless, the main goal of this tests was ensured as we were able to observe how real users react to the usage of their own personal information.

The general reaction was a lot different from the one experienced on the Usability Tests. While performing tasks with a foreign personal information the users were clearly more interested in completing the tasks as soon as possible rather than in observing and exploring their findings. However, when using their personal information, users become much more interested and most of the times went further in the task experimenting other configurations, trying to understand...
something, looking for a pattern, etc. Part of this reaction may have to do with the fact that the users were aware that the time was not being recorded. However, we believe that most of this reaction is due to the fact that users were excited about using their own personal information.

In fact, we noticed that the users that participated on these tests, tended to give better grades on the satisfaction questionnaire (averages of 81.5 and 71.6 respectively). Again, to verify such fact we performed another Student’s t-test to compare the grades given by the users that performed the Cases Studies and the users that did not performed the Case Studies. The computed P-value was about 0.0086, which indicates that the difference between the grades given by these two groups is statistically significant (for $p < 0.05$). Our interpretation of this finding is that the users that used their own personal information felt much more satisfied using the system because they were not just solving tasks after tasks without looking and interpreting the results. Users were interested on the results, they wanted to know more and have a better understanding of what was happening. This is an interesting result since that, with these tests, we wanted to examine how users would react to using their own personal information.

During the tests users were able to find interesting things and we had some interesting situations. One user for instance, had just commemorated his birthday the day before. When he added the Who&How plugin he immediately noticed this fact on the visualization he just added. As this plugin shows user’s interaction with friends via Gmail, Facebook and Twitter, this user had a lot off small contributions via Facebook: these were the “happy birthday” posts from the day before. Figure 4.4 is a print screen of the resulting visualization, compare it with figure 3.16. Notice that the figure 4.4 has a lot more contacts represented, due to the abnormal amount of wall comments that a user receives in his/her birthday. Notice the pattern created in each case, notice how different they are and what that difference means. This is the kind of thing that Personal Information Dashboard aims to reveal.

Other interesting situations occurred with the FriendsMap plugin. Many users discovered something with this plugin: a friend that just recently moved to another country, someone that was from a distance hometown, etc. One particularly funny situation occurred when one of our users noticed that one of his friends had miss-configured its current living place. As a consequence of existing different locations with the same name, his friend had configured its location in Seixal (Mafra) when, in fact, he was living in Seixal (Montijo), two locations with the same name but separated by 70 kilometers. This situations highlights an important idea of our work: a visual representation of data is much more effective than a text one. Notice that the information provided by FriendsMap is easily accessible via the Facebook graphical interface but with two limitations: first, the locations are not shown on a map, they are just text (in fact, an hyperlink that can then open a map); and second, the user cannot see all his/her friends in just one map/place. These two limitations are surpassed by FriendsMap, resulting in an interesting visualization that the user can use to learn a lot about his/her friends. In both situations the information is there, but the way it’s shown makes all the difference.

Comparing their personal information with the personal information they used on the Usability Tests was also a common event and, after that, users really started to understand how different can the personal information of two individuals be. Also, the users seemed to understand that those differences can indicate patterns of interaction with others. Remember the situation of the user that had just commemorated his birthday: the visualization he saw about his personal information was quite different from the one he saw on the Usability Tests using a foreign personal information, result of two completely different situations. Users commented that.
As expected, some of the tasks could not be completed for some of the subjects: some users have no photo on Facebook which is a requirement for one of the tasks, etc. This verified fact is one more proof that using a pre-selected personal information set for the Usability Tests was, in fact, a good idea.

### 4.3 Users’ Satisfaction

Following the scoring method described in [2] we calculated the SUS score for each user and then obtain the average SUS score, it’s standard deviation, maximum value and minimum value. The results are displayed on Table 4.2.

The average SUS score of 74.84 indicates a fairly good usability degree perceived by the users during the tests and the small (6.09) standard deviation means that individual SUS’ scores are clustered closely around the mean which means that there’s no big discrepancy between user’s scores. The maximum and minimum value were included to allow a better understanding and analysis. The maximum value (85) is a very good score that occurred twice.
To better understand user’s satisfaction from a more per question point of view, we have elaborated a Box-and-Whisker Plot reproduced in figure 4.5. Our intent was to identify the most problematic aspects of our solution by examining the grades granted in each one of the questions from the first part of the survey. The usage of this representation can help us easily understand the distribution of the dataset and therefore, better understand user’s satisfaction.

By quickly examining figure 4.5 we can identify, at a glance, some questions that had similar “behavior”: question 2 presents a box very much similar to the question 10’s box, and the same happens for the questions 3 and 9 and the questions 4 and 8. If we look back to those questions we can easily understand why: these are questions purposely similar/complementary to each others. For instance, question 3 asks if the user considered the system easy to use and question 9 if the user felt confident while using the system: if the user considered the system easy to use is just natural that he/she felt confident while using it. This results helps us confirm the validity of the responses has they make sense as a whole.

