THE FURN

ORGANIZING FURNITURE THROUGHOUT THE WEB

Authors:
R.R.S. VAN LON
J.P.M. VAN DER MEER

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
June 6, 2008

Company:
Floorplanner B.V.
Ir. J. BEKKERS

Supervisor:
Dr. A.H.J. OOMES
Ir. M. Sepers

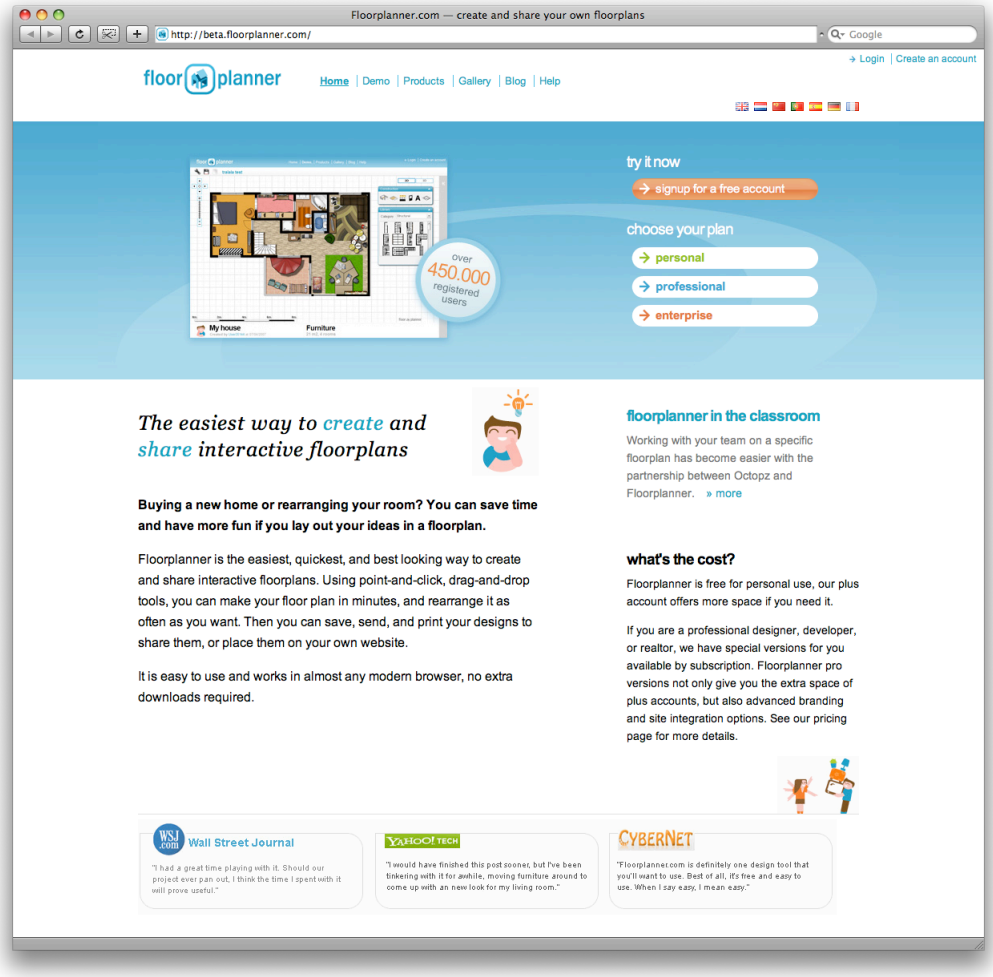

Figure 1: Floorplanner.com

Preface

All students who finish their bachelor Media and Knowledge Engineering at Delft University of Technology are required to do a project at the end of the bachelor phase. This project takes 3 months and can be done at a company of choice. We, Rinde and Jaap, choose to do this project at Floorplanner.com B.V., a software company based in Rotterdam. The tool they have created is called Floorplanner. It is an interactive online application with which users can design their interiors, by drawing walls, rooms and adding furniture.

About Floorplanner.com B.V.

This project is an assignment of Floorplanner.com B.V.. The idea for this project was created in collaboration with Floorplanner.com B.V. CEO Jeroen Bekkers. Floorplanner.com (Figure 1) is an international website where you can design your house online. By using a very user friendly application, rooms can be drawn and furniture can be placed. Furniture can be scaled and moved by easy drag and drop operations. Floorplanner.com BV is an international oriented company, the website is available in seven languages. It has over 450.000 registered users, which is growing by around 1500 new users everyday. All these users create around 3000 new floor plans everyday.

Abstract

In this paper the project that has been done for Floorplanner.com B.V. is presented. The project, called “The Furn”, was about creating a system that automatically crawled webpages and indexes furniture information on these web pages. This information must be coupled in smart ways to elements which can be placed into Floorplanner, a tool to design the interior of a house.

To do this, the design of the system was split up into parts. The most important parts are: the Den which contains all Floorplanner elements and indexed article information and which provides an API to third-party developers. The other part is the Wolf, which crawls webpages and which also provides a user interface to users who want to add a furniture element in their floor plan. The system was first designed, after that it was implemented and finally tested by both API users and Wolf users.

Glossary

Article	An item in the den, which is a representation of a physical object that has a product page somewhere on the internet.
Crawler	A program which browses the World Wide Web in an automated manner.
Den	The database where all information about furniture is.
Design	A floorplan design (furniture, surfaces, walls).
Floorplanner	The service with which users can design their homes.
Floorplanner element	A drawing of a physical object, which can be placed into Floorplanner.
Marker	A location in the DOM tree of a page, pointing to an element which contains specific information.
Perspective	A (summarized) view on the data, from a certain perspective.
Placed element	A relation between a Floorplanner object and a Den article and some specific characteristics like color, etc.
Product	The same as article.
Wolf	Web Object Library Finder, a program which crawls the web in search for furniture.

Contents

Chapter 1

Introduction

Floorplanner is a program to let people design their interiors online. People can use the element library of Floorplanner to see 'if that couch fits'. The elements that are currently available are all drawn by Floorplanner.com BV. The library contains a lot of elements, for almost every kind of furniture a Floorplanner element exists. By changing the size and color of these elements, users are able to virtually place their own furniture on the floor plan, although they actually place a Floorplanner element.

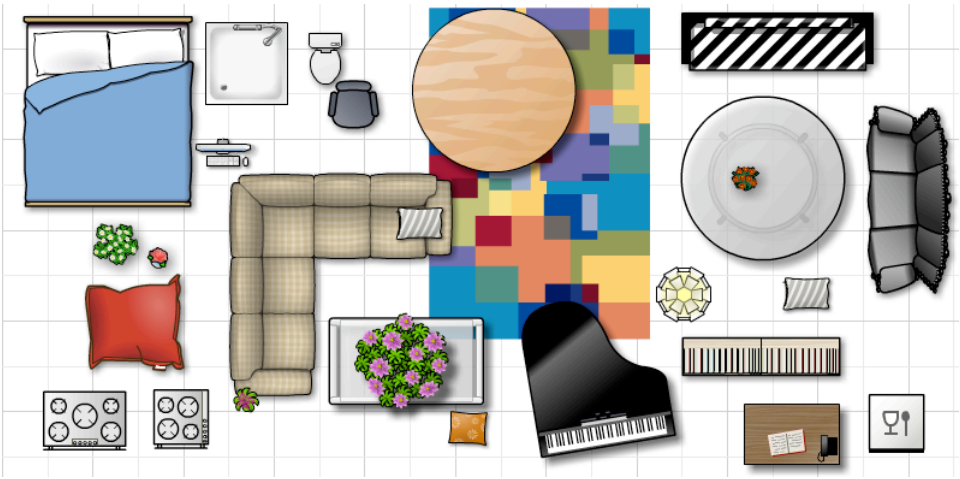

Figure 1.1: A random selection of Floorplanner elements

The idea Floorplanner.com BV has, is that users should be able to drag any image of "real world" furniture into their floor plan. This image should be automatically converted to a Floorplanner element once you drag it in. This way a user will have the feeling that they place their own furniture into Floorplanner. This also means

that users will in fact have an ‘unlimited’ set of furniture that can be placed, since every article that has been made, can be added in Floorplanner (as long as it is published on a webpage).

