Analysis and improvement of a publication system

Angel Rodriguez Fernandez
Abstract

The aim of this thesis is the research in modern web development tools to demonstrate in which ways can this raised technologies help to make a real application more user friendly, easy to maintain and extend, and more reliable.

Some new approaches to Web applications will be presented in comparison with the traditional way, and a real application will be redesigned for including the lasts advances in Web technologies.

Key words

Web application, Web development, Publication system, AJAX, Google Web Toolkit, Java.
Index

1. Introduction........................................................................................................... 1
   1.1. Structure of the report ......................................................................................... 1
   1.2. Content of the CD ............................................................................................. 1
2. Background ........................................................................................................... 2
   2.1. Web applications ............................................................................................... 2
   2.2. Problem with traditional Web applications ...................................................... 2
       2.2.1. Unresponsive browser ............................................................................... 3
       2.2.2. Problem with client side programming ..................................................... 3
       2.2.3. Overuse of server side programming ......................................................... 3
       2.2.4. Designs with dozens of small images ......................................................... 4
   2.3. Webs with asynchronous calls ........................................................................... 4
   2.4. DIVA .................................................................................................................. 6
3. Analysis ............................................................................................................... 8
   3.1. RICS ................................................................................................................... 8
   3.2. User requirements ............................................................................................. 9
   3.3. System requirements ....................................................................................... 10
       3.3.1. Functional requirements ......................................................................... 10
       3.3.2. Non-functional requirements ................................................................... 11
       3.3.3. Domain requirements .............................................................................. 11
4. Design ............................................................................................................... 12
   4.1. Concepts .......................................................................................................... 12
       4.1.1. Entry point ................................................................................................. 12
       4.1.2. Widgets ...................................................................................................... 12
       4.1.3. Sections ...................................................................................................... 12
       4.1.4. Server side and Client side ....................................................................... 12
   4.2. Layout and user interface ................................................................................. 12
   4.3. High level design ............................................................................................. 13
   4.4. Sections design ............................................................................................... 14
   4.5. Navigation system ........................................................................................... 16
   4.6. Services .......................................................................................................... 17
   4.7. Form system .................................................................................................... 18
5. Implementation ................................................................................................. 22
   5.1. GWT - Google Web Toolkit .............................................................................. 22
       5.1.1. Java oriented in client and server side ....................................................... 22
       5.1.2. Programming and debugging ................................................................. 22
       5.1.3. JavaScript works in all browsers ............................................................... 22
       5.1.4. Asynchronous calls ................................................................................... 22
       5.1.5. Light servers ............................................................................................... 23
       5.1.6. Image bundles ............................................................................................. 23
       5.1.7. Support for internationalization ............................................................... 23
       5.1.1. Third parties libraries ............................................................................... 23
   5.2. Database management ...................................................................................... 23
   5.3. Implementation in GWT .................................................................................... 24
       5.3.1. Basic GWT elements ................................................................................ 24
       5.3.2. Using GWT in the design ......................................................................... 26
   5.4. Implementation results ..................................................................................... 27
6. Maintenance ...................................................................................................... 29
6.1. GWT basic application ................................................................................. 29
6.2. Configuring the basics ............................................................................. 29
   6.2.1. GWT configuration file .................................................................... 29
   6.2.2. Server configuration file ................................................................. 30
   6.2.3. Application configuration file ......................................................... 30
6.3. Modifying the layout .............................................................................. 30
6.4. Editing the style .................................................................................... 30
6.5. Adding a new section ........................................................................... 31
   6.5.1. Understanding MenuPanel .............................................................. 31
   6.5.2. How create new page ..................................................................... 31
   6.5.3. How to create a new section .......................................................... 32
6.6. Different kinds of sections .................................................................. 32
   6.6.1. SectionAbstract ............................................................................. 32
   6.6.2. SectionLeaf .................................................................................. 33
   6.6.3. Section Decorator ......................................................................... 33
   6.6.4. SectionLinkSymbolic .................................................................... 34
   6.6.5. SimpleComposite .......................................................................... 34
   6.6.6. CustomComposite .......................................................................... 34
   6.6.7. Create your own type ................................................................... 34
6.7. Using image bundles ........................................................................... 34
6.8. Adding a new form .............................................................................. 35
6.9. Adding and editing publication forms .................................................. 36
6.10. Using the internationalization .............................................................. 37
   6.10.1. Constants ...................................................................................... 37
   6.10.2. Messages ..................................................................................... 37
6.11. Using the database ............................................................................... 38
6.12. Adding a new service ........................................................................ 38
   6.12.1. Creating the interface of the service ............................................. 38
   6.12.2. Creating the asynchronous interface ......................................... 39
   6.12.3. Implementing the interface .......................................................... 39
   6.12.4. Adding the service ....................................................................... 39
   6.12.5. Creating the service .................................................................... 39
6.13. Change the login system .................................................................... 40
6.14. Create a new form strategy ................................................................. 40
7. Conclusion .............................................................................................. 41
   7.1. Old system and new system ................................................................. 41
   7.1.1. Web application ............................................................................. 41
   7.1.2. Database design ............................................................................ 41
   7.1.3. User interface for publishing documents ....................................... 41
   7.1.4. Searches ....................................................................................... 42
   7.1.5. Login .............................................................................................. 42
8. Further improvements .............................................................................. 43
   8.1. Motivation .......................................................................................... 43
   8.2. Configuration of the web with XML files ........................................ 43
   8.2.1. Specification of the sections ......................................................... 44
   8.2.2. Forms ........................................................................................... 44
   8.2.3. Content ......................................................................................... 44
   8.3. Add a control panel and WYSIWYG editors ................................... 45
   8.4. Using drag & drop technologies ....................................................... 45
   8.5. Improvement of the database ............................................................ 45
   8.6. Link between RICS and Diva .............................................................. 45

III
Figure index

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Traditional Web applications</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Asynchronous Web application</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Asynchronous Call sequence</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Diva System workflow</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Requirements</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>Design (Layout)</td>
<td>13</td>
</tr>
<tr>
<td>4.2</td>
<td>Design (Architecture)</td>
<td>14</td>
</tr>
<tr>
<td>4.3</td>
<td>Design (Sections)</td>
<td>16</td>
</tr>
<tr>
<td>4.4</td>
<td>Design (Services)</td>
<td>18</td>
</tr>
<tr>
<td>4.5</td>
<td>Design (FormElement)</td>
<td>19</td>
</tr>
<tr>
<td>4.6</td>
<td>Design (FormView)</td>
<td>20</td>
</tr>
<tr>
<td>4.7</td>
<td>Design (Constraints)</td>
<td>21</td>
</tr>
<tr>
<td>5.1</td>
<td>Example of Image Bundle</td>
<td>23</td>
</tr>
<tr>
<td>5.2</td>
<td>GWT basics</td>
<td>24</td>
</tr>
<tr>
<td>5.3</td>
<td>ListMenu widget</td>
<td>26</td>
</tr>
<tr>
<td>5.4</td>
<td>DatePicker widget</td>
<td>26</td>
</tr>
<tr>
<td>5.5</td>
<td>MenuBar widget</td>
<td>26</td>
</tr>
<tr>
<td>5.6</td>
<td>Button widget</td>
<td>26</td>
</tr>
<tr>
<td>5.7</td>
<td>TextBox widget</td>
<td>26</td>
</tr>
<tr>
<td>5.8</td>
<td>Tree widget</td>
<td>26</td>
</tr>
<tr>
<td>5.9</td>
<td>HorizontalPanel widget</td>
<td>26</td>
</tr>
<tr>
<td>5.10</td>
<td>VerticalPanel widget</td>
<td>26</td>
</tr>
<tr>
<td>5.11</td>
<td>DockPanel widget</td>
<td>26</td>
</tr>
<tr>
<td>6.1</td>
<td>Example of a Page</td>
<td>31</td>
</tr>
<tr>
<td>7.1</td>
<td>Old publication interface</td>
<td>42</td>
</tr>
<tr>
<td>7.2</td>
<td>New publication interface</td>
<td>42</td>
</tr>
<tr>
<td>8.1</td>
<td>User interface</td>
<td>47</td>
</tr>
<tr>
<td>8.2</td>
<td>Parts of the user interface</td>
<td>47</td>
</tr>
</tbody>
</table>
1. Introduction

Web applications are taking an important role in nowadays companies and organizations, it allow them to remain competitive in the World Wide Web by giving users services from wherever they are.

Since the beginning of the web expansion, there have been lots of difficulties developing these applications; mainly this difficulty was caused by the different interpretations of the Web standards from different browsers. This affected not only the presentation; it also affected the programming part.

These problems made the developers take alternatives that made Web applications to be far less powerful and useful compared to what an application should be.

This thesis will analyze how Web applications use to be done, and show how the new approaches could help in building applications more user friendly, easy to maintain and extend, and more reliable. For demonstrating the advantages of these new approaches a real application will be redesign by taking advantage of these new ideas.

1.1. Structure of the report

1. Introduction: Introduction to this thesis
2. Background: Approach to the facts that motivate this thesis.
3. Analysis: Explains the system purpose and requirements.
4. Design: Design of a solution for the requirements.
5. Implementation: Information of how the design was implemented.
6. Maintenance: Explain how to maintain or extend the application in the future.
7. User manual: Shows the administrative operations that can be done in the system.
8. Conclusion: Conclusions got from the experience of this project.

1.2. Content of the CD

The current documentation came together with a CD with all the information about this thesis.

- **Source code**: Source code for the application
- **Compiled version**: Version of the system ready to execute in a browser.
- **Javadoc**: Documentation of the source code (API).
- **Report**: Contain this document.
- **Presentation**: Contains the final presentation of the thesis.
2. Background

The aim of this thesis is to study the newest Web technologies and test them by redesign an old application. The application chosen for this is the publication system that is being used in the Computer Science Department of Växjö University.

This chapter will show the two main systems related with this publication system, as well as an introduction of how a traditional web application works and why we would want to change that.

2.1. Web applications

Last decades we have been experimenting an exponential growth in the use of Internet, this made rethink the applications developers about another approach in software engineering; Web applications.

Web applications have the following advantages:

- **Distributed**: The application is not going to be in the local device, so the data will be stored in another machine. For this reason, the application would be accessible from any part of the world in the same way.
- **Accessible**: A webpage is accessible from any machine with any operating system as long as they have a browser; this includes mobile phones and PDAs.
- **Reliability**: Since the application is in a server, there could be another servers acting as backup server, if the server fails, a backup server can take the place of the main server.
- **Performance**: The last technologies like cloud computing, allows applications to make expensive computations that would be impossible in a desktop application.
- **Centralization**: All the data is centralized in one point, this is very useful for example for monitoring users activity.
- **No installation**: A regular browser would be enough for using a web application. Desktop applications need installations what make the computer to use their own resources (space, memory, processor).
- **Updates**: The developers control the upgrades because the user does not need to install the application, an update will be reflected immediately, this makes all the users to have the same version of the software. This way it is avoided the problem with software compatibilities of desktop applications.
- **Scalable**: Web applications can grow easily because since they are distributed, they can be hosted in several servers so each server may solve a small problem of the whole application.

There have been millions of web applications since Internet became so popular, lots of technologies were born, and others disappear, so now we are in the middle of something called Web 2.0, characterized for the high interaction with the users and for having applications that are maintained by the user community.