In addition to the previous analysis, we also analyzed which aspects were, relatively, the most problematic ones. However, we must first remind the reader that on the SUS’ questions a greater grade is not always a better one. Question 2, for instance, is “I found the system
unnecessarily complex” and a 5 would indicate the maximum agree-ability with such statement which indicates a unnecessarily complex system. That said, and making a long story short, the reader just have to keep two things in mind: for even questions, greater is better; for odd questions, greater is worse. Another interesting pattern that emerges from this figure is the absence of the representation of some of the quartiles: some questions show no upper quartile (as in the case of Q1), some don’t show the lower quartile (as in the case of Q4) and others don’t show any quartile what so ever (as Q2 for instance). Such facts have to do with the usage of a 1 to 5 range, which is a small range. When no quartile is shown, this indicates that the elements of that quartile have the median value, and that’s why that quartile is not represented as no element reaches the next/previous axis’ value. When using broader scales, such situation becomes less likely to happen.

Examining the even questions (greater grade is better) we find that the worst average\(^1\) (the lowest average) occurred in question 7 (“I would imagine that most people would learn to use this system very quickly”) which might indicate the belief that some people may take some time to learn how to use the system. However, if we look at the results for question 3 (“I thought the system was easy to use”) we can conclude that the testers did not found the system hard to use: an average of 4 with 3 as minimum and 5 as maximum indicates that all testers graded 3 or more, meaning that all responses were neutral or positive for this question. Also, the absence of representation of any quartile for question 3 indicates that most of the responses were, in fact, 4 which is a pretty good grade for this question. Again, making a long story short, testers seem to be a little worried that some people might take some time to learn how to use the system but they seem not to suffer from that as they considered the system easy to use. For such apparent dichotomy we can think of two possible explanations: one, a bad judgment from the users regarding the difficulties that others may have; two, maybe the users that were more likely to have difficulties using the system were not included on the test group. In the worst case, we can only conclude that our test group could have included more users, probably elder ones that are recurrently associated to difficulties learning new things. However, finding elder users that fit the remaining criteria (specially, the criteria of using at least one of the sources of personal information) it’s a challenge and therefore we can generalize that they are not the typical users of our system. Last but not least, we can also see that question 7 had responses that ranged from 2 to 5, the widest range noticed which might just indicate a controversial question.

On the other hand, the best average for the even questions occurred in question 5 (“I found the various functions in this system were well integrated”) a reflection of the positive comments users made regarding the inter-plugin communication that we’ll elaborate in 4.4. Also notice that all responses ranged from 4 to 5, meaning that there’s no big discrepancy on user’s feelings about question 5.

For the odd numbered questions, the best average occurred on question 6 (“I thought there was too much inconsistency in this system”) what we think it’s a symptom of the careful design of all the parts of our prototype. Also notice that all the responses were between 1 and 2. As for the worst average we find out that no question jumps as evident worst and, in fact, all of them seem to have very acceptable grades.

---

\(^1\)Notice that the average for each question is represented by a dark gray diamond shape
4.4 Users Feedback

The user’s feedback was extremely positive. In fact, users seemed quite excited about what Personal Information Dashboard has to give. As positive aspects, most of the users mentioned the ability to have all their information in just one place. Some more advanced users also mentioned the ability to cross-referencing information from two or more sources making all the information more cohesive. The following quote\(^2\) is one aspect point mentioned by one of our users. As the reader can see, this quote brings together some of these aspects:

\[ I \text{ can obtain a “big picture” of the use of multiple platforms and contact with several people.} \]

The users also really liked the plugin intercommunication mechanism as a way of filtering and cross-referencing information among several plugins. Some were even truly amazed with this functionality and praised its potential. However, and as depicted before, users not always used this mechanism when they could. This observation might just indicate that its use is not always obvious and that we may need to do some refinement on this regard in order to rectify that. The following quote is from a user that considered this mechanism as one of the most positive aspects of our solution.

\[ \text{The communication between plugins. I can see how the information from different origins is connected. I can click on an email and see where that person lives. It’s pretty cool.} \]

The look and feel of the application was also target of very positive comments, although some people have in fact suggested the use of another color scheme. An user went further and suggested that our project could have a theme mechanism that would allow each user to have a color scheme of his/her choice. This idea came as no surprise for us, as we had already considered this hypothesis. However, this idea was left for second

Users seemed pleased with the sources of personal information that our project currently supports but a couple of then mentioned that, in addition to aggregate information from several sources, Personal Information Dashboard could also be a place to create information that would be propagated to the other sources:

\[ I \text{ see that I received an email from a person and I want to reply. It would be nice to reply directly from the dashboard without having to go to my email.} \]

Although we must admit that it is an interesting and tempting idea, the truth is that is, currently, a little out of the context of our project. Personal Information Dashboard is all about showing information differently, as a whole, revealing patterns and hidden aspect. Therefore, creating new information is not a top priority. We must however agree with the users: it would be great.

As negative aspects, some users referred the lack of contextual help for the plugins or for some of the options on the plugin’s configuration panel. We totally agree with this comment. On a final version this contextual help would need to be implemented.

\(^2\)Carefully translated from the original quote in Portuguese
One user also mentioned the fact that the fetching of information is sometimes slow, what might annoy more impatient users. This is only a problem because we are in the context of a one-session test. Since the user is using the system for the first time (with his/her credentials), all the necessary information has to be fetched at the moment, which may take some time. However, the next time the user is using the system, all the necessary information will already be in cache thus eliminating the need for fetching information again. Moreover, the system will also automatically refresh the information from time to time, thus ensuring an ever fresh information. However, this problem can again exist if the users adds a plugins that requires information not yet fetched. On this case, we are hand-tied since we only fetch the necessary information, not all the information available in all the sources (for obvious reasons).