The advantage for Floorplanner.com BV is that it is not necessary to draw every kind of furniture as a Floorplanner element. It rather has to extend the current finite set of elements once in a while to create some elements that do not have a proper representation yet. Furthermore these elements can provide Floorplanner with a wealth of extra information, for example: popularity statistics about furniture usage can be collected.

Part I

Analysis

Chapter 2

Problem definition

The problem Floorplanner.com B.V. has an index of more than 800 furniture elements. These elements are designed by themselves and used in the online Floorplanner tool. Floorplanner.com B.V. wants a system to automatically create a mapping of furniture articles to Floorplanner elements. Furniture articles are found for example on websites. These articles must be added to a large database and indexed efficiently.

Assignment Based on the problem and in narrow cooperation with Floorplanner the following problem definition was developed:

Design a system that automatically indexes furniture found on furniture websites and couple this information in smart ways to Floorplanner elements.

2.1 Naming

When this problem definition was developed, it was clear the system was build up of four parts. Together with Floorplanner.com B.V. it was decided that a name had to be given to these parts. The overall project was named "The Furn", for organizing furniture throughout the web. The four parts were given the following names:

Den The part of the system that stores all the information about articles and furniture and that couples this information to Floorplanner is called Den.

Wolf back-end The part of the system that automatically searches furniture websites, is called Wolf back-end.

Wolf user interface The part of the system where users can add furniture from any webpage to the Wolf, which processes the page and sends it to the Den.

Floorplanner integration The part of the system that integrates the Den with the Floorplanner interface.

Chapter 3

Requirements

In this chapter the requirements of the system are presented. These are requirements formulated in cooperation with Floorplanner.com B.V. When the final system will be evaluated, it must respect these requirements.

3.1 Requirements Den

1. The Den must be able to provide an automatic mapping from Floorplanner elements to articles.

2. The system must be able to have an average request rate of less then 500 ms.

3. The system must be able to scale in size, and in request performance.

4. The system must be able to run in a distributed computing environment.

5. The system must be able to generate views that enable knowledge discovery.

3.1.1 Requirements Den API

1. The system must have an API interface.

2. The API must have access control.

3. Third party developers have to be able to access the API.

4. The API must be easy to use for third party developers.

3.2 Requirements Wolf

1. The wolf must have an automatic webpage crawler that indexes web pages.
2. The wolf must be able to determine whether an indexed web pages contains an article.
3. The wolf must remember the time it last indexed a web page, to prevent content duplication.
4. The wolf must be able to scale in size, and in performance.
5. The wolf must be deployable in a multi-server environment.

3.3 Requirements Wolf UI

1. Articles can be added using an easy-to-use user interface.
2. Articles can be added by third party developers.

3.4 Requirements Floorplanner integration

1. The articles in the Den must be showed on the Floorplanner website.
2. The articles must be easily placed in a Floorplan.
3. The user must be able to search for new articles by keyword.

Chapter 4

Planning

Based on the requirements a global planning was created. The planning in Figure 4.1 can be divided into six different partly overlapping stages. In this chapter the goal of every stage will be discussed.

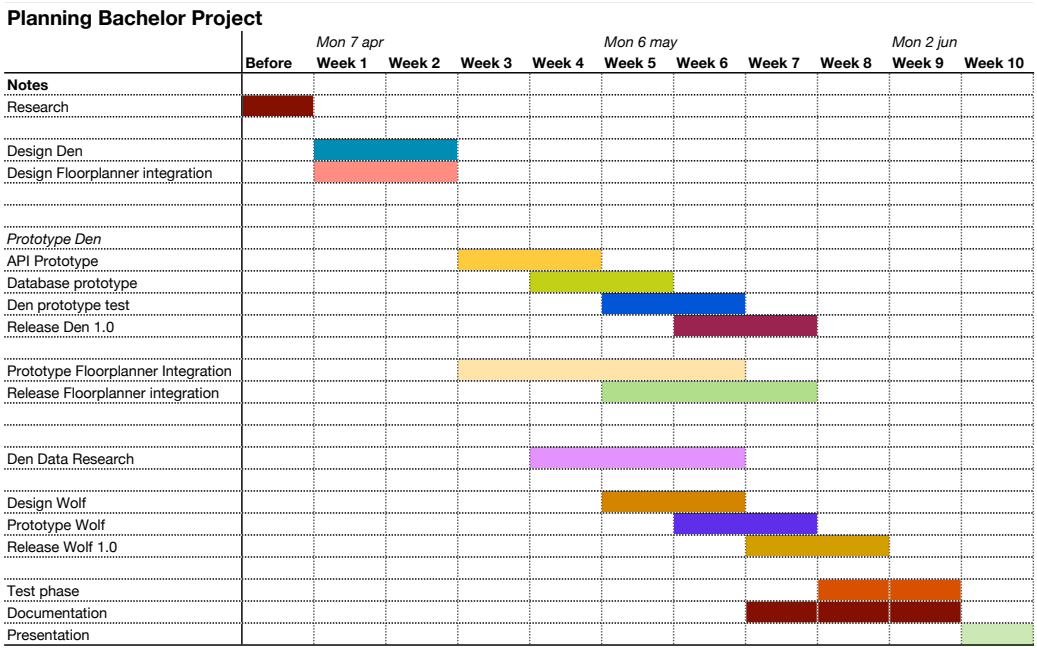

Figure 4.1: The initial planing

4.1 Design and research

This stage will consist of some basic research and design regarding the Den and the Floorplanner integration. Research will be done on various programming languages to determine which one will be the best to use. The goal is to have a good understanding of the problem domain at the end of this stage. Furthermore a detailed design about the functioning of both the Den and the Floorplanner integration should be created.

4.2 Den development

During this stage we plan to create a prototype for the Application Programming Interface (API) for the Den and to create a prototype of the database of the Den. When these prototypes are more or less complete, they should be tested to see if they work as intended. Subsequently after these tests, the prototypes should be adapted and finished for the release of Den 1.0. This 1.0 release will mean that the Den is finished in basis and that its API will not change anymore.

4.3 Floorplanner integration

Parallel to the Den development, work should be started on the Floorplanner integration. Because the Floorplanner integration depends on the Den API, these two stages have to be coordinated together. The goal of this stage is to unleash the functionality the Den provides in the Floorplanner. When the integration enters its release state, the functionality should be finished and stable, and only some polishing should be done to the user interface.

4.4 Den data research

During this stage, research should be done on the data that needs to be stored in the Den database. The fields that will be needed in the database will be determined in this stage and several methods to retrieve relevant information out of the database will be investigated.

4.5 Wolf design and development

In this stage the design of the Wolf should be created, then a prototype should be created, following by the release of Wolf 1.0. This release version should be able

to automatically index an entire domain, and retrieve all the furniture from it.

4.6 End phase

This stage can be used for testing the entire system, to create the necessary documentation, and to prepare for the final presentation. When this part is finished, the project should be completed successfully.

Part II

Design

Chapter 5

Global design

In this chapter the global design of the system is discussed. We first discuss all components and the relations they have with each other. The global design is divided into four components; Den, Wolf back-end, Wolf user interface and Floorplanner integration. Early in the design process these names were invented to make the communication about it easier.

- The database that stores all the furniture is called the Den, as the place to where Wolf drags its furniture to.
- The web spider that searches the web for furniture is called Web Object Library Finder, or Wolf.
- The user interface is called the Wolf UI.
- The Floorplanner integration part, is simply called 'integration', as this will be added to the Floorplanner front end.

5.1 Den

The Den is the central component in the system, it acts as the database where all furniture information is stored. An important aspect of this central role is its API, the API of the Den allows other components to interact with the Den.