2.2. Problem with traditional Web applications

For explaining which decisions have been taken in the implementation first we have to take a brief look into the structure of traditional Web applications.
Every web application has a server side and a client side. In the client side we have the user interacting with the browser, each time the user follow one link, the browser ask for an URL to the server by making a HTTP request. The server will process this request and generate an HTML file that once ready it will be sent back to the client.

In the meantime while the server is generating the HTML, the user is waiting for the result without being able to interact with the browser anymore.

2.2.1. Unresponsive browser
As we explained before, when the user makes the new request, the browser becomes frozen until it receives the new page from the server, so for some seconds the user can do nothing in the Web application, this make the web developers to think about other alternatives.

More over, normally when we change between two pages in a traditional Web application, there are some areas that remain still in the page; actually the part changing is only a small percentage of the whole page. However the server must recalculate also the unchanged areas, include them in the HTML page, returned to the client, send across the network using bandwidth for sending information that was already knows. Besides, the client has to analyze again something that was already analyzed.

2.2.2. Problem with client side programming
The client side programming it is usually done in JavaScript, this language is implemented by most of the actual browsers, and although is a very powerful technology its use is very complicated, mostly because every browser has its own implementation, and even different versions of the same browser need different JavaScript code, so the programmers have to deal with lots of different among the browser, making the Web application no reliable.

The programmers are force to use patches like the next example, where in pseudo-code we illustrate how the client receive the code for the different browsers.

```
if (browser=explorer v4.0) {do this and this....}
else if (browser=explorer v7.0) {do this and this....}
else if (browser=firefox v2.0) {do this and this....}
```

2.2.3. Overuse of server side programming
Because of the problems in client side programming, developers usually implement the main functionalities in the server, and the optional programming in the client; so in case it fails the client side, the application still works.
This produce, that servers are overloaded with thousands of requests, and they have to do complex calculations when the client machine could do them and make the servers to safe processing time for other tasks more important.

2.2.4. **Designs with dozens of small images**

When a designer, is making the user interface in a web application, it will use many small images like round borders, gradients, icons, etc.

Each image produce and HTTP request to the server, and normally the server will answer with one "you already have the last version of that image" message. So there is dozens of communications when normally these images are not changing that often.

And when the image change, the server sends a small image in a TCP package, this will have a large overhead since sometimes the size of the head of the IP package is even bigger than the size of the image.

Further more, HTTP 1.1 requires browsers to limit the number of outgoing HTTP connections, two per domain/port. A multiple image request will block the application avoiding them to do other request more interesting like RPC (Remote Procedure Calls).

2.3. **Webs with asynchronous calls**

Some of the problems exposed are solved in part by doing asynchronous calls to the server. These calls are done using RPC (Remote Procedure Call). An RPC is intercommunication between different processes in different machines in where one executes code of the other.

The Figure 2.2 shows the typical structure in an application with asynchronous calls. Now, the main functionality of the application is in the client side. The client interacts with the application locally, the application responds to the user input really fast. In case the application needs some data or service from the server it would make an asynchronous call.

An asynchronous call does not wait for the response of the server; it will demand information to the server without caring when this information is going to be ready. After the call the user continues interacting with the application while the call is being process. This means, that the client does not need to wait for the response in order to continue working.

Whenever the server finish processing the information, and has prepared the result, it will send this result back to the client side, so the application has the chance to take this result and present it to the client.

![Figure 2.2 - Asynchronous Web application](image-url)
This new approach has clear advantages compared to the traditional one. The client is always controlling the interface even when the server is processing a request. The Figure 2.3 shows the sequence of messages between client, browser, server of the page and a server with a service.

![Asynchronous Call sequence](image)

**Figure 2.3 - Asynchronous Call sequence**

An asynchronous call works in the following way:

1. The user type an URL in the browser
2. The browser makes an HTTP request of the web application.
3. The browser gets the Web Application
4. The browser display the application to the user
5. The user can interact with the user interface provided by the Web application. These interactions may cause the application to need to call a service. One example of a service could be a database query.
6. In case a RPC call is not needed, the application will act as consequence of the user interaction refreshing the parts of the interface that are needed.
7. In case a RPC call is needed, the browser will make it but without waiting for the result from the service. The service, will attend the call, and will be calculating the result that should be returned.
8. The browser update the interface right after making the call, it does not have the result, but it has a chance to tell the user that the calculation is being done. While the service is making the task, the user goes back to step 5. This
means that while the service is making the task, the client can still be interacting with the interface.

9. Whenever the result is ready, the asynchronous service will return the result to the application.
10. So the browser will show the result of the RPC call to the user.

The leading technology for this new approach is AJAX (Asynchronous JavaScript and XML). Actually is not a new technology, is the combination of technologies that already existed.

The problem of AJAX is that, because is based in JavaScript, every browser will have its own implementation; this will make the programming and debugging very difficult.

2.4. DIVA

The new aproaches for building a Web application are going to be applied into a real application. The application choosen is a system for publishing electronic documents in Växjö University, this application will be presented in chapter 3.1.

Diva is important for this thesis because is another system for publishing documents and if it is possible the system should work very close with Diva.
DIVA, the Academic Archive Online (Digitala Vetenskapliga Arkivet - in Swedish) is a publication system for universities in Scandinavia. The system was carried out in the Electronic Publishing Centre at Uppsala University Library. The project started in 2000, and it has been in full operation since 2003.

As we can see in Figure 2.4, the authors are allowed to publish a paper using two different procedures; a web interface or a word processor template. Once the publisher gives the information using one of these procedures, the system will transform all these information and generate an intermediate XML file (the DIVA Document Formant - DDF). This Diva document will be the internal representation of the document, and its used for generating any kind of representation of the document; Electronic book, PDF, etc.

All the documents have assigned a permanent identifier.

The problem with DIVA is that every time a document is published in the RICS system, the same document must be published in DIVA, so you have the same document in two different places and you do the same task twice.

Unfortunately, this problem does not have an easy solution due to DIVA does not allow to access to its database using our interface. In the other hand, they let to do searches using RSS feeds.

For more information on DIVA check the reference [2] in the literature section.

![Figure 2.4 - Diva System workflow](image-url)
3. Analysis

This chapter explains the system purpose and the requirements. The system that is required must improve the current system RICS, this new system must solve all the problems that the old system has. This chapter also shows the requirements of the system, the domain, and the user.

3.1. RICS

RICS (Research In Computer Science) is the Computer Science Department in Växjö University. There is lots of work getting done in this department; research, master thesis, degree thesis, papers, etc.

All this documents need a system for being stored, this way it would allow to look for previous papers, and provide a useful service for teachers, students and researchers.

RICS has a webpage built using traditional web application technologies; the webpage allows you to get information about all the projects ongoing and finished, about the people like students, researchers, and a lot more information. For more information about RICS, check the reference [1] in the literature section.

The idea of this thesis is not to redesign the whole application; the idea is to show how to redesign an important part of it so it would be easy to add parts left for the future. So from now on, we will be more focused in just the part of the publication system.

RICS uses a database for storing all the information. The webpage and the database have a problem in their design and luck of functionality. The main problems with the actual application are:

- **Web application**: It uses traditional technologies, for this reason it will have all the problems that were explained in section 2.2 Problem with traditional (Page 2).
- **Database**: The database is not very well designed and is not robust.
  - If you are not very careful with the data you insert you may have incoherent information on it.
  - The database does not check that the data inserted is correct. This produces the information not to be reliable.
  - All the publications are inserted in the same table. Each type of publication needs only a specific set of attributes so all the rows have most of the attributes null. This is a waste of space and confusion at the time of inserting a new row because you do not know which attributes you need to fill in.
  - It does not have any documentation
  - The attribute names are in different languages depending on the table (Swedish, English and German)
- **Publishing a paper**: The web page does not allow inserting data into the database.
  - For inserting data, a user must do it directly to the database.
  - Every publication has only a few fields to fill, the database will ask the user for all the common attributes of all the kinds of publications. It would be easy to make a mistake at this point.
The relations between the authors and the publications must be done also by hand, so the user is forced to remember the publication id, the person id and insert a new row directly into the table that make the relation between persons and publications.

- **Language support**: The language support presents also some problems
  - Although it supports English and Swedish, this is only partially true, this support is only for the main menu and a few more texts, and it is far from being multilingual.
  - It would be traumatic to add new languages into the system

### 3.2. User requirements

The following case use diagram show the main four functionalities that the system must have.

![Diagram](image)

**Figure 3.1 - Requirements**

- **Login**: It should have a login system that allows a user to identify its self and access some private areas.
- **Sections**: The application should allow having different sections. This section could:
  - Have subsections
  - Be invisible
  - Be private (need to login)
• Be forbidden

• **Publish paper:** This functionality allows the identified user to publish a document of any kind.

• **Search:** The system will let the user to perform searches within the documents. There will be a simple search, an advanced search and a simple search for the Diva system.

• **List:** The user can list the different documents. For each publication document the system it should allow to:
  o See the relevant information
  o See the authors. I will give the option of get more information about them.
  o Edit the publication.
  o Delete the publication.
  o Export it in different formats like Bibtex.

3.3. System requirements
The system requirements are the requirements from the developer’s point of view.

3.3.1. **Functional requirements**
The functional requirements describe the functionalities of the system from the developer's point of view.

- **Login:** Some areas will be protected by login
  o Once the user has successfully log in, can enter in the restricted areas
  o This login system should be generic so it could be implemented using different technologies.
  o A change in a login system must not affect other parts of the design

- **Publish paper:** A logged user can publish any kind of document.
  o The user will select which kind of document he wants to publish. Then the system will just require him the data needed by this particular kind of document.
  o The required fields will be shown, as well as the optional ones.
  o The system must check the user input for avoiding inserting inconsistent data into the database.
  o The relationship between person and publications should be done in a user-friendly way, for example selecting a name from a list and not remembering identification numbers.

- **Search:** The user can make searches for looking for publications.
  o The simple search in the RICS system requires the user to fill a simple text box and the system will show the result of matches with this word.
  o The advanced search in RICS system allows the user to make searches specifying a more specific search.
  o It should make a simple search in DIVA system.

- **Database:** It must allow connecting to a database

- **Internationalization:** The application must be multilingual
  o It should support multiple language
  o It should be easy to add a new language.

- **Messages:** The messages and dialogs with the user should be centralized in one point.
### 3.3.2. Non-functional requirements

- **Database:** The database connection must be designed taking into account:
  - The connection and queries should be done in an independent way of the Database Management System, this way in case of need to change it the system will not be affected.
  - The operations with the database should be done in an asynchronous way for keeping the interface responsive while the server is processing the query.

- **Internationalization:** When a user accesses the application, the system should dynamically send him only the language he needs instead of sending him all and being the user the one that choose language.

- **Browser:** It should work in all the main browsers

- **Reduced number of connections:** The system should try to reduce until the minimum the number of HTTP connections for loading the page. This means to reduce the number of images and files to load.

- **Asynchronous calls:** All the calls to the server should be asynchronous so the user does not have to wait for a response for continue using the application.

- **User interface:** The user interface must be easy to understand, the user should find the functions in a logic and natural way.