### 4.5 Discussion

Overall, the test results indicate that Personal Information Dashboard is a solid step towards a greater comprehension of our personal information and, therefore, a greater comprehension of ourselves.

Our solution was able to join several sources of personal information in an effective way that the user is able to understand and see as a whole. The test results show that users were able to understand all the plugins and perform all the tasks without major problems. Although the possibility of asking for help, truth is that such help was rarely needed. In addition to that, the recorded errors and requests for help did not seem to be associated with any particular task or group of tasks. Such statements allow us to conclude that our system is user friendly and simple to use and learn.

The mechanisms for filtering and for details-on-demand were a success with users having no difficulties in using it. The plugin intercommunication mechanisms was one of the highlights of our solution since it allows a much deeper and cohesive exploration of our personal information. However, the results also show that the use of this mechanism is not always obvious: users do not always use this feature, though they know of its existence and recognize its value. The fact of being a very particular and unique mechanism may have something to do with it, however, we believe that such behavior will tend to fade away with time. Anyway, although not dramatic, this observation should be cause for future considerations based on the principle of continuous improvement.

The users had also no problems in the system’s configuration aspects. The configuration of the plugin’s instance is a painless process that the users learned to master really quick. The management of contacts and groups of contacts is also simple apart from one specific task that can be less obvious: assigning a color to a contact or group. Nevertheless, our tests also show that this task is simple to assimilate and reproduce.

The user’s feedback was extremely positive with users being really impressed with the potential of our project. While using their personal information, users really started to understand the kind of hidden things that Personal Information Dashboard can highlight. The fact they had performed the Usability Tests first (with a information that was not really personal) helped users understand that. As a consequence, we’ve received many ideas for improving our system.
Chapter 5

Conclusion and Future Work

The amount of information we produce about ourselves nowadays is huge and scattered over many different places. Therefore, it becomes very difficult to make an effective and efficient management of all that information without losing anything. In addition to that, the information is broken not functioning as a whole, making it difficult to make sense from it. The goal of Personal Information Dashboard is to use and combine several sources of personal information and reveal interesting facets and patterns of our lives giving an idea, at-a-glance, of what’s important.

On this dissertation we revealed what has been done in this area over the past years by describing some of these works. Many has been done in the field of personal information management and we believe that most of these works can be found in chapter 2. However, if we analyze each one of these works, none of them fully meet all the objectives of Personal Information Dashboard: show a cohesive vision of one’s personal information from multiple perspectives.

Personal Information Dashboard is a dashboard like application meant to run in any modern browser. The dashboard is populated with rich and interactive visualization of the user’s personal information that can be configured and personalized. Currently, the information is retrieved from five different sources of personal information (Gmail, Facebook, Twitter, Flickr and Panoramio). The interface tries to be simple yet attractive and a drag-and-drop mechanism allows users to organize the dashboard at their will. We’ve also implemented a details-on-demand mechanism based on tooltips as well as a filtering mechanism. The later one is particularly interesting since it allows the user to filter the information in all the visualizations in the dashboard, not just in one visualization. As a result, we can perform much more deeper and wider analysis of our information.

Or architecture is based on plugins and it was designed to easily allow the addition of new behaviors. New sources of personal information, new plugins and new filtering events are just some of the points where our solution can be extended.

Our tests showed that users had no major problems learning and using Personal Information Dashboard. The Usability Tests were important to detect potential problems (or room for improvement) and to get a sense of the system’s real effectiveness to transmit the information. The Case Studies were also fundamental to have a better idea of how the real user will use the system in the “real world”. These testes were also useful to see how the system reacts to different sets of personal information. The evaluation process we performed also indicates that the users really liked to use our prototype. In fact, many users were quite excited about our project’s potential which led to a series of ideas of how the system could be improved.
5.1 Future Work

Speaking of improvements, in terms of future work we have already detected some interesting starting points. A theme system would be perfect to make the experience even more personal as it would allow each user to personalize its dashboard to a whole new level. Our prototype is already partly ready for this idea, it’s however necessary to make some adaptations.

Another idea is having full-screen visualizations for the plugins. The idea is simple: a new button in the plugin’s header would make the plugin’s visualization grow to fill the entire screen. At that time, the visualization itself would be adapted to show new things that were previously hidden for space reasons. Also notice that with more space new iterations would be possible. Imagine for instance applying this idea to the Spark Stats plugin: on the full-screen version the user would be able to, for instance, go down to the day level and see exactly which emails were sent at a specific point in time and, why not, see the threads of emails exchanged over the months.

The merging of contacts can also be improved by adding a semi-automatic approach. The rational behind this idea is the observation that contacts belonging to the same person usually have similar names or usernames. With this in mind, it should be possible to create an algorithm to detect the similarity between a pair of names/usernames. Then, the pairs with high level of similarity would be presented to the user that would ultimately decide whether or not those two contacts belong to the same person.

Also the current version of the plugin catalog must be improved. Currently the catalog is simply a list of available plugins but with the addition of more plugins, a catalog like this become very inefficient and, therefore, we should look for alternatives. Searching for terms or tags would be the very first and obvious approach that would really help minimizing the problem. A search by used sources would also be welcome as the user might just want plugins that use a small set of personal information sources. Finally, it would also be interesting to have a plugin’s recommendation system based on several aspects such as: age, genre, location, the plugins the user already has, the plugins his/her friends have, etc.