5.2 Wolf

The Wolf will be a separate process that searches the web for furniture pages. The Wolf reads a web page using its HTML DOM structure, if the Wolf knows where in this structure the content is, it can save this information and push it to the Den. In the Den this information will be stored so it can be used in the Floorplanner.

5.3 Wolf user interface

In the Wolf user interface, users can add their own furniture to the Wolf. They do this by clicking on the location in the page where the relevant information resides. The Wolf uses this information to save it to the Den.

5.4 Floorplanner integration

The integration part is the front end, and the most visible part of the system. It interacts heavily with the Den, it displays furniture from the Den, and facilitates the placement of furniture in the Floorplanner.

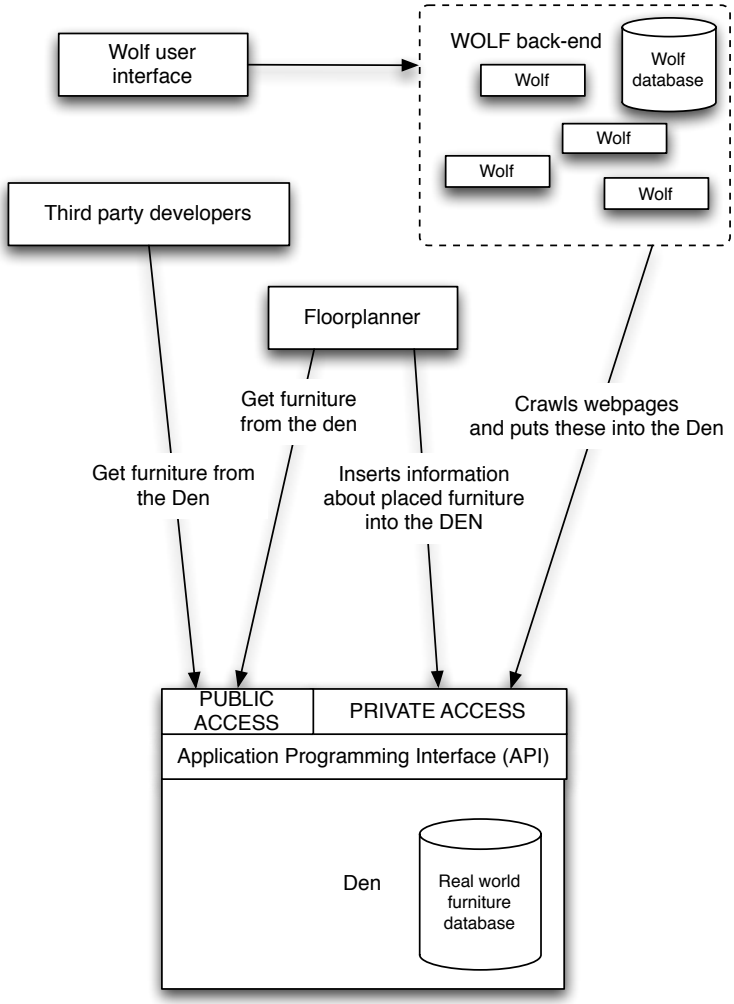

Figure 5.1: The global design

Chapter 6

Den Design

In this chapter the design of the Den will be explained.

6.1 Programming language

As can be seen in Table 6.1, we investigated the following four programming languages; Ruby, Java, C++ and PHP. Each programming language was investigated with the following criteria in mind:

- Learning curve, the ease of learning this language, given our experience with it.
- Performance, execution speed of the code.
- Development speed, speed of development for us.

We believe that these criteria are sufficient for determining which one is best for this project. From this table it can be seen that Java is the best applicable programming language for us by far, this is mainly because we both have followed many courses on Java at TU Delft. Furthermore it can be said that although Java is not the fastest programming language available, it is definitely the fastest language to develop in.

6.2 API

The Den must have a public interface in the form of an API, as can be seen in chapter 3. This is to give third party developers the ability to use the Den.

Language	Ruby	Java	C++	PHP	Explanation
Learning curve	+-	++	–	+-	difficulty of learning this language given our background
Performance	–	+-	++	++	executing speed of the code
Development speed	+-	++	–	+-	speed of development in this language given our background

Table 6.1: Comparison of programming languages

6.2.1 Actors using the API

We have the following three actors that will be using the Den.

Wolf
This actor will be mainly inserting data into the Den. It is actively searching for information about furniture. He sents this information to the Den using the API.

Third party developers
These developers will be mainly getting data from the Den. Important for these users is, as can be seen in chapter 3, that the API will be easy to use.

Floorplanner
The Floorplanner application will be getting data from the Den and will also be giving feedback to the Den.

6.2.2 Research on API's

Research was done on different approaches to API's. The advantages and disadvantages have been summarized in table 6.2 on some criteria, which were created according to the requirements of Floorplanner.com B.V..

- Complexity of the API for a developer.
- Performance of the API calls on the server.
- How easily can this technique be used over the internet.
- Compatibility with current technology Floorplanner.com B.V. uses.

6.2.3 Comparison of protocols

REST

Representational state transfer (REST) is a web service protocol based on the existing HTTP protocol. The HTTP protocol is used for communicating over the internet.

SOAP

SOAP is an XML based protocol for exchanging XML-based messages over a network.

TCP/IP

Another approach is using sockets over TCP/IP by defining a protocol to exchange data.

API	REST	SOAP	TCP/IP
Developer complexity	++	+-	–
Latency (ping)	+-	–	++
Performance	+-	–	++
Web enabled	++	++	+-
Compatibility	++	+-	–

Table 6.2: Research on API's

6.2.4 Conclusion protocol

The conclusion of this research is that REST is the best choice for the Den. It is not difficult to use for a developer, performance is good, latency is good, it is extremely web enabled as it is built upon HTTP and it is totally compatible with the system Floorplanner.com B.V. uses.

6.2.5 Use cases for the API

UC1 Floorplanner sends a design to the Den.
Actor: Floorplanner

1. Floorplanner authenticates itself to the Den.

2. Floorplanner sends the design (placed elements only) to the Den.

UC2 Floorplanner sents once every x minutes all new designs to the Den.
Actor: Floorplanner

1. Floorplanner sends all changed designs to the Den (UC1).

UC3 User adds a new item using the Wolf user interface, this has a high degree of certainty, since it is added by a human.
Actor: User

1. Information is sent to the Wolf, this is all textual information and the degree of certainty.

UC4 Wolf adds a new item, this has a lower degree of certainty since it is added by a robot.
Actor: Wolf

1. Wolf authenticates itself to the Den.

2. Wolf sents a new item to the Den, this is all textual information and the degree of certainty.

UC5 Anonymous request for items.
Actor: Public

1. An anonymous request for a single item given some parameters describing the desired item, and the amount of items desired.

2. The Den returns the best match(es).

UC6 Anonymous request for suggested items.
Actor: Public

1. An anonymous request, sending a set of items. Parameters can be: Style, Popularity, Room in a house, Language.

2. The Den returns a set of items that share close relations to the given items based on the given parameter.

UC7 Anonymous request for a summary view of all items.
Actor: Public

1. An anonymous request, with the parameters of interest, max items (which is limited to N).

2. The Den returns a set of items, with for every item a position of every given parameter. The number of items does not exceed the maximum number of items (it does a sampling).

6.2.6 API specification

For the complete specification of the REST API that is developed, see appendix C.

6.3 Database

As the Den needs a database to store it's data, research was done on MySQL[4] and SimpleDB[5]. MySQL is an opensource database engine, widely used these days. Floorplanner.com BV uses this in their web application framework. Therefore it is really useful to research the capabilities of these two engines. Also research was done on SimpleDB, a new storage engine, developed by Amazon.com. SimpleDB is a distributed tuple space, which according to Amazon can scale very nicely.

Our database needs to have these features:

- Be very scalable.

- Can be clustered over multiple servers.

- Order the data.

6.3.1 SimpleDB

SimpleDB looks promising as scaling is important to the system, as can be read in the requirements. SimpleDB works as a tuple space. Traditional databases are based on fixed column tables. SimpleDB has most of the features a modern database has: sending queries, conditioning and ordering the data. SimpleDB doesn't support ordering of the data, according to Amazon.