- **Navigation:** The navigation throw the different sections of the application should refresh just the part of the interface needed in order to go from one section to the other. In any case the entire application should be reloaded entirely.

- **Messages:** The messages and dialogs with the user should be centralized in one point.

- **Easy to extend:** The application should be prepared to grow in time.
  - Easy to add and delete a section
  - Easy to add or modify a form for publishing a document
  - Easy to modify the layout
  - Easy to build new asynchronous services

### 3.3.3. Domain requirements

There are some restrictions in the way the application can be designed. Because we are taking an application that already exists there are some requirements that should be followed.

- It should work with the actual database, is not possible to make a redesign in the short term.
- The fields that are required and optional are specified by the Bibtex format
4. Design

All the designs were done using the Object Oriented approach due to the demonstrated benefits that brings its use, like abstraction, encapsulation and modularity. It is also used many design patterns because they are elegant solutions for many common problems.

4.1. Concepts

There are some concepts that are important to have clear in order to understand the design documentation.

4.1.1. Entry point

There is a single object that represents the whole application. The role of this object is to coordinate all the components of the application, and is an entry point. An entry point is the part of the application that will be inserted in a web page.

In this design, the entry point is called PublicationSystem. This object will have the main functionalities of the system and will be composed by panels that form the layout.

4.1.2. Widgets

The application is designed in a modular way. Widgets are small pieces of functionality with a graphical representation. A widget could be from a check box, to a complex interface. There are some special widgets called Panels that can contain other widgets.

The “entry point” will use panels with widgets for constructing the interface.

4.1.3. Sections

The content of the application is organized using Sections. A section could be seen as a part of the application. Each section contains information about the content as well as the content itself. This content that is associated with a section is called Page, and is a kind of widget.

Sections can contain other sections.

4.1.4. Server side and Client side

All Web application has to main parts: the client side and the server side.

The client side is the part of the system that is executed in the client’s computer; this is in the browser.

There are some tasks that cannot be executed in the client side. These tasks must be executed in the server. An example of this is a database query.

For the tasks that cannot be executed in the client side we need a Service running on a server. The service will do the task and return the result to the client side.

A service has also a client side and a server side. The client side of a service is in charge of making the connection to the server, sending the parameters or data of the call, and get the result. The server side of the service has to be listening for calls, get the parameters or data, execute the service itself and send back the result.

4.2. Layout and user interface

In the layout we need a:

TopPanel: Is a panel that contains basic information about the application, it will show the logo and some controls for changing the language of the user.
**ContentTypePanel**: It shows the content of one section. It is the variable part of the application. When the application change from one section to another, this part will be refreshed.

**MenuPanel**: It contains the list of sections of the application.

**LoginPanel**: It contains the login form. Once the user is logged in, it contains logging information.

**BottomPanel**: It is the bottom part of the application; it will show just contact information and the copy write.

![Figure 4.1 - Design (Layout)](image)

### 4.3. High level design

The main class in the system is **PublicationSystem**, this is the class that represents the whole application. It is represented in the Figure 4.1 - Design (Layout).

This **PublicationSystem** class will have associated the panels **TopPanel, MenuPanel, LoginPanel, and ContentPanel** as can be seen in the Figure 4.2 - Design (Architecture) and as it can be deduced from the layout distribution in the last section.
Although there is more panels (like BottomPanel), here it will be explained the relevant ones.

- **MenuPanel**: It will show all the sections of the application. For this reason it will have associated a class that would represent all the sections of the application. This class is a **CustomComposite** that will be explained in the chapter 4.4 Sections design (page 14). This **CustomComposite** is a kind of section that contains other sections and at the same time has associated one **Page**. This class is acting as the root of the hierarchy of sections, and this is why the page associated to this class is the home page.

- **ContentPane**: represents the actual page, it has associated one section as it will be explained in the chapter 4.5 Navigation system (page 16).

- **LoginPanel**: is going to be the component in charge of log-in the user as it will be point out in the chapter 4.6 Services (page 17).

- **TopPanel** is just a panel with the title of the application and some links to the different language versions of the application.

All the Panels will be assembled in the **PublicationSystem**; this class will use a singleton design pattern for giving access to its methods for all the different components of the application.

### 4.4. Sections design

In order to have different content in different sections we need some kind of structure to hold this content. As it was specified in the requirements, this structure should supports
nested sections, private sections and forbidden section. All the different kinds of section should have some common properties, this means that it should belong to the same abstract class. This class is SectionAbstract.

All the sections will extend from SectionAbstract, this define the interface of one section.

- **SectionLeaf**: Is the basic section, it does not have any subsection and has a Page associated. This page is actually the content of the section.
  - **Page**: Represents an UI element that can be displayed in the content panel. A SectionLeaf is associated with a page because when the application wants to load a section, it will load this page in the content panel.

- **SectionComposite**: is an abstract section that may contain other sections. It has two implementations and it may appear more in the future.
  - **SimpleComposite**: It is a section that shows a list of all the subsections of the section. It has a page associated; this page has the duty of drawing all the subsection. This page is called SimpleCompositePage.
  - **CustomComposite**: It allows you to set up a custom page for this section.

- **SectionDecorator** is an abstract section that decorates or modifies another section. For doing this, redirects all the calls to its methods into the section that is decorating. This is like a default behaviour that will inherit the concrete decorators. This way, any class that extends SectionDecorator could override the methods that want to alter the behaviour.
  - **SectionLinkSymbolic**: Allows having in the menu one section two times with a different name (a similar concept to a symbolic link in UNIX).
4.5. Navigation system

The navigation system is how the application changes for one section to another. The navigation system is different as the traditional one in web applications. The traditional navigation system makes an HTTP request each time that wants to change the page or make some processing. This application is being designed for using the new technologies of the web 2.0, this is AJAX, which make asynchronous calls to the server and update only the elements that should change in the application.

The web page will use something call tokens for managing the navigation, a token is a string of text in the URL after the character #. For example:

http://localhost:8080/PublicationSystem.html#Home
http://localhost:8080/PublicationSystem.html#Search

When a hyperlink redirect to "page.html#token", the web application does not produce a refresh of the page, instead you have a chance to read this token, and load the information that is needed, in the case of this application it will set the SectionAbstract needed into the ContentPanel. For looking for the section, it can be used the method

+ search(classname: String): SectionAbstract

from the CustomComposite that is associated with MenuPanel. This CustomComposite will contain as children all the sections from the web application, and as the home page.

This design, achieve to separate the content from the functionality, and make the application easier to maintain.
4.6. Services

The application will need to make some tasks that can be done just in the server. This is the case of the login system and the database service.

Each service should have an interface, which the server and client share; this interface will work as an agreement of the service between the client side and server side.

We cannot use synchronous calls for serving the client because this will freeze the application in the client side until the server sends the result. So what is done is an asynchronous version for each method of the interface of the service.

This asynchronous version of a method will not return anything because we are not waiting for the result. Instead, we send an `AsyncCallback` class that will be called by the server whenever the result is ready for being sent to the client.

This callback class will have to methods, one in case the operation succeed and one in case the operation fails (for example because one exception is thrown). The succeed method will have as argument the same data type of the result of the operation of the synchronous version.

For instance, if we have a method that sum two integers the synchronous interface will have the following method:

```java
public Integer sum(Integer a, Integer b);
```

And the asynchronous version will have:

```java
public void sum(Integer a, Integer b,
                AsyncCallback<Integer>);
```

So the server will have just to implement the method:

```java
public Integer sum(Integer a, Integer b) {
    return a+b;
}
```

In the same way as this example, we create the interfaces and implementation for the services of login and database that are shown in the next figure.
As we can see in the figure, the duty of invoking the login is of PublicationSystem, so that means that when the login panel wants to log in a user, it calls to the method in PublicationSystem. The publication system will update all the parts of the interface that needs to be refreshed.

The database service implementation is transparent to the user; it allows you to make a query in case of a SELECT sentence, or update in case of INSERT, UPDATE or DELETE sentences.

4.7. Form system
The form system allows building forms for different kinds of treatments of the data captured on them.

The form system makes a separation between the logic part and the physical part. The logical part consists in the data entries and the constraint associated to them. The physical part consists in the form fields that the user will use for input data. At the same time, the design separate also the functionality, that means that the same form can have different treatments for the data on it.

The physical part is composed by elements; these elements are subclasses of the abstract class FormElement. The simplest form elements are subclasses of SimpleField, for example HiddenElement and TextBoxElement.

We can compose form elements in a row or in a column by using HorizontalFormComposite and VerticalFormComposite respectively. Both of them are subclasses of FormComposite.
In a similar way to the decorator in the Figure 4.3 - Design (Sections), the **FormDecorator** allows to decorate a form element. For example **LabelFormDecorator** adds a label to a form element, and **WidgetOverElementDecorator** adds any widget over a form element. A widget could be just anything, like an image, text, HTML, etc.

![Figure 4.5 - Design (FormElement)](image)

For constructing the form, we use a **FormView**, this class represents the whole form, and it gathers the physical part, the logic part, and the functionality.

**FormView** will have several **FormElements** associated, at the same time; each **FormElement** contains at least one **AbstractFormEntry**.

The **AbstractFormEntry** is the logic part of the form, it represents one data unit. It could have associated a constraint. Each constraint implements the interface **AbstractConstraint**.

**FormView** can be configured with different functionalities for the data treatment; we achieve this by using the strategy pattern. This way a **FormView** will be configured with a **StrategyForm**. We can see in the Figure 4.6 - Design (FormView) that the method **execute()** in **FormView** delegate the treatment of the data to the strategy.

When the strategy finish processing the form, it calls to all the callbacks that are set up in the strategy. For instance, one possible strategy, is the one for inserting data in a database (**InsertStrategy**), but could be done any other for example for updating a row in a database, send a mail, send a SMS, generate a file, etc.
For the constraints, it was defined also a hierarchy using composite for making the AND and OR operators. Then there is some constraints implemented, so it would be easy to create new ones by compose them with the AND and OR operators.

For instance if we associate a `IsInteger` constraint with an entry, when the form is validated the method will return `null` in case there is no error, or it will return a `String` explaining the problem in case is not an integer.
It is easy to create new constraints, you only need to implement the **AbstractConstraint**. You could always add new restrictions to the forms.

For more complex constraints, we can compose them using the **AndCompositeConstraint** and the **OrCompositeConstraint**.

In case we need constraints that affect to more than one entry, we need to extend the class **FormView** for overriding the method "validate", and add the new complex constraint, for example it could be useful for declaring logic like "The value of entry A must be more than the double of the sum of B and C".

![Diagram of Design (Constraints)](image-url)
5. Implementation

In this chapter the technologies for the chosen application will be shown. Furthermore it will also be illustrated how this technology fits into the design.

5.1. GWT - Google Web Toolkit

GWT is a quite new technology, it was chosen for the development of this project because it solves in an elegant way most of the problems that other web technologies have.

This section presents this technology and explains briefly which advantages present GWT compared to traditional technologies, for more information on GWT check the reference [3] in the literature section.

5.1.1. Java oriented in client and server side

The GWT uses Java for designing the client side (interface and programming), and then when you compile the application the code translated into JavaScript.

When you are programming the server side, you use real Java code, so you will be able to use client objects in the server since both are written in Java. The client and server could invoke methods and send objects across the network in a transparent way for the user.