Last but not least, we would also like to support new sources of personal information in the future as well as implementing new plugins and types of intercommunication events. The architecture is prepared to be easily extended at these points and we must take advantage of that fact.
Appendix

A  Extensibility

Extensibility is a fundamental aspect for our work reflected in the R9 (The system must be easily extensible ie, it should be easy to create and add new visualizations, sources of personal information, etc.) requirement. Therefore, we’ve dedicate some of our effort to create a mechanism that didn’t required any manual configuration.

Our system’s extensibility is based on a set of super-classes that define the necessary work-flow with the rest of the application. To add something new it’s only necessary to create a sub-class of the intended class and place it in the correct directory. The system will detect that class and automatically import it with no configuration process in between.

In the next pages of this section we have some tutorials for some of the extensible points of our solution. These tutorials are the first step of the process, our advise is that you should first read this tutorials and then inspect the source code.
A1 Supporting a new source of personal information

To add a new source you need to create a sub-class of the class “SourceManager” and put that file in a folder. Next, you shall put the created folder on the following directory: “src/services/-sources”.

Along this section we’ll guide the reader through the creation of a simple manager for RSS feeds. We’ll make it step by step explaining each one of the steps.

1) First, we must create the file “RSSManager.py” and reproduce the code below.

```
from services.sources.sourceManager import SourceManager

RSS_SOURCE_ID = 'rss'
RSS_SOURCE_COLOR = 'rgb(250, 157, 57)'

class RSSManager(SourceManager):
    _source_id = RSS_SOURCE_ID
```

As you can see, we’ve created a sub-class of SourceManager called RSSManager. The _source_id property is the most important thing here: this is the string that will identify the source of personal information associated to this manager. Use the source’s name to minimize the chances of name collision. The RSS_SOURCE_COLOR is also important if you want to associate a color to the source for visualization purposes.

2) Next, we need to fetch the information from the source. To do that, we must first create a suitable method that will be used by the plugins to get the information. On this case, we’ll create a method that will receive a RSS link as a parameter.

```
def get_feed(self, caller, rss_link):
    # TODO
```

As usually for a python method, the first argument is the self. The second argument for this methods should always be the caller. This argument holds a reference to the object that called this method (most likely, the plugin that called the method to obtain the information) and we’ll explain its importance further in this tutorial. The next arguments should be any arguments you find suitable to your needs. On our case, we have the rss_link that holds the link for the desired rss feed.

3) Now we’ll actually return something from the source. However, an important detail arises: the caller cannot wait until you have fetched the link, which can take several seconds. The solution is having an asynchronous behavior with a cache that should work like this:

- If you already have the link fetched, you should return whatever is fetched.
- If you don’t have the link fetched, you should start fetching the link.

The next piece of code explains how you could do it.

```
class RSSAccount(SourceAccount): pass

class Request():
    def __init__(self, rss_link):
```
self.rss_link = rss_link

class RSSManager(SourceManager):
    _source_id = RSS_SOURCE_ID
    def __init__(self, account_token, processors = []):
        super(RSSManager,self).__init__( RSSAccount(account_token), processors)
        self._cache = {}

def get_feed(self, caller, rss_link):
    if rss_link in self._cache:
        return self._cache[rss_link]
    else:
        self._add_request(rss_link)
        return SOURCE_MANAGER_NOTHING_YET

def _add_request(self, rss_link):
    self._pending_requests.append(Request(box, state, since, until))
    self._cache[rss_link] = SOURCE_MANAGER_NOTHING_YET

Notice that we have created the cache property in the __init__ method, a map that associates a link to a rss feed. The get_feeds checks for the cache to see if the link is already fetched and, if not, uses the _add_request method to add the request to the _pending_request list (a list from the super-class).

4) The request is added, but is not actually processed. To process the request you should implement the _dispatch_request method that should dispatch the method to the entity that will actually process it.

def _dispatch_requests(self):
    while True:
        request = self._get_next_dispatchable_request()

        if request is None:
            time.sleep(2)
            continue
        request = Request()
        rss_link = request.rss_link
        RSSFetcher(request)

In turn, the RSSFetcher would be something like this.

class RSSFetcher:
    def __init__(self, manager, request):
        self.__manager = manager
        self.__rss_link = request.rss_link

        th = Thread(target = self.__process_request)
        th.start()

    def __process_request(self):
        feed = RSS.fetch(self.__rss_link) # TODO
        self.__manager.add_feed(self.__rss_link, feed)
5) Our class is almost done, but first we must support the update of information. There are many ways of doing this but our suggestion is that you should store the time of the next update and only update the information when that the information is requested after that time (and not only when that time has passed). Consider the following class

class Info():
    def __init__(self, next_update):
        self.next_update = next_update
        self.feed = SOURCE_MANAGER NOTHING_YET

Instead of putting the feed directly on the _cache, we store an object of type Info that will hold the feed and the next update time. This requires an adaptation of the get_feed, add_request and add_feed methods. We’ll also define the UPDATE_INTERVAL.

def get_feed(self, caller, rss_link):
    try:
        rss_info = self._cache[rss_link]
    except KeyError:
        rss_info = None
        now = time.time()

        if rss_info is None or rss_info.next_update >= now:
            self._add_request(rss_link)
            return self._cache[rss_link].feed
        else:
            return self._cache[rss_link].feed

def _add_request(self, rss_link):
    self._pending_requests.append(Request(box, state, since, until))
    if not rss_link in self._cache:
        self._cache[rss_link] = Info(rss_link)

def add_feed(self, rss_link, feed):
    rss_info = self._cache[rss_link]
    rss_info.feed = feed
    rss_info.next_update = time.time() + UPDATE_INTERVAL

6) Finally, there’s only one thing left: persistence of data. The easiest way of supporting this is using the close method. This method is automatically called when the program exists which works just like a charm for prototyping purposes. Also, do not forget to create the _persistence_dir variable at the class level: this variable should hold the name of the folder were the information will be stored.