6.3.2 MySQL

MySQL is in constant development and as already said, used by Floorplanner. It is able to scale, to cluster multiple MySQL servers and it is able to do joins on data. This DBMS supports all of the requirements.

6.3.3 Conclusion database

As SimpleDB doesn't respect the requirements of the database and MySQL does, MySQL will be used.

6.4 Webservers

The API has to be run on a webserver, several choices are available. After choosing an API protocol, two options are available: Mongrel (based on ruby) and RESTLet(based on Java). Some benchmarks were done as can be seen in appendix A. RESTlet was the fastest of the two.

6.5 Floorplan Element Mapping

The important part of the Den is to automatically create a mapping from Floorplan elements to articles in the Den. Research was done on different methods to accomplish such a mapping and we came up with the following model to map Floorplan elements to articles. First is described what is known, secondly what it must do, and finally the actual implementation.

6.5.1 Known information

The information known about Floorplanner elements is for example the name and a description, furthermore there is an image and we know a color when the element is placed in Floorplanner. For articles we also have a name and a description, a color, a price etc. For a complete list of attributes see appendix B.

6.5.2 Application

As a call is done via the API that requests for an article some Floorplan elements, the floorplan element that 'is most equal' to the article, must be returned.

6.5.3 Vector space model

The Den is based on a vector space model to create a fast performing search. First a vector space is constructed, the Floorplanner elements that have the smallest angle with the article that was requested, are returned.

The vector space

A vector space is defined with as dimensions scores on a collection of words. Each dimension represents the score of a article on that specific word. The collection of words are unique words constructed out of the words that appear in all articles.

Dimensions: scores on words

As already said, the scores on the words are calculated by comparing the string to be indexed on, with the given string. If there are equal words in the given string, these will be used mainly in calculating the score. Also to prevent spelling errors, a comparison of letters is done. A score is calculated on the outcomes of this.

Selecting an element

To select an element we return the elements with the smallest angle to the given article (see equation 6.1).

$$cos(\theta) = (v \bullet w)/(\|v\| \cdot \| w\|) \tag{6.1}$$

6.5.4 Example vector space search

As can be seen in table 6.3, some FP Elements are given with their scores on the words. In figure 6.1 these words are plotted. As can be seen, the angle between FP Element 3 and the given article is the smallest of all angles with the given articles. Therefore FP element 3 is returned, because this is the most relevant match.

Object	Score on word chair	Score on word table
FP Element 1: Wooden chair	0.9	0.2
FP Element 2: Outdoor chair	0.8	0.3
FP Element 3: Outdoor table	0.05	0.95
Given article: Table	0.1	0.9

Table 6.3: Articles and their coordinates in the vector space

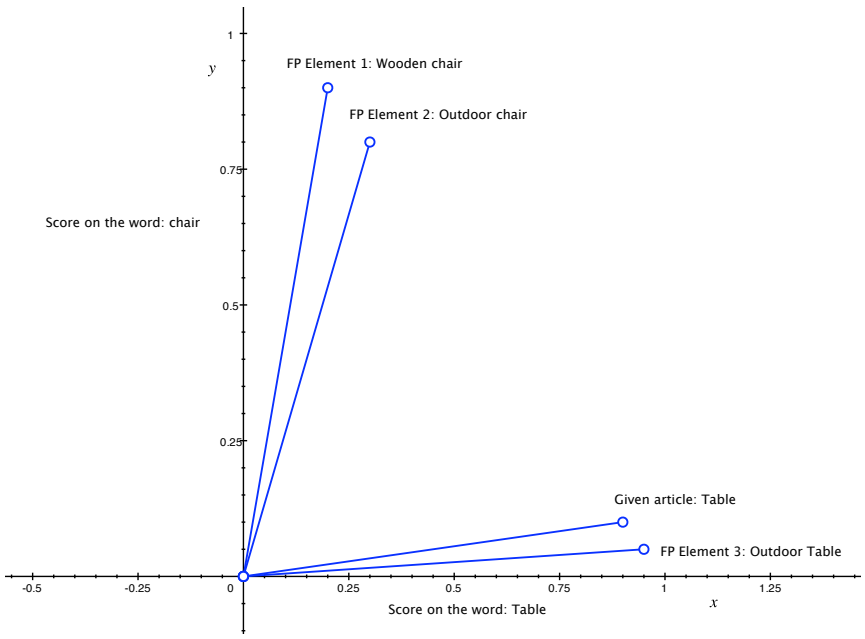

Figure 6.1: An example of the vector space

Chapter 7

Wolf back-end design

In this chapter the design of Wolf is discussed. In Figure 7.1 the overall design of Wolf is shown, this image will be used as a guide for this chapter. The wolf consists of three separate components. A database, a domain crawler and several page crawlers. The database is used to store Wolf relevant information, the domain crawler is the host process which monitors the page crawlers. A page crawler is a process that retrieves and fetches a web page. In figure 7.1 the way Wolf works is illustrated.

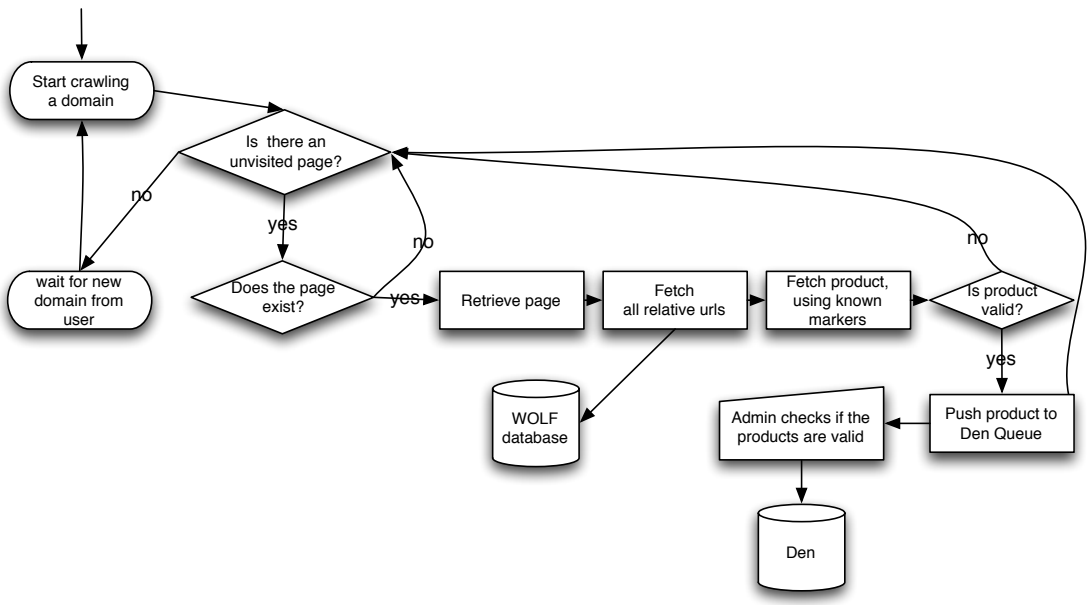

Figure 7.1: Wolf flow diagram

7.1 Crawling a domain

The wolf is able to crawl an entire domain, the user can specify which domain. Crawling a domain is started from the home page of the domain, it is assumed that every page is indirectly accessible by following links on this home page. Prerequisite for the crawler is the existence of one or more markers in the database. These markers are used to determine the location of the content on a page. The markers are discussed in depth in Chapter 8.3.

For performance purposes the wolf can instantiate multiple page crawlers in separate threads, this is an advantage because most of the time a thread is waiting for the loading of a page. With multiple threads, pages will be crawled concurrently. Furthermore multiple threads also take advantage of any distributed system the Wolf might be deployed on in the future. Probably a good number of threads to use is based on the number of processor cores that is available in the system where it is deployed. Wolf can be given the following set of parameters:

Number of threads This is the number of webpage crawlers that are instantiated.

Maximum number of page visits An estimate to the maximum number of pages that will be visited.