The server side will not be translated when compiling the application. This means that it will run in the server side using the JRE (Java Runtime Environment), therefore it will use 100% java code, and it could import any library like any other Java software.

5.1.2. Programming and debugging

Because we use Java in both, the client and the server, we can use object-oriented techniques for designing the application, or debugging and testing techniques, like using jUnit testing and so forth.

We are also inheriting all the advantages from the object-oriented programming, like reutilization of the model for other applications, even for desktop applications.

5.1.3. JavaScript works in all browsers

When we make the compilation of the Java code of the client side, GWT will generate the JavaScript equivalent for all different browsers, and it will give in execution time just the version the client needs. The client will not receive the version for other browser as it usually happens with in AJAX programming. This concept of dynamically fetch only the information the client needs is called "Deferred Binding" and is a powerful feature of GWT.

5.1.4. Asynchronous calls

Following the same idea than in AJAX, GWT allows you to make asynchronous calls to the server and make the application to remain listening user events and attending them.

The client makes a call to one of the methods of the server, by passing a call-back object with the other arguments. The invocation of the method finish before the server attends it, so the client is attending events while the server is processing the call. When the server has the result ready, it will pass the result to the call back, who will use this
result for performing some tasks, for instance updating the user interface to inform the user about the result.

5.1.5. **Light servers**
Since we can use more the client side programming, we live the server to have more processing resources for the important job; serving these asynchronous calls.

At the end, we will have faster clients, and faster servers, this is more performance that implies a happy user and more chances of accomplish our goal.

5.1.6. **Image bundles**
In order to solve the problem exposed in the chapter 2.2.4 Designs with dozens of small images (page 4), GWT proposed an elegant solution; Image bundles.

The idea of Images bundles is that instead of having a group of small images that will be sent one by one, we group all of them in a single image for travel across the network, and once in the client side, they will be spitted again using client side programming.

GWT have mechanism for doing this in a transparent way for the user and developer. By reducing the HTTP connections done because of the small images, we avoid blocking other request more critical for the correct workflow in a web application.

![Figure 5.1 - Example of Image Bundle](image)

The Figure 5.1 could be an example of an ImageBundle after compiling the application.

5.1.7. **Support for internationalization**
GWT has also integrated a module for internationalization that makes easy to build and maintain a multilingual application.

All the messages that are showed in the applications are centralized in file. There is a file for every language the application supports. The client only receives the language he needs.

It goes deep, not only the language is translated; also the formats of the dates and numbers will be displayed in the proper format.

5.1.1. **Third parties libraries**
Around GWT, there is thousands of open source projects, ready to plug & play in your application with different functionalities like drag and drop, PDF creation, lots of widgets, sounds handling, image transformations, etc.

5.2. **Database management**
GWT accept using almost all the java classes, however, there are classes that cannot be translated. It is not possible to use reflexion or to use directly an external library.

For this kind of scenarios we need the server side programming. It is going to be created a service in the server side that will make the connection to the database, execute the queries and return the result for the client.

Because this application has to be integrated to the current publication system of RICS, it will use the MySQL driver for making this connection, however it could be changed in the future without recoding the client side.
5.3. Implementation in GWT
This section tries to show how GWT can be applied to the design.

5.3.1. Basic GWT elements
GWT is a large library with lots of classes, this sections does not pretend to be an exhaustive list of classes, for this use the GWT API.

An entry point, widget and services are concepts that were explained in the section 4.1 Concepts (Page 12).

As we can see in the Figure 5.2, the heart of a GWT application is the entry point, it contains several widgets. The entry point is inserted into the body of an HTML document called "Host page". This host page is the web application that will run in the user browser.

Whenever the application wants to make a call to a service it will be done using an interface that is the contract between the client side and server side.

For understanding better GWT, some of the most representative classes will be listed bellow:

- **EntryPoint**: The most important element in GWT is the entry point; this element is going to be inserted into the body in the host page. It is the class that represents the whole application. It is an interface with only one method:
  - `void onModuleLoad()`: Called when the application has been loaded.

- **Widget**: The base class for the majority of user-interface objects. Widget adds support for receiving events from the browser and being added directly to panels. Some of the main subclasses of Widget are:
  - **Image**: Displays an image
  - **Label**: A widget that contains arbitrary text
- **HTML**: A widget that can contain arbitrary HTML.
- **Hyperlink**: A widget that serves as an "internal" hyperlink.
- **Panel**: Abstract base class for all panels, which are widgets that can contain other widgets.
  - **HTMLTable**: Has to subclasses:
    - **FlexTable**: Table that creates cells on demand
    - **Grid**: A rectangular grid
  - **SimplePanel**: Base class for panels that contain only one widget.
    - **PopupPanel**: A panel that can "pop up" over other widgets
  - **ComplexPanel**: Abstract base class for panels that can contain multiple child widgets.
    - **CellPanel**: A panel whose child widgets are contained within the cells of a table.
      - **DockPanel**: A panel that lays its child widgets out "docked" at its outer edges.
      - **HorizontalPanel**: A panel that lays all of its widgets out in a single horizontal column.
      - **VerticalPanel**: A panel that lays all of its widgets out in a single vertical column.
    - **FlowPanel**: A panel that formats its child widgets using the default HTML layout behaviour.
  - **Composite**: A type of widget that can wrap another widget. It is very useful for building custom widgets using other widgets. This widget that the composite builds have to be passed as parameter to its method `initWidget(Widget widget)` in the constructor.
- **FocusWidget**: Abstract base class for most widgets that can receive keyboard focus.
  - **TextBoxBase**: Abstract base class for all text entry widgets. It will have subclasses like `TextArea` and `TextBox`.
  - **ListBox**: A widget that presents a list of choices to the user, either as a list box or as a drop-down list.
  - **SimpleCheckBox**: A simple checkbox widget, with no label.
  - **ButtonBase**: Abstract base class for `Button`, `CheckBox`, `RadioButton`.
Some visual examples of widgets are:

- **Figure 5.3** - ListMenu widget
- **Figure 5.4** - DatePicker widget
- **Figure 5.5** - MenuBar widget
- **Figure 5.6** - Button widget
- **Figure 5.7** - TextBox widget
- **Figure 5.8** - Tree widget
- **Figure 5.9** - HorizontalPanel widget
- **Figure 5.10** - VerticalPanel widget
- **Figure 5.11** - DockPanel widget

It is easy to create new widgets by composing other widgets. Then, you could reuse your own widgets for different parts of the application, or even for other projects.

### 5.3.2. Using GWT in the design

Let's explain how was the matching done between GWT and the design. Since it is going to be mention many different classes in this section, it will be written in italics the GWT classes for not confuse them with the ones of the design.

The main class of the application is **PublicationSystem**. This class must be the **EntryPoint**, so it will have the method `onLoadModule()`. In the body of this method it creates all panels, create the layout and insert this layout in the body of the host page.

For creating the layout, the main class must create all the panels, once created it needs a container for laying them. We need a widget that contain other widgets, this is a
Panel, but we need to setup all the panels in a kind of cells, see Figure 4.1 - Design (Layout), so we need a CellPanel. There are different kinds of cell panels, but the easiest way to create something like the layout in the image is by using a DockPanel.

Once we have the dock panel, we add the panels in the following way:

- We add the TopPanel in the "north".
- We create a BottomPanel, since this is very simple it will be just an HTML widget. We add it to the dock in the "south".
- We create a VerticalPanel (LeftPanel) and add MenuPanel and LoginPanel to it. Then we add this vertical panel into the dock's "west".
- We add the ContentPanel into the "centre".
- The last thing will be inserting the dock panel in the body of the host page.

Every panel in the design (TopPanel, MenuPanel, LoginPanel, ContentPanel) are user-interface elements with support for receiving events from the user. This means that they all are Widgets. But a widget is not enough for representing the panels; these panels will be composed by other widgets. Thus all panels, and in general all custom widgets, extend the class Composite.

For constructing the panels we follow a similar strategy as the way we composed the dock panel:

- Draw in a paper the widget we want to construct.
- Think which panels we need for laying all the smaller widgets in a similar way as our design.
- Create the smaller widgets. In case these widgets are complex is a good idea to create them in its own class (back to the first step).
- Add them to the panel
- Initialize the widget in the constructor with the panel you just built;
  initWidget(YourPanel);

5.4. Implementation results

During this chapter, we saw how GWT solves every problem of the traditional way of programming a web. You can check the problems of traditional technologies such as PHP in section 2.2 Problem with traditional Web applications (page 2).

With GWT we achieve to build a powerful application with:

- **Object Oriented methodologies**: Help in the software engineering tasks.
- **Asynchronous calls**: Allow the client to work on the application while the server is doing some computations.
- **Client side programming**: Make application to work smoothly and faster, free load in the server side so they have more resources for important tasks.
- **Internationalization**: Provides support for building multilingual applications.
- **Third parties libraries**: Allow using external Java libraries.
- **Services in pure Java**: Let us to communicate the client side with a service built in Java code. This is especially useful for example to connect to the database.
- **Easy user interface**: The user interface is built by composing widgets in a similar way to libraries such SWING.
This features allows us to accomplish our goal in an elegant way as it will be explained in chapter 7 - Conclusion, on page 41.
6. Maintenance

This chapter explains how to maintain the application and extend some important parts. It also helps to give a better understanding of how the application works.

6.1. GWT basic application

A better understanding of this project's file organisation is reached if you understand the structure of a basic GWT project.

There are two main parts in every GWT project:

- **Public folder**: It contains all the static elements of the application. It contains the host page, the CSS and images, the server configuration, the compiled servlets, and the JavaScript generated by the client side code. This folder is called **war**

- **Source code**: It is the code of the application. It is organized in packages. For instance, the root of the project for a company with home page "www.company.com" should be "com/company". In the case of our application the root of the source code is in "se/vxu/msi/cs" so the logical name of the package is "se.vxu.msi.cs".

There are three elements inside this source code:

- **public**: It is a package that contains all the client side of the application. When compiling the application, all the classes inside this package will be translated into JavaScript and put it in the folder **war/publicationsystem**. The entry point of the application will be in this package, **PublicationSystem** in this case.

- **server**: It is a package with the server side of the application. It contains all the services that the client side can use. The classes in this package will be compiled as regular Java code and the generated bytecodes will be put it in the folder **war/WEB-INF/classes/**

- **PublicationSystem.gwt.xml**: It is the configuration file, it will be explained in 6.2.1 GWT configuration file (page 29).

6.2. Configuring the basics

The web application has different configuration files; each one has its function.

6.2.1. GWT configuration file

This file configures a GWT application, define the modules to load and set up the application. This small guide does not pretend to be very detailed; the aim is to show the function of this file for understanding how GWT works. For more information on this visit the GWT developers guide (see the Literature section).

The file is placed in **se/vxu/msi/cs/PublicationSystem.gwt.xml**

Individual units of GWT configuration are called **modules**. A module bundles together all the configuration settings that your GWT project needs. The following lines load the user module and the internationalization one:
The user module allows you to use the GWT objects like \texttt{Widget}, \texttt{SimplePanel}, etc. The internationalization gives language support to the application (see \texttt{6.10 Using the internationalization} on page \texttt{37}).

You can load third party modules in the same way, as well as create your own module and reuse it for future deployments.