    _persistence_dir = "rssSource"

(...)

def close(self):
    super(RSSManager, self).close()
    storage.store(self._cache, "cache", self._persistence_dir, active = True)
A2 Supporting new contact and group sources

Supporting the fetching for contacts and groups from new sources is an important feature in order to completely support a source of personal information. In order do that you need to create a sub-class of the class “SourceContactsFetcher” and a sub-class of “SourceGroupssFetcher” and put those files in that source’s folder.

1) The lines above show a typical contact’s fetcher.

```python
class SourceAContactsFetcher(SourceContactsFetcher):
    def _fetch_contacts(self):
        #TODO
    def _compute_contacts_list(self):
        return self._fetch_contacts()
    def get_source_id(self):
        return SOURCEA_SOURCE_ID
```

The _compute_contacts_list method should return the list of contacts for the source with ID equal to the one specified by the get_source_id. This is enough, all the rest of the work will be source specific and, therefore, out of scope.

2) The code for fetching groups is pretty much the same, with obvious modifications:

```python
class SourceAGroupssFetcher(SourceGroupsFetcher):
    def _fetch_groups(self):
        #TODO
    def _compute_groups_list(self):
        return self._fetch_groups()
    def get_source_id(self):
        return SOURCEA_SOURCE_ID
```

Again, the get_source_id method should specify the source’s ID and the _compute_groups_list the list of groups for that source.
Implementing a new plugin

To create a new plugin you need to create a sub-class of the class "Plugin" and put that file in a folder. Next, you shall put the created folder on the following directory: "src/plugins".

In this section we’ll create a very simple plugin that only show the RSS feeds in a textual form. More advanced visualization are completely out of scope of this section.

1) First, we must create the file “MyRSSFeeds.py” and reproduce the code below.

```python
class MyRSSFeeds(PlugIn):
    plugin_name = "My RSS Feeds"
    plugin_author = "me"
    plugin_description = "Use this plugin to read your feeds."

    plugin_img_path = "/plugins/MyRSSFeeds/image.jpg"

def __init__(self):
    #TODO

def is_configured(self):
    #TODO

def get_used_sources(self):
    #TODO

def print_configure_menu(self):
    #TODO

def _configure(self, field_storage):
    #TODO

def _compute(self):
    #TODO

def _display(self):
    #TODO

def get_resize_function(self):
    #TODO
```

This is the base skeleton for any plugin. Notice the variables on the first lines of the class. The first holds the plugin name, the second the name of the author, the third the description that will be placed on the catalog and the fourth the path for the plugins image, that will also show on the catalog. On the following pages we’ll briefly describe and implement each one of the method of this class.

2) First things first, we must first implement the __init__ method. Here we must first call the method with the same name from the superclass and then initialize any useful variable.

```python
def __init__(self):
    PlugIn.__init__(self)
```
The variable _width and _height are inheritance from the superclass and define the plugin’s width and height on the dashboard. If no value is specified, the system will try to assign a size based on the content. The rest of the variables are specific for this plugin: the _rss_links will hold all the link to be fetched and _rss_feeds the fetched text for each link.

3) The system needs to know what sources of information is each plugin using and if the plugin is already configured. In order to know that you should implement the following two methods:

```python
def is_configured(self):
    return RSS_SOURCE_ID in self.OAuth_tokens

def get_used_sources(self):
    return [RSS_SOURCE_ID ]
```

4) The next step are the configuration related methods. See the code bellow:

```python
def print_configure_menu(self):
    links = ';' .join(self._rss_links)][0]
    rss_list = self.get_default_text_input_layout("RSS Links", "links", links, "large")
    return super(MyRSSFeeds, self).print_configure_menu([
        ("Options",
         rss_list)
    ])

def _configure(self, field_storage):
    links = field_storage["links"]
    self._rss_links = links.split(";")
```

The print configure menu is the method that will generate the configuration menu. The Plugin superclass has a method with the same that can be used in order to speed this process. This method receives a list of tuples. Each tuples is a section of the configuration menu. The first element of this tuple should be the section name and the following elements the fields for that section. Note here we use the get_default_text_input_layout method to create a text input. Other similar methods are available for other kinds of input fields. The _configure method is called when the user configures the plugin. Notice that this method receives field_storage, a map with the values for the field defined in the previous method. Keep in mind that the field’s id must match in order to retrieve the correct value.

5) Next we have the _compute method. This method is responsible by getting all the needed information in place and processed.

```python
def _compute(self):
    manager = self.get_source_manager(RSS_SOURCE_ID)
    for link in self._rss_links:
```
feed = manager.get_feed(this, link)
self.__rss_feeds[link] = feed

For this plugin, we just need to retrieve the information from the respective manager but keep in mind that any information processing must occur here.