Monitor interval This is an interval in milliseconds, every interval a check is done on the progress, if the number of visited pages is equal or greater than the maximum number of page visits, the process is stopped.

7.2 Crawling a webpage

As can be seen in Figure 7.1, when a page exists it is first retrieved, then all relative urls are fetched and finally a product is tried to be fetched. Note that this crawling of a webpage happens in a separate thread, this means that at the same time multiple pages can be crawled.

7.2.1 Fetching URLs from a page

Every URL found on a page is stored in the Wolfs database, in this way, all unique pages are aggregated from a domain. All external links are ignored, only the links to content on the same domain are added to the database.

7.2.2 Fetching a product from a page

The Wolf tries to fetch an article out of a page by checking the DOM locations that are stored in the markers. If on these locations no content is found, the page is rejected as an article page, the page will not be pushed to the Den. If however, on every domain location information is found, this is added to the Den.

7.2.3 Sending articles to the Den

Pages that are added to the Den are not immediately pushed to the Den, for now there is a user check in between. On a web page of the Wolf user interface, all fetched articles are shown, and the user (somebody working at Floorplanner.com BV) can decide which articles are to be added to the Den. This adding to the Den is done in a separate update process which checks the Wolf database regularly for updates.

7.3 Database

In the database all the Wolf's findings are stored. For every fetched url a database entry is created, this allows the Wolf to keep track of its visited pages in a central location. Every thread of the Wolf is in this way always up to date with the latest page status. In the database the markers are also stored, these markers are created in the Wolf user interface. They have an HTML DOM location for every field of a product per page.

Chapter 8

Wolf user interface design

With the Wolf user interface (Wolf UI), page structure information can be added to the Wolf back-end, so complete domains can be crawled. The UI of the Wolf must be able to find the structure of a webpage. As a little research was done on product pages of furniture websites, the conclusion was drawn that all of the pages inside a domain have the same structure.

8.1 Users

Floorplanner users
The goal of these users is to add furniture into their floorplan using the Floorplanner application. They have, for example, seen a couch on a website. They want to see if this couch fits into their house. So they want to let it show up in their online floor plan.

Floorplanner.com B.V. employees
Employees want to add complete websites containing furniture to the Den.

8.2 HTML Document Object Model

The HTML DOM is the way webpages are built up. It is a tree containing elements, which mostly represents some element on the screen (see figure 8.2. More can be read about the DOM at [6].

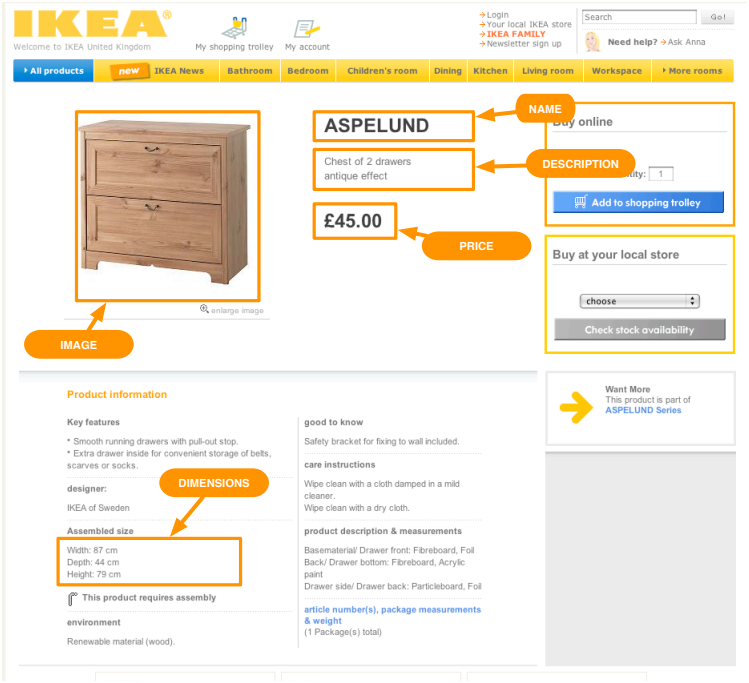

Figure 8.1: The targets of a product page

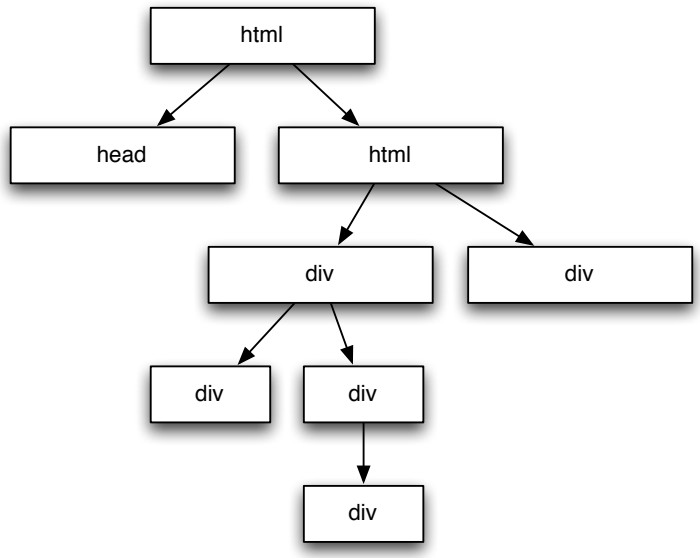

Figure 8.2: The HTML DOM

8.3 Targets

As product pages must be crawled using the wolf, we must know about the structure of such a page. A product has attributes(see appendix B). A product page contains some or all of these attributes. These attributes will be called targets, when the Wolf UI is searching for them. See figure 8.2 for a visual representation of targets on a productpage.

8.4 User interface

As browsers already use HTML it is handy to use them for this purpose. Therefore the Wolf UI will be implemented in HTML using PHP as back-end and Javascript as a DOM walker.

8.5 Finding the targets

The UI of the Wolf must give the markers of the targets to the Wolf. We use JavaScript to display a product page and to be able to, when a target is selected, point to the location on the page. Then the path in the DOM to this element is calculated. This is the marker for a given target. After the user has pointed out all information that was searched on a product page, this collection of markers for attributes is sent to the wolf.

Chapter 9

Floorplanner integration design

The Floorplanner integration is about integrating the new functionality provided by the Den into the Floorplanner user interface.

9.1 Programming language

Because the goal of this project is to integrate the system seamlessly into the Floorplanner, it was decided to only use techniques and languages that are already used in the front end of the Floorplanner. The languages currently used are:

- Flash ActionScript
- JavaScript
- HTML
- CSS
- Ruby (this is also used for back end)

Since we both have extensive experience in the languages listed above, it was clear that we should use one of these. An other import reason for one of these languages is, that it is a lot easier to combine two program parts in one language, than to combine two program parts of different languages.

9.2 Drag and drop articles

In the current version of Floorplanner it is possible to drag furniture from a library on a plan. The idea is to use the same approach, but then instead of dragging a piece of furniture from Floorplanner, an image of an article from the Den can be dragged in to the Floorplan. This is illustrated in Figure 9.1.