This file is also the place for specifying the theme. In this case it is used the standard one.

The most important line in this file is the specification of the \texttt{entry point}. The \texttt{entry point} will be the class who implements \texttt{EntryPoint} (and its method \texttt{onModuleLoad}()). This class will be the class who represents the application; it will build the entire user interface and insert it in the body of the \texttt{host page}. The \texttt{host page} will be in the public folder (\texttt{war/}) and will have the same name as the entry point.

\texttt{6.2.2. Server configuration file}

There is a file for configuring the services in the server side, so the client can use them. This file is placed in:

\texttt{war/WEB-INF/web.xml}

For learning how to specify a new service see chapter \texttt{6.12 Adding a new service} (page \texttt{38}).

\texttt{6.2.3. Application configuration file}

The publication system has a class with static methods that is used for configure the application. This class is called \texttt{Constants} and is placed in the package \texttt{se.vxu.msi.cs.client.model}.

It defines information like user and password for connecting to the database, the driver that will be used, other important constants for the application and the classes for supporting internationalization.

\texttt{6.3. Modifying the layout}

The design of the layout was explained in the section \texttt{4.2 Layout and user interface} (page \texttt{12})

\texttt{6.4. Editing the style}

All widgets from GWT have a CSS style associated. Their name is \texttt{gwt-<widget-name>}. For instance \texttt{CheckBox} has associated the style \texttt{.gwt-CheckBox}, so we could modify its style from the CSS file.

If we check in the GWT API, we also find the rule \texttt{.gwt-CheckBox-disabled}, this is because sometimes the widgets have more than one style associated to different components of the widget.

Furthermore you can add your custom style into a panel.

\begin{verbatim}
leftPanel.addStyleName("LeftPanel");
\end{verbatim}

Once we now the styles name that modify the widgets and panels we are ready for edit the CSS file. This CSS file is in the public folder:

\texttt{war/PublicationSystem.css}
6.5. Adding a new section
The application is prepared to allow adding new sections easily. The sections where explained in 4.4 Sections design (page 14).

6.5.1. Understanding MenuPanel
The class in charge of registering all the sections is MenuPanel. All the sections are children of a CustomComposite called "sections". The CustomComposite is an aggregate composed from other sections and it has a Page associated as its content. In this case, because this class represent the root of the sections, it will be configure with the class Home.
Home, or any other page we want to display as content must extend the class Page. Page extends the GWT class Composite, that is the basic class for creating your own widget.

6.5.2. How create new page
For showing how to create a new page lets show the hello world example.

```java
public class HelloWorld extends Page{
    public HelloWorld() {
        initWidget(new Label("Hello world!"));
    }
}
```
In this example, we create a simple page. Every page must have a widget associated; in this case is a Label, but it could be any panel (they all extends from Widget).

For example we could do a more interesting webpage by adding some more widgets:

```java
public class ProjectsOngoing extends Page{
    HorizontalPanel outer;
    Label numTimes;
    int count=0;

    public ProjectsOngoing() {
        outer = new HorizontalPanel();
        outer.add(new HTML("You <strong>clicked</strong> "));
        numTimes=new Label("0");
        outer.add(numTimes);
        outer.add(new HTML("times");
        outer.add(new Button("Click me!", new ClickHandler(){
            public void onClick(ClickEvent event) {
                count++;
                numTimes.setText(count++);
            }
        }));
        outer.setSpacing(10);
        initWidget(outer);
    }
}
```

So this page is going to look like the following image.
In this example, we have a HorizontalPanel that is a panel that lays all the widgets in a horizontal layout. This panel will be the widget associated with the page.
We add four widgets to the panel.

- First a HTML one, that can contain arbitrary HTML code
- Then a Label that will contain the number of clicks
- Then another HTML.
- The forth one, is a Button. The button can have a ClickHandler associated that will attend the click events on that button.

The action specified for the click is just increment the counter and set up the label with the value of the counter.

6.5.3. How to create a new section
There are different kinds of sections as it will be explained in the point 6.6, but in this section we explain how to create the basic section and add it to the menu.

The MenuPanel has associated a CustomComposite called "sections". This section represents the root of the hierarchy of the sections for the application. So if we create a new section, we have to add it to this root of sections.

For understanding how to add a new section, lets show how to add a basic section. The basic section is SectionLeaf and it has a page associated. For creating a section of this type we need the following parameters:

- The page associated with the section.
- The name of the section.
- The token name (see 4.5 Navigation system on page 16). By convention we use the class name of the page.
- The image of the icon

For example, for creating a section for the page made in the last point:

SectionLeaf helloWorld = new SectionLeaf("Hello world!", "HelloWorld", null, new HelloWorld());

Where instead of the image we indicate with null that we want the default image. We could specify an image of the icon using image Bundles.

For info about image bundles read the chapter 5.1.6 Image bundles (page 23). For learning how to use an image bundle read the chapter 6.7 Using image bundles (page 34).

Once created the section, we are ready to add it:

sections.add(helloWorld);

The application will know how to display this section, so it will create an entry in the section menu and it will integrate the new section in the navigation system.

For other kinds of sections we add in the same way the section to the composite because the add method accepts a SectionAbstract.

6.6. Different kinds of sections
The sections hierarchy is quite flexible and allows making different kinds of sections for different scenarios. Further more, it allows creating more kinds of sections in an easy way.

6.6.1. SectionAbstract
Is the root of the hierarchy, it represent a section, it offer lots of methods that will be inherit by all the subsections.
In the Figure 4.3 - Design (Sections), you can see the most important methods of this class; these methods are for getting/setting the name, description, etc. Also there are structural methods for composing sections. Adding one section to another only make sense if the section is a SectionComposite, where SectionComposite is a section that contains other sections.

All the sections have properties that allow configuring its behaviour:

- **Visibility**: It is modified by the method `setVisible(Boolean b):void`. If a section is not visible, it will not be displayed in the menu, however you could access it by using its history name as a token in the URL (see 4.5 Navigation system on page 16).
  A invisible section can have visible subsections, this is useful for example for creating sections that we do not want to display in the section's menu. An example of this is the "Forgot password" section, this section is important for when helping the user in case he lose the password, however it is not a good idea to add the section into the menu.

- **Private section**: The method for setting up this property is `setPrivatePage(Boolean b):void`. A private page is a section that can only be seen if the user has logged in the system.
  The section will be invisible in the menu if the user has not done the login. Even if a non-logged in user try to access that section using the token name, the system will not allow him to see the section.

- **Forbidden section**: For making a section forbidden there is the method `setForbidden(Boolean b):void`. A forbidden section does not allow seeing the content of it.
  It is useful for blocking sections, for maintenance, for logic reasons, etc.
  Another use would be, if we combine this with a invisible section with subsections, we have sections that are hidden, and nobody can see them (even using the token name in the URL).

### 6.6.2. SectionLeaf

SectionLeaf is the basic section, it does not have any subsection, for this reason it will have a page associated. For creating this kind of sections you can follow the example in 6.5.3 How to create a new section (page 32).

### 6.6.3. Section Decorator

SectionDecorator is an abstract sections, it is used for modifying the behaviour of a regular SectionAbstract. The idea is that SectionDecorator has associated another section (the one it is decorating), so this decorator it is going to override all the methods inherited from the father (the abstracts and not abstract). This overriding is for redirecting every single call to the decorated section giving all methods in this abstract class default behaviour.

The interesting idea of this design is that any class who extends this decorator class could override the methods needed in order to modify the original behaviour of the decorated section.

For instance, think we want to make a new kind of section who prints the date before the name. We know that the `getLabel():String` method has been redefined in SectionDecorator in the following way:

```java
@override
public String getLabel() {
    return component.getLabel();
}
```
Now, we want to create the new section, we want to make this section to behave like the original one excepting that when we ask for the name we want the name and date, so we only need to redefine the `getLabel():String` method. The class could be like this:

```java
public class DateSectionDecorator extends SectionDecorator {
    public DateSectionDecorator(SectionAbstract component) {
        super(component);
    }

    @Override
    public String getLabel() {
        Date date = new Date();
        return "["+date+" ]" + component.getLabel();
    }
}
```

As you can see in the given example, it would be easy to create a new section that modifies the appearance or behaviour of the original one.

6.6.4. SectionLinkSymbolic
This section is a decorator. It is just a section that has a different name than the original one. It is useful for example, when a section could be placed in two (or more) different places in the section's menu, but we do not want to make to different sections.

For instance, if we want to place a section of a new car "Mercedes XX", we can added into the section "Car models" directly, and in the sections "News" as a SectionLinkSymbolic with the name "Last Mercedes".

Changes in one place will affect the other.

6.6.5. SimpleComposite
It is the simplest way for creating a section that has other subsections. It will just have a SimpleCompositePage associated, this page, will display a list of subsections of the SimpleComposite.

6.6.6. CustomComposite
This kind of composite allows you to set up the section with any custom page. It is used for example for setting up the home page as the content of the root of the sections in the MenuPanel.

6.6.7. Create your own type
This section hierarchy is very flexible and allows you to create new kinds of sections, new decorators and new composites.

We can make new sections types and they will work in the system as long as they extend the AbstractSection class.

6.7. Using image bundles
As we saw in the chapter 5.1.6 Image bundles (page 23), GWT have a mechanism for creating image bundles.

Every class where we want to use an image bundle is having an inner interface that extends ImageBundle. This interface is where we specify the images that we need for each panel.

```java
public interface Images extends ImageBundle {
}
```
The `AbstractImagePrototype` is a GWT class for a lightweight representation of an image. The two main things you can do with it is the `getHTML()` method that returns the HTML representation of the image, and the `createImage()` that returns a `Image`, that extends directly from `Widget`, this means that we could insert that image in any place where a widget is expected.

In the main class of the application (`PublicationSystem`), we create another inner interface that gathers all the interfaces.

```java
}
```

Once defined the interfaces, we can create our instance of this interface using the GWT mechanisms:

```java
private static final Images images = GWT.create(Images.class);
```

This image bundles are used for example for the sections icon. The next code creates a simple panel and adds the home page icon into it:

```java
SimplePanel outer = new SimplePanel();
AbstractImagePrototype homeImg = images.MenuPanel_home();
outer.add(homeImg.createImage());
```

### 6.8. Adding a new form

The form system was explained in chapter 4.7 Form system (page 18).

One easy way of creating a form is by extending the class `FormView`.

```java
public class MyForm extends FormView {
    public MyForm() {
        super("My form");
        (** HERE THE CODE **) 
    }
}
```

So, this will create a simple form called "My form". But we still need to add the entries and elements from the form.

Let's say we want to record name and age of a user, and we only accept people that were born between 1970 and 1990. We create an entry for each data we want to record:

```java
StringEntry name = new StringEntry("Name", "name", null);
IntEntry birthYear = new IntEntry("Birthday year", "birthYear", null);
```

Then we set up the restrictions, these are:

- Name and year are mandatory
- Year must be between 1970 and 1990

```java
name.setRequired(true);
birthYear.setRequired(true);
birthYear.addConstraint(new IntBetweenConstraint(1970, 1990));
```

Now we have to create the elements for input the data. Since we want to create the typical text field that has a label at the left, we first create the `TextBoxElement`, and then we wrap it in a `LabelFormDecorator`

```java
FormElement nameElement =
    new LabelFormDecorator(new TextElement(name));
```
FormElement birthYearElement =
   new LabelFormDecorator(new TextBoxElement(birthYear));

This could be done easily using a helping class called FormFactory that has among its methods makeSimpleText(AbstractFormEntry):FormElement.