6) Finally, we have the _display function, the function that generates the plugin's visualization. As depicted before, this plugin will have a simple visualization. Technically speaking, this method should return a string containing the html/javascript code containing the intended visualization.

```python
def _display(self):
    if len(self.__rss_feeds) == 0:
        return self.info_not_yet_available()

    to_return = ''
    for link in self.__rss_feeds:
        feed_text = self.__rss_feeds[link]
        to_return += "<div class='%s'>%s</div>" % ('feed' + self.id, feed_text)

    to_return += "<script type='text/javascript+protovis'>"
    to_return += "function %s(bus) { /* nothing*/ }" % self.get_intercommunication_function_name()
    to_return += "</script>"

    return to_return
```

Nothing too fancy here, just divs with text. Notice however the last lines where we define the intercommunication function: the get_intercommunication_function_name function should always be used in order to get the correct name for the intercommunication function. On this case, we do nothing in case of a filtering event.

7) Last but not least we should implement the get_resize_function that will return the javascript code responsible by processing resize events.

```python
def get_resize_function(self):
    return ""
```

Again, we do nothing in case of resize. Notice however that if we did no implement this method, the plugin would not resize at all. As it is, the plugin will resize and the content adapt to the new size.

Notice that much much was left to explain about the plugins. Explaining everything in a tutorial would be impractical, so we chose to address only the most important issues. For more details, please see the code and the already implemented plugins.
A4 Implementing a new merger

A merger is the entity responsible by automatically merging contacts from different sources of personal information. To create a new merger you need to create a sub-class of the class “BaseMerger” and put that file in the following directory: “src/services/sources/contactsMergers”.

Here we’ll create a dummy merger, just to explain how it should be done.

1) The first step is to create the new file and reproduce the following code

   ```python
   class SourceASourceBMerger(BaseMerger):
       _targets = (SOURCEA_SOURCE_ID, SOURCEB_SOURCE_ID)

       def merge(self, contacts, new_contacts):
           #TODO
   ```

   Notice the _targets variable, this variable should hold the ID from the intended sources. Is based on this variable that the merger will be called to act.

2) The merge method is where the all magic happens. The idea is simple: the method receives a list of the current contacts and the list of the new contacts to add. Base o those lists, the merger should use some king of merging criteria and return the id of the contacts to merge. For more detailed information about to actually implement this, we suggest that you should inspect the code as it is, in most cases, too long to put here.
A5 Implementing a new processor

A processor is an entity with the ability to process each atomic piece of retrieved information: an email, a post, etc. However, each processor is suppose to only deal with information from one source. It’s a mechanism that can be used for virtually anything. To add a new processor you need to create a sub-class of the class “SourcePieceProcessor”. Typically, this subclass is placed on the same file as the “SourceManager” for that source.

As an example, we’ll implement a processor that splits the text from a RSSFeed object into words and stores those words on the piece for future use.

1) The first step is to reproduce the code bellow:

```python
class RSSPFeedTextProcessor(SourcePieceProcessor):
    _source_id = 'rss'

    def _process_piece(self, piece):
        # TODO
```

On this step we defined the skeleton for the processor. Notice the _source_id variable, this variable should be equal to the intended source id.

2) For the next step we’ll just need to implement the _process_piece method. This method is called for each piece that arrives (in this case, for each RSS feed). So, in other words, this is where everything happens. In this case, we use some already implemented method to extract word from a piece of text and store the results in the piece itself

```python
class RSSPFeedTextProcessor(SourcePieceProcessor):
    _source_id = 'rss'

    def _process_piece(self, piece):
        bag_of_words = utils.extract_words(piece)
        piece.words = bag_of_words
```
Implementing a new intercommunication feed

The plugion intercommunication in a dashboard level filtering mechanism that filters all the visualization into the same state. The implementation of a new feed is simple but requires some discipline. In fact, implementing a new feed or supporting an already existing feed is pretty much the same: the filtering event is propagated (no matter its type) to the rest of the dashboard and is up to each plugin to adjust to the new filtering event.

In this tutorial, we’ll reproduce the implementation of the words feed.

1) For a plugin to support a (new) feed type, its _display method needs to be changed. First the plugin must know how to trigger that event.

```python
def _display(self):
    to_return = ""

    # put here some other code
    
    to_return += """function wordClick%s(wordObject) {
    var words = [wordObject.words];
    var feed = [ ['words' , words]]; 

    propagate_PID(%s, feed);
} """ % (self.id,self.id)

    return to_return
```

Here we suppose that there’s already a wordClick%s function that is called when a word on this plugin is clicked. How that click event is detected is out of this tutorial’s scope. Notice however that the final name of this function is concatenation of “wordClick” and the plugin’s id. The appending of the plugin’s id is important to ensure that the correct function is called (as other plugins may define a function with the same name). Using the plugin’s id at the end of each function, avoids this.

Notice how the feed variable is created: it’s an array of arrays. Each inner array holds the information from one type of field. The first position of these inner arrays is the feeds type (“words”, in this case) and the second position the necessary information to understand the corresponding filtering event (an array of words, in this case).

Finally, the plugin must call the propagate_PID function that will propagate the event to the bus. The first argument for this function is the plugin’s id and the second one the created feed object.