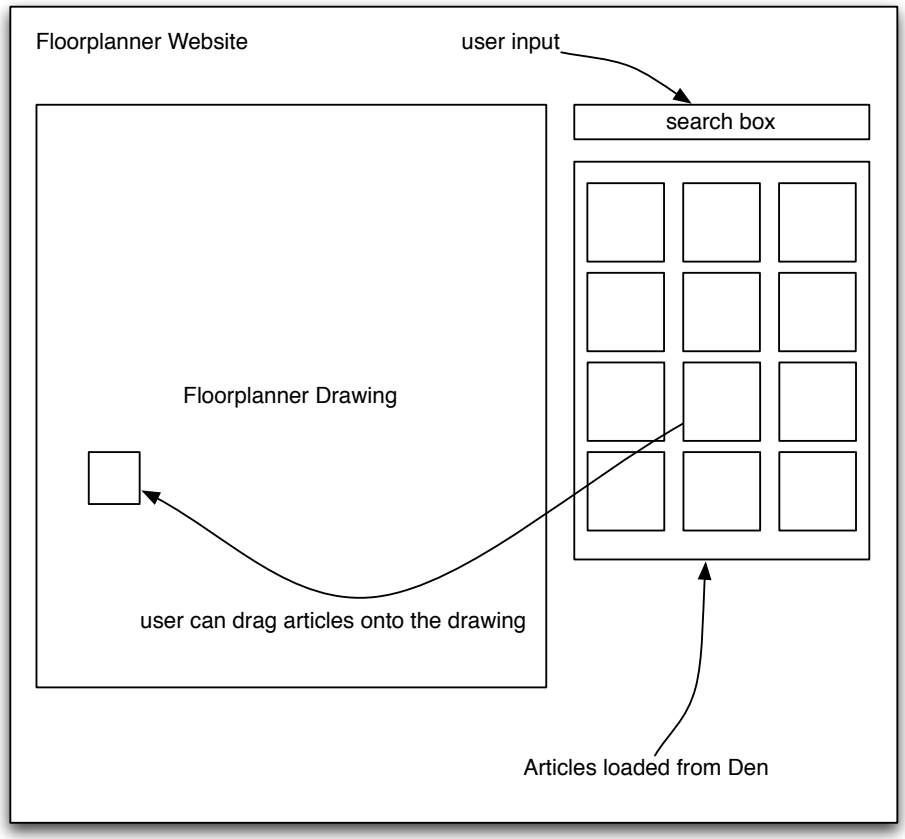

Figure 9.1: A sketch of the Floorplanner integration

Part III

Implementation

Chapter 10

Implementation

In this chapter the final implementation is showed. The whole process of an user, who wants to add furniture elements to the Den, is described. It will be showed with screenshots of the final implementation that was done.

10.1 The user

In the process of adding furniture two kind of users are identified:

- A Floorplanner customer, a person with a Floorplanner account, this will be called 'user'. The goal for this user is to add furniture from a website, for example Ikea.com, to his or her Floorplan.
- A Floorplanner.com BV employee, somebody working at Floorplanner.com, this will be called 'employee'. The goal for this user is to do a quality check on the added furniture, to confirm that the added furniture is indeed good enough to be added to Floorplanner.

10.2 Step 1: Finding a product page

The first step is to search for a product page that contains a furniture element. See figure 10.1.

10.3 Step 2: Adding the product page

The second step is to click the button in the browser, to add the product page to the Wolf. See figure 10.2.

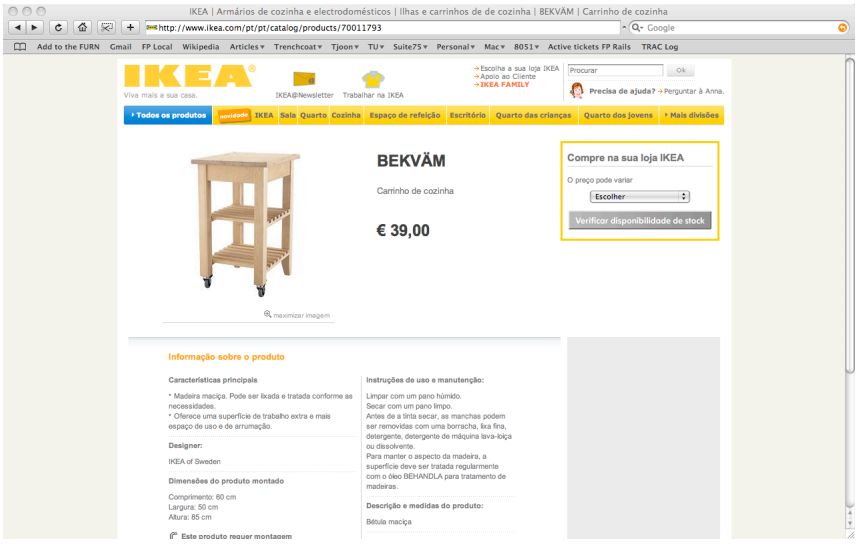

Figure 10.1: Finding a product page

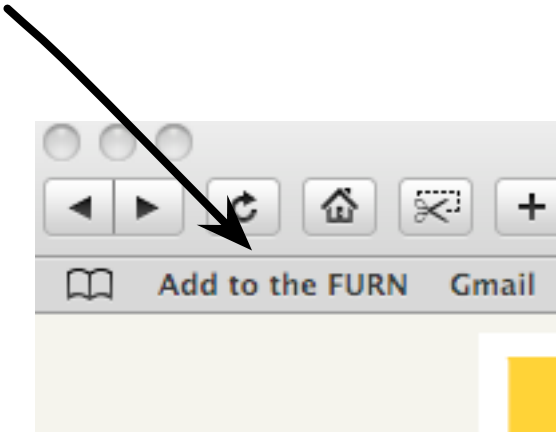

Figure 10.2: Clicking the add button

10.4 Step 3: Tell how the page is structured

Now the page is loaded with a sidebar (the Wolf UI), containing targets as can be seen in figure 10.3. These targets can be selected and after that an element in the HTML Dom can be pointed to that target.

Figure 10.3: Product page with the Wolf UI

Figure 10.4: The targets

10.5 Step 4: Push the result to the Wolf

When all targets are selected, the result can be added to the Wolf, which can use the information to automatically index the just added page, or an entire domain. The adding is done by pressing the 'add result' link.

10.6 Step 5: Automatically indexing using the Wolf

The wolf can add the pages that are selected by the user, these pages will come up in the Wolf user interface which is discussed in step 6. The wolf can index an entire domain based on the targets that are selected. The Wolf will crawl the domain, and try to fetch a product from every page.

10.6.1 Wolf back end usage

The Wolf is intended to be an automatic process that can be turned on and off by the Floorplanner team. Because no end user will ever use this, a command line interface is sufficient. Through this interface feedback is given about the progress of the crawling process. Furthermore it can be easily started and stopped from this interface. In Figure 10.5 the command line output is shown for a typical run of the Wolf. It can be seen that the Wolf has fetched 24 pages, and that every page was recognized to contain one product. Normally not every page on a domain will contain a product, but in this artificially created situation it does.

10.7 Step 6: Checking the results

In Figure 10.6 the quality check page is displayed. This page is accessible for Floorplanner employees only, and this allows the company to control the article adding process. This prevents the adding of invalid articles to the furniture. When the button with 'Results are fine' is pushed, the selected articles will be saved in the Den. From now on this article will be available in the Floorplanner.

10.8 Final step: Floorplanner integration

In this section the process where the user adds an article in their Floorplan is described.

```
Console
<terminated> Wolf [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/
Starting 4 threads...
Starting fetcher
Set DOM Server Locations for: http://www.ikea.com
Starting fetcher
Set DOM Server Locations for: http://www.ikea.com
Identified: GOLIAT as product
Starting fetcher
Set DOM Server Locations for: http://www.ikea.com
Identified: JULES as product
Identified: SKRUVSTA as product
Identified: KOLON as product
Starting fetcher
Set DOM Server Locations for: http://www.ikea.com
Identified: ANTIFONI as product
Identified: MAMMUT as product
Identified: TROFAST as product
Identified: MAMMUT as product
Identified: LEKSVIK as product
Progress: 45.0 %
Identified: BARNSLIG RUND as product
Identified: BARNSLIG RUND as product
Identified: MAMMUT as product
Identified: KVARNÖ as product
Progress: 65.0 %
Identified: TROFAST as product
Identified: KARLSÖ as product
Identified: DYNING as product
Identified: GRENÖ as product
Identified: SMILA BLOMMA as product
Progress: 90.0 %
Identified: SOMMAR as product
Identified: SALVIA as product
Identified: DYNING as product
Identified: HENSVIK as product
Progress: 100 %
Identified: MAMMUT as product
Identified: HUSÖN as product


---Stopped fetching---
Visits: 24
Dead links: 0
Fetched products: 24


running time: 3 s
```

Figure 10.5: The Wolf in action

10.8.1 Article library

In Figure 10.7 the articles are displayed in the sidebar of the Floorplanner. These articles are directly loaded from the Den, and can be dragged as described in the next section into the Floorplanner.