We already have the elements of the form, so we just have to add them to the form:
addElement(nameElement)
addElement(birthYearElement)

6.9. Adding and editing publication forms

The forms for creating new publications are created as it was described in the last section. It is necessary just to modify the corresponding form class. All of these classes are in the package:
se.vxu.msi.cs.client.pages.publications;

There is a factory class with the duty of create all the forms. This class is also in this package. The factory class is called PublicationFactory. This factory class have some methods that allow the different parts of the application to know which publications can be created. Each publication has associated a number; this number is the "publication kind" (pubart_id in the database, see Appendix B for more information about the database fields). This "publication kind" has a relationship with the bibTex standard that is a standard for publishing electronic documents. The bibTex standard is in Appendix D (page 61).

This factory class offers methods or getting the publication id from the name of the publication and the opposite. It also allow to get the form (FormView) from the number of "publication kind":
public static String idToString(int id)
public static String idToBibtex(int id)
public static int bibtexToId(String bibtex)
public static FormView getPublication(int typePublication)

For instance, if we want to add the publication type, "New publication" that has "newpublication" as a bibTex id, and "N" as the "publication kind" number, we modify PublicationFactory in the following way:

- Modify the static String[][] variable publications, for adding in the element number N the information:
  final static String[][] publications={
   { "Licenciate thesis", "phdthesis"},
   { "Book", "book"},
   { "Conference Article", "conference"},
   ...after N elements
   { "New publication", "newpublication"},
   ...}

- Create a form as it was explained in the section 6.8. And allow the factory to create this form by modifying the "getPublication" method for returning the new form.

public static FormView getPublication(int typePublication){
   switch(typePublication){
      case 1: // Licenciate thesis
         return new LicenciateThesis();
      case 2: // Book
         return new Book();
      ... after N cases
      case N: // New publication
         return new NewPublication();
      ...
6.10. Using the internationalization

GWT support internationalization as is explained in chapter 5.1.7 Support for internationalization (page 23).

The languages that are supported by the application are specified in the file PublicationSystem.gwt.xml placed in the root of the source code. For adding a new language we just add a new property:

```xml
<extend-property name="locale" values="se"/>
```

The last line says that the application is available in Swedish.

There are two kinds of messages for internationalization:

- **Constants**: They are invariable strings, typically used for the user interface and messages that does not have any parameter.
  - For instance: The "Close window" tag in a button.

- **Messages**: They are messages that have, at least one parameter. Messages are used for giving variable messages,
  - For instance: "The field {fieldName} must be a number."

For specifying both of them we use a property file, they are all in the package se.vxu.msi.cs.client.properties.

6.10.1. Constants

Let's begin by adding a new constant to the application. We edit the file LocalConstants.properties for the default language, English in this case. We edit LocalConstants_se.properties for Swedish. Notice that the files name for other languages are formed by adding just the code of the country and it should match with the value specified in the PublicationSystem.gwt.xml file. For adding the close window message we add:

```properties
close: Close window
```

Then, we need to execute a script that will create some Java classes needed in order to use these constants in the application. This script is:

- **LocalConstants-i18n.launch**: If we execute it from eclipse
- **LocalConstants-i18n**: If we execute it from a UNIX terminal

Once executed, we can use the String in any part of the client side. For example:

```java
String closeString = Constants.localConstants.close();
```

6.10.2. Messages

In an analogue way to the constants, we edit the files:

```properties
fieldNotNumber: The field {0} must be a number
```

Then, we execute the script:

- **LocalMessages-i18n.launch**: If we execute it from eclipse
- **LocalMessages-i18n**: If we execute it from a UNIX terminal

Once executed, we can use the String in any part of the client side. For example:
String fieldNotNumberString = Constants.localMessages.
fieldNotNumber("First name");

6.11. Using the database

For making connections to the database it have been done a servlet as it was explained
in the chapter 5.2 Database management (page 23).
The class in charge of making the remote procedure call (RPC) is
PublicationSystem. This class offer four methods for using the database:

• public void connectDB(AsyncCallback callbackConnect)
  o Connects to the database
• public void disconnectDB()
  o Disconnects from the database
• public void query(String sql, AsyncCallback<DatabaseResult>
callbackQuery)
  o Execute a SELECT instruction in the database
• public void update(String sql, AsyncCallback<Integer>
callbackUpdate)
  o Execute an INSERT, DELETE or UPDATE instruction in the database.

One of the arguments for these methods is an AsyncCallback<T>, this class is the
class needed for the call-back in the asynchronous call. It is an interface with two
methods:

• void onFailure(Throwable caught);
  o Is called when something goes wrong during the RPC, or if the server
    sends an DatabaseException.
• void onSuccess(T result);
  o Called when the method in the server side has finished processing the call
    and has the result ready.
  o In the query method, the type of this parameter is DatabaseResult, this is
    a class that wraps all the rows and the headers of a SQL result.
  o The update method has Integer as a parameter. This integer will be the
    number of affected rows.

6.12. Adding a new service

The application can have more services in the future. There are some steps to follow for
creating a service. Let’s explain it with an example:

We want a service for getting the time from the server.

6.12.1. Creating the interface of the service

We create the interface of the service. This will be the agreement between client side
and server side. The interface must implement RemoteService and specify one
annotation with the name of the service (time in this case).

```java
@RemoteServiceRelativePath("time")
public interface TimeServiceInterface extends RemoteService {
    public String getTime();
}
```
6.12.2. Creating the asynchronous interface

For create the asynchronous version of the interface we have to meet the following requirements:

- It has the same name ended in "Async"
- The methods return nothing (void), instead we add an AsyncCallback<T> to the arguments, being T the result type of the synchronous version.

```java
public interface TimeServiceInterfaceAsync{
    public void getTime(AsyncCallback<String> callback);
}
```

6.12.3. Implementing the interface

We implement the interface in the server side; this is in the package `se.vxu.msi.cs.server`.

```java
public class TimeService implements TimeServiceInterface{
    public String getTime() {
        return (new Date()).toString();
    }
}
```

6.12.4. Adding the service

We need to set up our service in order to use it. This configuration is done in the file `./war/WEB-INF/web.xml` where we add the following code:

```xml
<web-app>
  <servlet>
    <servlet-name>TimeService</servlet-name>
    <servlet-class>se.vxu.msi.cs.server.TimeService</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TimeService</servlet-name>
    <url-pattern>/publicationsystem/time</url-pattern>
  </servlet-mapping>
</web-app>
```

We have to make sure, that we specify the same name of service than in the annotation of the interface of the service, in this case it was `time`.

6.12.5. Creating the service

Now we are ready for getting an instance of the service from the client:

```java
TimeServiceInterfaceAsync timeService =
    (TimeServiceInterfaceAsync) GWT.create(LoginServiceInterface.class);
```

Then we need a call-back class.

```java
AsyncCallback<String> callbackTime=new AsyncCallback<String>(){
    public void onSuccess(String result) {
        Window.alert("The time is "+result);
    }
    public void onFailure(Throwable caught) {
        Window.alert("Error getting the time:" + caught.getMessage());
    }
};
```

And we make the asynchronous call:

```java
timeService.getTime(callbackTime);
```
6.13. Change the login system
The login system is implemented as a service, for implementing a new login system it is necessary to modify the class LoginService and give a body to the method:

```java
public boolean login(String user, String password);
```

6.14. Create a new form strategy
It is possible to create a new strategy for processing a form; this is very powerful because it allows you to make different data treatments for the same form.

For doing this, you extend the abstract class StrategyForm<T>. The main methods of this class are:

```java
// Concrete methods
public void addCallBack(StrategyCallBack<T> callback) { ... }
public void notifyCallBack(T result) { ... }

// Abstract methods
public abstract void preTask() throws StrategyFormException;
public abstract void task(ArrayList<AbstractFormEntry> entry)
    throws StrategyFormException;
public abstract void postTask() throws StrategyFormException;
```

For example, if you want to do a StrategyForm, that process the form and generate a Page, we create a class that extends StrategyForm<Page>. This class will implement these abstract methods.

In these methods, the strategy will process all the entries for creating a Page. After the postTask() it will do a notifyCallBack(Page result), so all the call-backs that have been added to the strategy form by addCallBack(StrategyCallBack<Page> callback) will receive the result.

Another example of a strategy form is the InsertStrategy, which insert a form in the database; it implements the class StrategyForm<Integer> because the result is the number of rows affected by the insertion.
7. Conclusion

In this chapter it will be presented some conclusions got from the experience of this project as well as a comparison of the old system with the new one.

7.1. Old system and new system

The old system had lots of design problems what were making the page difficult to maintain and the information was not reliable.

For explaining in which ways the actual solution improves the old system it will be presented all the problems and how the new application solve them.

7.1.1. Web application

The old system has a traditional PHP web application, the problem with this application was explained in chapter 2.1 Web applications (page 2).

However, the new application uses the last technologies like asynchronous calls, image bundles, multi-browser and multilingual, client side programmed, object oriented, and scalable. For more information on this see chapter 5.1 GWT - Google Web Toolkit (page 22).

7.1.2. Database design

The database is not very well designed and is not robust. If you are not very careful with the data you insert you may introduce incoherent information on it. As it was proposed for future improvements in the chapter 8.5 Improvement of the database (page 45), it would be a very good idea to redesign the database, however this application is design robust for supplement this luck of reliability.

The new system, allows you to build reliable forms for sending coherent information into the database. When you specify the logic of the data by specifying data types, restrictions and relations, you are making sure the data is reliable, so it can be enter in the database.

7.1.3. User interface for publishing documents

For publishing documents in the old system there was not user interface at all, the user was forced to deal directly with the database using the phpMyAdmin, that is a tool build for administration proposes, never for being used by the end-user web as a interface for publishing documents. More over, since the database is not documented, is not well design and the attribute names are written in different languages it was confusing for a user to know the meaning of each attribute and which attributes need to be fill and which not for a particular kind of document.

Even if the user is experimented, and supposing that he perfectly understand the database and know all the attributes that a particular document needs, he even have to remember all the foreign keys for linking a document with its author, and remember the identification for all different kinds of possible documents.

The new interface for publishing a document offer a list of different publication documents. When you select one, only the fields that are relevant to this publication type are displayed. The system will also know which of them are required and which are not.

Apart from all these advantaged, the user does not have to memorize the identification numbers of types of publications or the ones of the authors, this numbers
are hidden to the user, so he select for instance a "Technical report" and not the number 7, or they select the name of the author, and not the key of the table authors.

Figure 7.1 - Old publication interface

Figure 7.2 - New publication interface

7.1.4. Searches
The old system only allow to make simple searches, and the result is difficult to browse. The new system allows doing simple and complex searches in the documents from the RICS database and in Diva. It is also much more easy to use, see Making a search (page 53).

7.1.5. Login
Due to it was access the database directly: the person publishing the document should have a user in it. What happen if there is one hundred publisher? or, what happen if it became necessary to give the student the chance of publishing documents?