2) After propagating the event, the plugin must know how to handle that event. For that, implement the intercommunication function in the plugin’s _display.

```python
def _display(self):
    to_return = ""

    # put here some other code
    
    to_return += """function wordClick%s(wordObject) {
    var words = [wordObject.words];
    var feed = [ ['words' , words]]; 

```
propagate_PID(%s, feed);
} """ % (self.id,self.id)

to_return += """function%s(bus) {
    if(bus.isEmpty()) {
        # nothing selected
    } else {
        # some words selected
    }

} """ % (self.get_intercommunication_function_name(),self.id)

return to_return

The concrete implementation of this function is plugin dependent so we wont go further on this regard. Just keep in mind the existing of the isEmpty function, a useful bus function that returns true if nothing is selected. For more details, we suggest the examination of the code of the already implemented plugins.
First of all, we appreciate your willingness to participate in the test you are about to perform. Your participation is very valuable because it will help us test and improve our system. Also, please keep in mind that the purpose of this test is to evaluate our system and not to evaluate you.

During the test you’ll perform two sets of tasks that will use pre-selected personal information that is not yours. The first set of tasks, the “Configuration Tasks” are tasks that will help us evaluate the graphical interface of our system regarding the implementation of some of some basic configuration tasks. The second set of tasks, the “Visualization Tasks”, will test the way that the information is displayed and interconnected in the various available plugins. The idea of this test is to determine how effective the system is to transmit information that is unknown to you.

Please note the following aspect:

- Possible responses to the tasks must be written on a separate sheet.
- There’s no time limit to complete the tasks. Take your time.
- If you have a question that does not allow you to continue, you can ask for our the help. However, we ask you to only use such help as a last resort.
- The time spent, number of errors and the number of times you ask for help during the execution of each task will be recorded in this test as a measure of success for the system under evaluation.
- During the test, the observer can take notes that will complement the test results.
Configuration Tasks

1) Configure the Twitter Account.

2) Merge the two contacts with the name “Daniel Gonçalves”.

3) Change the resulting contact’s color to a light blue.

4) Create a group named “Teste”.

5) Add the contact named “Daniel Gonçalves” as an element of the group you just created.

6) Change the group “Teste”’s color to a dark green.

Visualization Tasks

Add the plugin “You’ve got bubbles”.
Go to your dashboard.

1) How many unread emails do you have on your inbox? Who sent those emails?

2) Configure the “You’ve got bubbles” in order to display all the emails (i.e. read and unread) with the label “Tese” received in the last 7 days. How many emails were found?

Add the plugin “Keywords Cloud”.
Go to your dashboard.
Wait until the “Keywords Cloud” has all the information.

3) Identify the top 3 more important words for the last 7 days.

4) Can any of those words be found on the label “Tese”?

5) Configure the “Keywords Cloud” in order to display the keywords for your email account. Can any of those words be found on the emails received, last week, for the label “Tese”?

Remove the plugin “Keywords Cloud”. Add the plugin “My E-emotions”.
Go to your dashboard.
Wait until the “My E-emotions” has all the information.

6) Identify the emotions found for the selected period of time. Can you find any of those emotions on the emails labeled with “Tese”?

Add another instance of “My E-emotions”.
Go to your dashboard.

7) Configure the first instance in order to only display Gmails data and the second instance to only display Facebook data. Can you notice any difference?
Delete both instance of “My E-emotions”.
Configure the plugin “You’ve got bubbles” to display emails labeled with “Receitas”.
Add the plugin “Photo Search”.
Go to your dashboard.
Configure the plugin “Photo Search” to use the tags “comida” and “portuguesa”. Wait some seconds while the plugin fetches the photos.

8) Click on the larger email on the label “Receitas”. Did you notice any change on the “Photo Search”? 

Delete the plugin “Photo Search”.
Add the plugins “My Feeds”, “Who and How” and “SparkStats”. 
Go to your dashboard.

9) Did someone sent you an email, posted on facebook or made a tweet for the last 6 hours?

10) And for the last 24 hours?

11) Identify who made more tweets on the last 24 hours. How many emails did you received from that person in the last 7 days?

12) Who sent you more emails in the last 7 days?

13) How many emails did you sent and received in the last 7 days?

Delete the plugins “My Feeds”, “You’ve got bubbles” and “Spark Stats”. Add the plugins “Friends Map” and “Stacked Memories”.
Go to your dashboard.

14) Who sent you more emails in the last 14 days? Where that person lives?

15) Can you find any photo with that person?

16) Is any of your contacts currently living on Panamá? Who is that person? What is his/her hometown?

Delete the plugins “Friends Map”, “Stacked Memories” and “SparkStats”. Add the plugin “Our Favourite Stuff”.
Go to your dashboard.
17) What are the three top things your contacts like the most?

18) Consider the contacts that sent you an email in the past 14 days. From those contacts, who likes one or more of the three things you answered on the previous task?
First of all, we appreciate your willingness to participate in the test you are about to perform. Your participation is very valuable because it will help us test and improve our system. Also, please keep in mind that the purpose of this test is to evaluate our system and not to evaluate you.

During the test you'll perform two sets of tasks that will use your own personal information. The tasks were carefully chosen, nevertheless, if you find one task that cannot be totally or partially executed, you should explain us why it is so and continue to the next task.