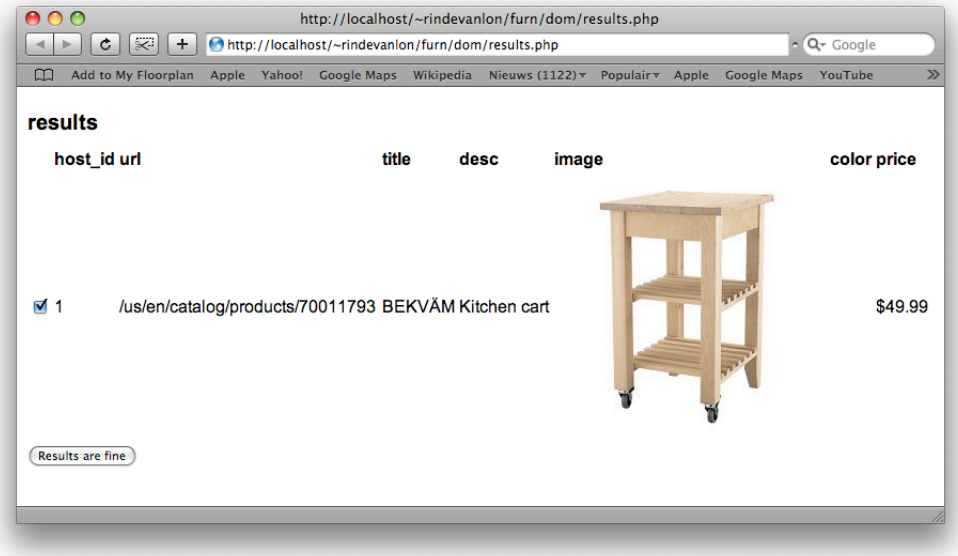

Figure 10.6: Quality check page for added articles

10.8.2 Article placement

In Figure 10.8 the integration can be seen in action, the rightmost image displays an article image that is being dragged by the user from the article library. The middle image shows what happens when the article is dragged into the Floorplanner. It can be seen that while dragging the article the Floorplan element appears when hovering over the Floorplanner. In the left image the user has released the mouse, and successfully placed an article from the Den into the Floorplanner. From now on the article is just a Floorplan element, but with added information about the article. When the user clicks on the element, a properties form can be shown, as can be seen in Figure 10.9. It can be seen that the original article image is shown in this form.

10.8.3 Gathering statistics

When an article is placed into the Floorplanner, a API call is done, to inform the Den about this placement. The Den saves this information to do calculations on these statistics later. Based on this information popularity statistics for furniture can be generated.

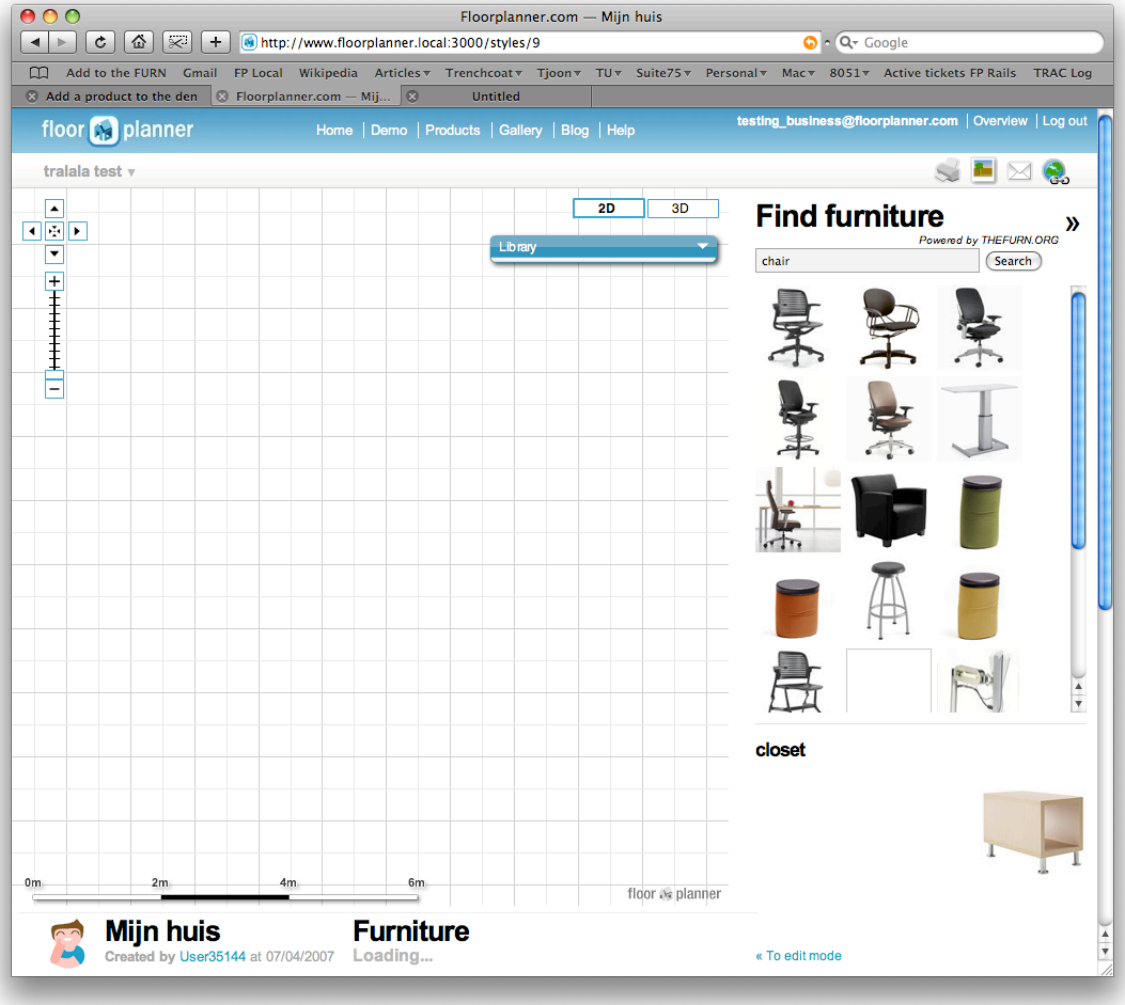

Figure 10.7: The articles in a sidebar next to Floorplanner

Figure 10.8: From right to left; dragging an article into the Floorplanner

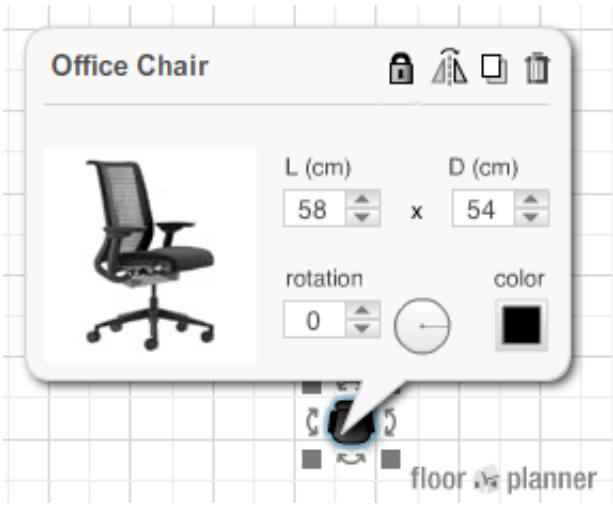

Figure 10.9: Floorplanner properties form showing the original article.

Chapter 11

Deployment

After implementing the Den and the Wolf the products had to be inserted into a running environment. As was stated in the design the two parts of the system had to be deployed in a distributed environment. Amazon EC2 was chosen to deploy the applications on.