With the new approach, the login system allows to make an implementation for the login method allowing using an old login system currently in use. This makes easier to control the access to the database and to have the login system centralized.

Besides, it allows you to build future improvements, for example a control system that monitors the access to the application. In the old system that was impossible, and if a table is dropped and more than one user can access it, you will not find the responsible person of the data loss.
8. Further improvements

In this chapter, it will be exposed some ideas for further improvements of the systems. The improvements proposed are oriented to make the application easier to maintain, simpler to understand by a non-technician, more user friendly, and to maintain more reliable information on it.

8.1. Motivation

As it was showed in the chapter 6 Maintenance (page 29), it is very easy to maintain the application. For adding new sections, forms or modifying the layout you only need to make a few changes in some Java classes.

The problem is that the content of the page is depending on a programming language. An average user does not have to know how to program. He must not deal with the source code, more over; it would be a security problem to let people to manage directly the code.

For this reason we need an intermediate language, easy to understand, that allows to specify all these contents and structures within the application.

The most worldwide extended format for these propose is XML, it is designed for being easy to read for machines and people, so is the perfect candidate.

The use of this format for specifying the content would be a really step forward to make the application easier to maintain. However, although is easy to read for humans, XML require to know about the format and about the tags you can use. Instead it would be a really good idea to give the user a graphical tool, so it can configure this XML file by drag and drop components in an intuitive interface.

Apart from the maintenance problem, it was shown in section 3.1 RICS (page 8) a potential problem with the database. By adding a control to the data that is inserted in the application in part solved this, but this is not enough, the data control should be in the database level no in the application one. Besides, there were other design problems that should be solved.

8.2. Configuration of the web with XML files

Although the application is easy to configure, it will be more elegant and easy to configure if these configuration files are written in XML.

By using XML format as the configuration of the application we get some advantages:

- **Easier to modify**: A plain text editor is enough.
- **No recompilation needed**: The current way of configuring the web, adding content, functionalities, etc. is by hard coding in Java. With an XML file we would not need to recompile the application because is dynamically read.
- **Possibility of a user interface**: Another consequence of "no recompilation needed" is that it could be done a user interface (for example a web interface) for modifying this XML files.
- **Other applications could use it**: Since XML is an open standard, and is not engaged with other technologies, it could be use in other systems.
- **Separation of logic and functionality**: It would separate the configuration and content from the interface and functionality, that means that it would be
easier to modify both of them separately making the system easier to update and upgrade.

8.2.1. Specification of the sections

It will be interesting to define the sections in a file similar to the following one:

```xml
<sections>
  <section type="CustomComposite">
    <name>Home</name>
    <historyName>Home</historyName>
    <image>images/home.png</image>
    <content class="se.vxu.msi.cs.client.pages.Home" />
  </section>
  <section type="SectionLeaf">
    <name>Search</name>
    <historyName>Search</historyName>
    <image>images/search.png</image>
    <content class="se.vxu.msi.cs.client.pages.Search" />
  </section>
  ...
  ...
</sections>
```

8.2.2. Forms

In the same way as the sections, it will be very useful to create a form using XML, it will allow to build in the future tools for building forms automatically.

For instance, lets give an example by redoing the form e of the example of the chapter 6.8 Adding a new form (page 35).

```xml
<form name="My form">
  <element type="LabelFormDecorator">
    <element type="TextBoxElement">
      <entry type="StringEntry" required="true">
        <name>Name</name>
        <id>name</id>
      </entry>
    </element>
  </element>
  <element type="LabelFormDecorator">
    <element type="TextBoxElement">
      <entry type="IntEntry" required="true">
        <name>Birth Year</name>
        <id>birthYear</id>
        <constraint type="IntBetweenConstraint">
          <min>1970</min>
          <max>1990</max>
        </constraint>
      </entry>
    </element>
  </element>
</form>
```

8.2.3. Content

It is a very good idea to create the class `XMLPage` that extends from `Page`. It would be a page that its content would be stored in a XML document. It should still support
internationalization, so this dynamic loading will choose the XML document according to the language of the user.

The structure of this document could be like the `<body>` tag in a XHTML document.

8.3. Add a control panel and WYSIWYG editors

Once the application is using XML for the configurations, it could be easily built a web interface such a control panel for editing the preferences of the project.

This control panel could also be integrated as part of the application, for example by creating the roles for user and giving them permission for editing or not the application.

WYSIWIG stands for "What You See Is What You Get" are a kind of editors, in this case for HTML, that allows you to write HTML code without knowing HTML, just using an interface similar to the word processors one.

This would make even easier to add content to the web page.

8.4. Using drag & drop technologies

GWT have libraries for implementing drag and drop. This technology makes it very easy for the user to do complex tasks, especially for a non-technician user.

A possible use of this would be, making easier the creation of new forms. The user would be able to build a form in minutes just by dragging and dropping form elements.

Other possible use is for configure the page layout, it could be added to the control panel. So by simply drag and drop you could move or add new the panels to the application allowing the average user to configure it.

8.5. Improvement of the database

The design of the database is not very good, it does not respect the database principles, this produces the information to be redundant having a waste of space, and the database allows entering inconsistent data.

It does not respect the Boyce-Codd Normal Forms. I would be a good idea to make an exercise of rethink and build a new design. For example, the publications are all written in the same table, but the fields that they have are very different, it seems to be a good idea to use a Enhanced Entity-Relation model, for modelling using subclasses and super class for a general publication and its particular types of it. This will make the database more reliable only by adding a new indirection level.

It will be very useful to create some restrictions over the data, so the data is checked before adding to the database.

The actual database has names that are not very clear, they are even written in different languages, and there is not documentation about the actual design, name and meaning of the attributes, or restrictions in the data.

A good design will make easier to improve the application for creating new services and will make the system more reliable.

8.6. Link between RICS and Diva

Unless Diva release an API for publishing a document the system is not allowed to insert new documents in the Diva database.

For making this problem smaller, it could be added a field in the database with the unique id of the document in Diva, so when a document is displayed, it would offer a link to see the same document in the Diva system. This way, there will be a little control between the two possible versions of the same document.
Literature

All materials in these sections are WebPages because all the official documentation and information are in these sites. The last visit to all these sites was in June 2009.

1. **RICS**: Home page of the Research in Computer Sciences of Växjö University.
   - **Home**: [http://cs.msi.vxu.se/](http://cs.msi.vxu.se/)

2. **Diva System**: Publication system for Universities in Scandinavia.
   - **About**: [http://www.diva-portal.org/about.xsql](http://www.diva-portal.org/about.xsql)

3. **GWT - Google Web Toolkit**: Webpage of the technology used to implement the design of the publication system.
   - **Home**: [http://code.google.com/webtoolkit/](http://code.google.com/webtoolkit/)

4. **MySQL**: Homepage for the database management system used for recording the publications.
   - **Home**: [http://www.mysql.com/](http://www.mysql.com/)
   - More information on the database fields in Appendix B (page 47)

5. **Bibtex**: Homepage of the standard for the digital publication of documents followed on this thesis.
   - **Home**: [http://www.bibtex.org/](http://www.bibtex.org/)
   - More information about the standard in Appendix D (page 61).
Appendix A: User manual
This appendix explains how to do the main tasks in maintaining the RICS system.

1. User interface

The application interface is like the one in Figure 8.1:

![User Interface Image]

Figure 8.1 - User interface

The main parts of the user interface are:

- **Sections menu**: Shows the visible sections
- **Language**: Allows changing the language of the interface.
- **Current section**: Show the current section. If it is a subsection shows the path of the section (i.e. home > Projects > On going)
- **Login**: Allows a user to log in.

![User Interface Parts Image]

Figure 8.2 - Parts of the user interface
2. Adding a new publication

For adding a new publication is simple. First we log in, due to the publication area is a private area. Then we select "Public document" in the section's menu.

Then you select the kind of document you want to publish.

For example, we select "Book" and automatically the system will display the form with the fields that should be recorded, the fields that are optional and it will let you add authors.

The first tab shows the required fields. Make sure you fill all of them or the system will not let you add the publication.

The optional tab, show just the fields that are not required.
One of the optional fields allows you to upload a file up to 50 Mb. For uploading a file make click on "Upload File", it will display a form for uploading it. This files are the electronic version of the publication.

Once the file is selected, you press "Upload file" for starting the upload.

A percentage with the completion of the task will be displayed.

When the upload is finished, the system shows the uploaded file and lets the user to delete it.

The "add authors" tabs, allow you to add authors to the publication. For adding a new person press "Add new" and the application will show a new row with a text box. Type the name or surname and a menu will show all the people that match with the string in the textbox.

3. View the publication's information

For viewing the publications, select "List of publications". It will show all the publications ordered by year.
The result is paginated in tables of 15 rows

For changing to another page, use the controls in the button part of the table. The button "Jump to" allows jumping to a concrete page. Once you click in "Jump to" a pop up window will ask the page to jump. Insert the page and press "enter" to go to the page, or "esc" for closing the pop up window (click out of it have the same effect).

For viewing the info of a publication click on it.
4. Edit, add authors and delete publication

For editing a publication, first you need to log in because it is a private area. A non-logged in user can only see the tab of the info, but when you log in, three new tabs appear in when you view the publication information.

The edit tab allows to modify the information.
The optional fields are hidden. Click on "Optional" for showing them.

"Add authors" allows adding, removing or modifying the range, for an author of a publication.

The tab "Delete" allows deleting a publication.
5. Making a search

The system allows to search for publications in three different ways; simple search, advanced search, diva search.

For making a search, select "Search" in the section's menu.

a. Simple search

For making a simple search select the first tab. It has a simple text box; you can enter a title or part of it.

If you enter more than one word, the system will understand that the title must have all of the words in the title.
Once you make the search, a list of results is displayed. The user can click on any element of the list to check the details or even modify the selected publication in case the user has permission.

b. **Advanced search**

For making an advanced search click on the "advanced search" tab.

This search allows you to filter the result using different criteria. By default, all criteria fields are hidden, so if you want to filter the result using, for example, "Year of publication" you click on its label, and the fields will show up.

- **Title**: Filter the result using the title name.
- **Type of publication**: Filter the result using the kind of publication.
Year of publication: Filter the result using the year. It allows to specify:

- A concrete year.
- Year after a given one.
- Year before a given one.
- Year between two years.

Author: Filter the result using by author. It allows to select several authors and look for the publications where have participated:

- All of them
- Any of them

Order by: Order the result by one field.

An example of an advanced search would be: "Conference articles of Morgan Ericsson in the year 2005".
c. Diva search

For making a simple search in diva, just fill the field and press "Search"

For example, by looking for XML we get:
Appendix B: Database fields

The application records the information needed in a database, this appendix shows the different fields of the table of publications.