Please note the following aspect:

- Possible responses to the tasks must be written on a separate sheet.
- There’s no time limit to complete the tasks. Take your time.
- If you have a question that does not allow you to continue, you can ask for our the help. However, we ask you to only use such help as a last resort.
- Aspects as the time spent, number of errors, etc will not be recorded in this test.
- During the test, the observer can take notes that will complement the test results.

Last but not least, we would also like to ask you to follow a loud voice protocol. Such protocol consists on explaining out loud what is happening during the tests, what you are doing, what you find out, questions might you have, etc. Don’t be shy and share everything even if you thing that it doesn’t matter. We’ll be the judge of what matters.
Add the plugin “You’ve got bubbles”.
Go to your dashboard.

1) Do you have any unread email on your inbox? If so, can you identify the senders of some of those emails?

2) Configure the “You’ve got bubbles” in order to display all the emails (i.e. read and unread) with the label “[PID]/Received” received in the last 7 days.

Add the plugin “Keywords Cloud”.
Go to your dashboard.
Wait 1 minute while the “Keywords Cloud” gets some information.

3) Identify up to 4 of the most important words for the last 7 days. Do you recognize some of those words?

4) Can any of those words be found on the label “[PID]/Received”?

5) Configure the “Keywords Cloud” in order to display the keywords for your email account. Can any of those words be found on the emails received, last week, for the label “[PID]/Received”?

Remove the plugin “Keywords Cloud”. Add the plugin “My E-emotions”.
Go to your dashboard.
Wait 1 minute while the “My E-emotions” gets some information.

6) Identify the emotions found for the selected period of time. If no emotions were found, please configure the plugin in order to show a larger interval of time. Can you find any of those emotions on the emails labeled with “[PID]/Received”?

Add another instance of “My E-emotions”.
Go to your dashboard.

7) Configure the first instance in order to only display Gmails data and the second instance to only display Facebook data. Can you notice any difference?

Delete both instances of “My E-emotions”.
Add the plugin “Photo Search”.
Go to your dashboard.
Configure the plugin “Photo Search” to use two word/tags of your choice. Wait some seconds while the plug-in fetches the photos.

8) Click on some of the emails on the larger email on the “You’ve got bubbles”. Did you notice any change on the “Photo Search”? Can you understand why?

Delete the plugin “Photo Search”.
Add the plugins “My Feeds”, “Who and How” and “SparkStats”.
Go to your dashboard.
9) Did someone sent you an email, posted on facebook or made a tweet for the last 6 hours?

10) And for the last 24 hours?

11) Identify who made more tweets (or, alternatively facebook posts) in the last 24 hours (you can increase this value if necessary). How many emails did you received from that person in the last 7 days?

12) Who sent you more emails in the last 7 days?

13) How many emails did you sent and received in the last 7 days?

Delete the plugins “My Feeds”, “You’ve got bubbles” and “Spark Stats”. Add the plugins “Friends Map” and “Stacked Memories”.
Go to your dashboard.

14) Who sent you more emails in the last 14 days? Can you find out where that person lives?

15) Can you find any photo with that person?

16) Where, geographically, are located most of your contacts? Do you have any contact living in another country?

Delete the plugins “Friends Map”, “Stacked Memories” and “SparkStats”. Add the plugin “Our Favourite Stuff”.
Go to your dashboard.

17) What are the three top things your contacts like the most?

18) Consider the contacts that sent you an email in the past 14 days. From those contacts, who likes one or more of the three things you answered on the previous task?

19) You can now use the remaining time to freely explore our system. You can add, remove, configure, etc plugins at your wish. Just don’t forget to keep the loud voice protocol.
D Satisfaction Questionnaire

Personal Information Dashboard

Satisfaction Questionnaire

Once again, we thank you so much for participating on this test. To end this session, we would like you to answer the following satisfaction questionnaire that will help us understand and get more information of what can we improve on our prototype.

1. In the next 10 statements answered, by marking with a cross, how much do you agree with it on a scale from 1 (Strongly Disagree) to 5 (Strongly agree).

1.1. I think that I would like to use this system frequently

Strongly disagree  |  Strongly agree
1  2  3  4  5

1.2. I found the system unnecessarily complex

Strongly disagree  |  Strongly agree
1  2  3  4  5

1.3. I thought the system was easy to use

Strongly disagree  |  Strongly agree
1  2  3  4  5

1.4. I think that I would need the support of a technical person to be able to use this system

Strongly disagree  |  Strongly agree
1  2  3  4  5
1.5. I found the various functions in this system were well integrated

Strongly disagree                      Strongly agree
1  2  3  4  5

1.6. I thought there was too much inconsistency in this system

Strongly disagree                      Strongly agree
1  2  3  4  5

1.7. I would imagine that most people would learn to use this system very quickly

Strongly disagree                      Strongly agree
1  2  3  4  5

1.8. I found the system very cumbersome to use

Strongly disagree                      Strongly agree
1  2  3  4  5

1.9. I felt very confident using the system

Strongly disagree                      Strongly agree
1  2  3  4  5

1.10. I needed to learn a lot of things before I could get going with this system

Strongly disagree                      Strongly agree
1  2  3  4  5
2. List those that you think are the greatest advantages of the system.

3. List those you consider to be the major drawback of the system.

4. Write down any comments or suggestions to help us improve our system.
Bibliography