The RICS database is implemented using MySQL, you can get more information on MySQL in reference [4] of literature section.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data base name</th>
<th>Kind of variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publication id</td>
<td>*publikations_id</td>
<td>int(11)</td>
<td>Unique identification of the publication.</td>
</tr>
<tr>
<td>Title</td>
<td>title</td>
<td>text</td>
<td>The work's title.</td>
</tr>
<tr>
<td>Publication kind</td>
<td>pubart_id</td>
<td>int(11)</td>
<td>See Appendix C (page 60) for detailed information.</td>
</tr>
<tr>
<td>Title of the book</td>
<td>booktitle</td>
<td>varchar(200)</td>
<td>Title of a book, part of which is being cited.</td>
</tr>
<tr>
<td>Year</td>
<td>year</td>
<td>int(11)</td>
<td>The year of publication or, for an unpublished work, the year it was written. This field's text should contain only numerals.</td>
</tr>
<tr>
<td>Month</td>
<td>month</td>
<td>varchar(10)</td>
<td>The month in which the work was published or, for an unpublished work, in which it was written.</td>
</tr>
<tr>
<td>Organization</td>
<td>organization</td>
<td>varchar(200)</td>
<td>The organization sponsoring a conference.</td>
</tr>
<tr>
<td>Publisher</td>
<td>publisher</td>
<td>varchar(200)</td>
<td>The publisher's name.</td>
</tr>
<tr>
<td>Volume</td>
<td>volume</td>
<td>varchar(20)</td>
<td>The volume of a journal or multivolume book work.</td>
</tr>
<tr>
<td>Number</td>
<td>number</td>
<td>varchar(20)</td>
<td>The number of a journal, magazine, or technical report. An issue of a journal or magazine, it is usually identified by its volume and number; the organization that issues a technical report usually gives it a number.</td>
</tr>
<tr>
<td>Pages</td>
<td>pages</td>
<td>varchar(20)</td>
<td>A page number or range of numbers such as &quot;42--111&quot;; you may also have several of these, separating them with commas: &quot;7,41,73--97&quot;. The standard styles convert a single dash to a double.</td>
</tr>
<tr>
<td>Chapter</td>
<td>chapter</td>
<td>varchar(20)</td>
<td>Chapter number</td>
</tr>
<tr>
<td>Journal</td>
<td>journal</td>
<td>varchar(200)</td>
<td>A journal name. Abbreviations are provided for many journals.</td>
</tr>
<tr>
<td>Abstract (Swedish)</td>
<td>abstract_d</td>
<td>text</td>
<td>Abstract summary about its content in Swedish.</td>
</tr>
<tr>
<td><strong>Abstract</strong> (English)</td>
<td>abstract_e</td>
<td>text</td>
<td>Abstract summary about its content in English.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><strong>External author</strong></td>
<td>ext_author</td>
<td>varchar(200)</td>
<td>The name(s) of the author(s).</td>
</tr>
<tr>
<td><strong>Editor</strong></td>
<td>editor</td>
<td>varchar(200)</td>
<td>Name(s) of editor(s). If there is also an &quot;author&quot; field, then the &quot;editor&quot; field gives the editor of the book or collection in which the reference appears.</td>
</tr>
<tr>
<td><strong>School</strong></td>
<td>school</td>
<td>varchar(200)</td>
<td>The name of the school where a thesis was written.</td>
</tr>
<tr>
<td><strong>Institution</strong></td>
<td>institution</td>
<td>varchar(200)</td>
<td>The institution that published the work.</td>
</tr>
<tr>
<td><strong>howpublished</strong></td>
<td>howpublished</td>
<td>varchar(200)</td>
<td>How something strange has been published.</td>
</tr>
<tr>
<td><strong>Series</strong></td>
<td>series</td>
<td>varchar(200)</td>
<td>The name of a series or set of books. When citing an entire book, the &quot;title&quot; field gives its title and an optional &quot;series&quot; field gives the name of a series in which the book is published.</td>
</tr>
<tr>
<td><strong>Edition</strong></td>
<td>edition</td>
<td>varchar(50)</td>
<td>The edition of a book - for example, &quot;second&quot;.</td>
</tr>
<tr>
<td><strong>Bibkey code</strong></td>
<td>bibkey</td>
<td>varchar(50)</td>
<td>Unique entry for generating a bibtex entry automatically. By convention, we use the initials of the authors, publication year, and “a, b, c …” if necessary, i.e. LL2004b for the second publication of Löwe and Lundberg in 2004.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or Used for alphabetizing and creating a label when the &quot;author&quot; and &quot;editor&quot; fields are missing. This field should not be confused with the citation key that appears in the \cite command and at the beginning of the entry.</td>
</tr>
<tr>
<td><strong>Address</strong></td>
<td>address</td>
<td>text</td>
<td>Publisher's address. For major publishing houses, just the city is given. For small publishers, you can help the reader by giving the complete address.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>note</td>
<td>text</td>
<td>Any additional information that can help the reader.</td>
</tr>
<tr>
<td><strong>URL of the pdf document</strong></td>
<td>url</td>
<td>varchar(200)</td>
<td>Pdf file of the publication for download. It must start with &quot;/papers/&quot; followed by the filename. By convention, we use bibkey + &quot;pdf&quot;</td>
</tr>
<tr>
<td>Special</td>
<td>special</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>Intern</td>
<td>intern</td>
<td>enum('intern', 'extern')</td>
<td>- Intern - Extern</td>
</tr>
</tbody>
</table>

(*) primary key
(underline): mandatory field (not null)
## Appendix C: Publication types

<table>
<thead>
<tr>
<th>Id number (pub_art)</th>
<th>Type (English)</th>
<th>Type (Swedish)</th>
<th>BibTex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Licentiate Thesis</td>
<td>Licentiatavhandling</td>
<td>phdthesis</td>
</tr>
<tr>
<td>2</td>
<td>Book</td>
<td>Bok</td>
<td>book</td>
</tr>
<tr>
<td>3</td>
<td>Conference Article</td>
<td>Konferensartikel</td>
<td>conference</td>
</tr>
<tr>
<td>4</td>
<td>Journal Article</td>
<td>Tidsskriftsartikel</td>
<td>article</td>
</tr>
<tr>
<td>5</td>
<td>Booklet</td>
<td>Häfte</td>
<td>booklet</td>
</tr>
<tr>
<td>6</td>
<td>PhD. thesis</td>
<td>Avhandling</td>
<td>phdthesis</td>
</tr>
<tr>
<td>7</td>
<td>Technical Report</td>
<td>Teknisk rapport</td>
<td>techreport</td>
</tr>
<tr>
<td>8</td>
<td>Poster</td>
<td>Poster</td>
<td>misc</td>
</tr>
<tr>
<td>9</td>
<td>Unpublished</td>
<td>Opubliserat</td>
<td>unpublished</td>
</tr>
<tr>
<td>10</td>
<td>Masters Thesis</td>
<td>Magisteruppsats</td>
<td>mastersthesis</td>
</tr>
<tr>
<td>11</td>
<td>Thesis</td>
<td>Uppsats</td>
<td>misc</td>
</tr>
<tr>
<td>12</td>
<td>Talk</td>
<td>Föredrag</td>
<td>misc</td>
</tr>
<tr>
<td>13</td>
<td>Book Chapter</td>
<td>Book Chapter</td>
<td>inbook</td>
</tr>
<tr>
<td>14</td>
<td>Book Chapter (with title)</td>
<td>Book Chapter (with title)</td>
<td>incollection</td>
</tr>
</tbody>
</table>
Appendix D - BibTeX standard

In this appendix it will be shown the Bibtex standard and the relation of the Bibtext fields with the database publication documents. The number in parenthesis corresponds to the ID number (\textit{pub\_art} field of the database).

For more information on the BibTeX standard check the reference [5] in the literature section.

1. i) Phdthesis entry (1, 6)
2. ii) Book entry (2)
3. iii) Conference entry (3)
4. iv) Article entry (4)
5. v) Booklet entry (5)
6. vi) Techreport entry (7)
7. vii) Misc entry (8, 11 and 12)
8. viii) Unpublished entry (9)
9. ix) Mastersthesis entry (10)
10. x) Inbook entry (13)
11. xi) Incollection entry (14)
12. xii) Inproceedings entry
13. xiii) Manual entry
14.xiv) Proceedings entry

i. Phdthesis entry

A PhD thesis.

Format:
\begin{verbatim}
@PHDTHESIS{citation_key, 
required_fields [, optional_fields] }
\end{verbatim}

Required fields:
\begin{verbatim}
author, title, school, year
\end{verbatim}

Optional fields:
\begin{verbatim}
address, month, note, key
\end{verbatim}

ii. Book entry

A book with an explicit publisher.

Format:
\begin{verbatim}
@BOOK{citation_key, 
required_fields [, optional_fields] }
\end{verbatim}

Required fields:
\begin{verbatim}
author or editor, title, publisher, year
\end{verbatim}
Optional fields:
volume, series, address, edition, month, note, key

### iii. Conference entry

An article in the proceedings of a conference. This entry is identical to the 'inproceedings' entry and is included for compatibility with another text formatting system.

Format:
```
@CONFERENCE{citation_key,
    required_fields [, optional_fields] }
```

Required fields:
author, title, booktitle, year

Optional fields:
editor, pages, organization, publisher, address, month, note, key

### iv. Article entry

An article from a journal or magazine.

Format:
```
@ARTICLE{citation_key, required_fields
    [, optional_fields] }
```

Required fields:
author, title, journal, year

Optional fields:
volume, number, pages, month, note, key

### v. Booklet entry

A work that is printed and bound, but without a named publisher or sponsoring institution.

Format:
```
@BOOKLET{citation_key,
    required_fields [, optional_fields] }
```

Required fields:
title

Optional fields:
author, howpublished, address, month, year, note, key
vi. Techreport entry
A report published by a school or other institution, usually numbered within a series.

Format:
@TECHREPORT{citation_key,
required_fields [, optional_fields] }

Required fields:
author, title, institution, year

Optional fields:
type, number, address, month, note, key

vii. Misc entry
Use this type when nothing else seems appropriate.

Format:
@MISC{citation_key,
required_fields [, optional_fields] }

Required fields: none

Optional fields:
author, title, howpublished, month, year, note, key

viii. Unpublished entry
A document with an author and title, but not formally published.

Format:
@UNPUBLISHED{citation_key,
required_fields [, optional_fields] }

Required fields:
author, title, note

Optional fields:
month, year, key

ix. Mastersthesis entry
A Master's thesis.

Format:
@MASTERSTHESIS{citation_key,
required_fields [, optional_fields] }

Required fields:
author, title, school, year
Optional fields:
  address, month, note, key

x. Inbook entry

A part of a book, which may be a chapter and/or a range of pages.

Format:
  @INBOOK{citation_key, required_fields [, optional_fields] }

Required fields:
  author or editor, title, chapter and/or pages, publisher, year

Optional fields:
  volume, series, address, edition, month, note, key

xi. Incollection entry

A part of a book with its own title.

Format:
  @INCOLLECTION{citation_key, required_fields [, optional_fields] }

Required fields:
  author, title, booktitle, year

Optional fields:
  editor, pages, organization, publisher, address, month, note, key

xii. Inproceedings entry

An article in the proceedings of a conference.

Format:
  @INPROCEEDINGS{citation_key, required_fields [, optional_fields] }

Required fields:
  author, title, booktitle, year

Optional fields:
  editor, pages, organization, publisher, address, month, note, key

xiii. Manual entry

Technical documentation.
xiv. Proceedings entry

The proceedings of a conference.

Format:

@PROCEEDINGS{citation_key,
    required_fields [, optional_fields] }

Required fields:
    title, year

Optional fields:
    editor, publisher, organization, address, month, note, key