11.1 Creating a virtual server

Server instances can be started on Amazon EC2 by using an API call. After starting a server, a complete Unix environment was available to install the parts of the system on. Together with Floorplanner.com B.V. the domain name 'thefurn.org' was pointed to this instance.

11.2 Database

As was stated in chapter 6 the database management system to be used was MySQL. This was installed and the database structure defined was created in this database manager.

11.3 Den

Because the Den was already, in the implementation phase running in a unix environment, the deployment was easy. The distribution that was build was checked out from the version control system and the Den was running.

11.4 Wolf

Deployment of the Wolf was done on the same server. This can be easily extended to another server as it was designed to be able to do this.

Part IV

Testing

Chapter 12

Testing

12.1 Unit tests

While doing the implementation, unit test cases were created. JUnit [7], written for testing Java applications, was used as test framework. The test-suite is splitted in parts and these parts are tested individually. The tests were created to improve the development process. All test-cases succeeded in the end.

12.1.1 Testsuite: Database

The DBMGR package was used for database communication. For this package test-cases already existed. They were extended to cover the usage in the implementation.

12.1.2 Testsuite: REST

For the communication based on the REST protocol we used the RESTLet package. This package provides superclasses for RESTful resources. As we extended these classes, extensive test classes were provided. These test classes test wether the Application Programming Interface (API) works as expected.

12.2 Performance testing

Since the system had to be built for success, it should be able to scale as it gets more used. As can be read in chapter 3 the performance of the DEN must be less than 500ms per request. We did a randomization on the restful resources that could be invoked. For this testing the Apache HTTP server benchmarking tool

Number of random requests done	10000
Time per request (mean)	15.862ms

Table 12.1: Performance testing

citeab was used. In table 12.1 are the results of this performance testing. As can be seen the request rate of the DEN is far below what was required.

12.3 User tests

12.3.1 Third party developers

While the API is opened up for third party developers tests had to be done, to see if the API works as expected. As Floorplanner.com B.V. was also doing another project, which was about adding tips to Floorplanner elements. This was an ideal test case for the Den. The API of the Den was given to one of the developers at Floorplanner.com B.V..

12.3.2 Conclusions third party developers

After working with the API the developer was questioned. The programming language the developer used was Javascript. He explained there were some difficulties due to browser security, but he concluded this was a browser and language specific issue and he said it was simply fixed by using a proxy. For the rest he said it took only 3 hours to implement the tips engine in Javascript using the DEN and was very enthusiastic about how easy it was to add 'things' to Floorplanner elements, so he did not need to care about the mapping.

12.3.3 Wolf users

As the users also want to add furniture to their floor plans we observed a user after providing him with an assignment.

The assignment

The assignment that was provided consists of:

1. Go to http://www.furniturefromhome.com.
2. Index a product item.

The observation

Some observations that were done:

- After clicking the toolbar it was not clear what has to be done. The user saw it after some experimentation, but it was not obvious.
- When he wanted to select the title of product, he selected the title of the page, instead of the name of the article.
- After adding the result to the Wolf, there was no feedback, which made the user unsure about what to do.

Conclusions Wolf user test

After the observation was done, we looked at the design and came up with the following improvements:

- Before showing the design a little movie must be played, that tells the user what to do.
- A description must be added to targets, to explain to people what this target is.
- After adding the element, the Wolf UI must give feedback, what was done.

Part V

Conclusions

Chapter 13

Conclusions

In this chapter we look back on this project and present our conclusions. We first present the conclusions for each component, followed by a general conclusion.

13.1 Den

A system was created that couples Floorplanner elements in smart ways to indexed articles. As this was the main goal of this project, we can conclude that this system works as was stated in the requirements. One of the requirements was to have an API that enables both Floorplanner as well as third party developers to access information in the Den. This API can handle requests in less then 500ms, only 15ms are needed per request. Also access control was implemented. The system had to be able to scale in size and in request performance and to run in a distributed environment. By deploying to Amazon EC2 we established this. Also views on the data are possible through the API. The Den respects the requirements and we can therefore conclude the Den was successfully developed.

13.2 Wolf back-end and user interface

The Wolf had to be able to crawl and index webpages. As we created a user interface for the Wolf, it is possible to give clues to the Wolf about the structure of a webpage. After that it can index webpages succesfully. The Wolf had also to be deployed in a distributed environment and take advantage of this environment. Which we made possible by implementing the Wolf back-end multi-threaded.

13.3 Floorplanner integration

The Den had to be integrated into the existing Floorplanner technology. This was designed and implemented by us. It meets it's requirements because the integration of the Den is totally invisible to the end user since the process of adding furniture did not change in basis.

13.4 General

As we look back on this project, we have a system that scrapes webpages and couples this information in smart ways to Floorplanner elements. Also every requirement has been met. Unit tests were done, which indicate the software works as expected. Also the API was tested by a Floorplanner employee who was very enthusiastic. Finally we can conclude, this project was a success.

Chapter 14

Recommendations

The project can be extended in several ways. Some things we want to recommend:

- More perspectives on the data can be generated.
- More classes of elements can be put into the Den.
- The points described after doing the user tests, can be implemented.
- The Wolf should be improved to be stable enough to add the articles unsupervised.
- The system now has a certain focus towards furniture, but this should be expanded to include all kind of things, for example paintings, floors, windows and doors.
- Furniture manufacturers should be able to manage their collection of furniture in the Den. For example they should be able to pay for a higher ranking in the Den.
- Extend the scraping mechanism as used in the Wolf user interface, to make it more generic.
- Extend the Den API to give more insight in the Den, to open it up more.
- Every user should be able to have an own personal furniture set.
- Users should be able to add a price-watch to a piece of furniture, to receive a notification when the price changes, or drops below a certain value.

Part VI

Evaluation

Chapter 15

Process

15.1 Evaluation

15.1.1 Personal

Overall the cooperation between us went very fluently. As we already did a lot of projects together, we know exactly how to work together. Nevertheless sometimes we had some conflicts. We are both very stubborn types, which sometimes gave some conflicts about where to go next. We resolved this conflicts by debating them, but sometimes we just waited until the next day to cool down a bit. One other problem we sometimes faced, was that sometimes we were so deep into the technical matter, that we barely knew what the other was doing. In these situations we should have communicated a bit more, but it wasn't a real problem.

15.1.2 Floorplanner.com B.V.

We experienced the cooperation with Floorplanner.com B.V. as very good. Floorplanner.com BV is a real no nonsense company. They appreciate prototypes and good arguments much more than a report or other paperwork. We liked this valuation very much, since we like to do practical things, and not to create a lot of paperwork. Furthermore they were also very assistive, if we needed to know something, there was always time to explain things. Overall we experienced it as a very nice cooperation and a fun place to work.

15.1.3 Delft University of Technology

Overall we have the feeling that our contact with our supervisor, Stijn Oomes was good. He was to the point, and had some nice recommendations for us. Especially regarding users, who we tend to forget sometimes. In the second half of

the project we were so busy, that we sometimes forgot to send Stijn updates about our progress. We believe that at the end of the project we improved this by collaborating closely with Stijn about this final report. In a future project we would pay more attention to the communication with our supervisor as this can be very useful for identifying possible pitfalls early.

15.2 Planning

Afterwards we can say the planning was remarkably accurate. Several parts were finished well within their time, but some other parts took a little bit more time. One thing that we would do differently the next time, is to begin earlier with the final report. Although we managed to deliver the report on time, it caused us quite some stress on the last days.

Acknowledgements

We would like to acknowledge the following very helpful people who have assisted us in our work:

- *Ir. Jeroen Bekkers* - CEO of Floorplanner
  For giving us this opportunity and bringing in his endless creativity during this project and giving us a lot of freedom during the project.
- *Dr. Stijn Oomes* - Assistant Professor at TU Delft
  For guiding us in this project and reminding us of the existence of users.
- *Ir. Gert-Jan van der Wel* - CTO of Floorplanner
  For helping us with our technical questions and with the writing of this report.

Appendix A

Server performance tests

A.1 Mongrel

Mongrel is a web server written in Ruby.

Server Software: Mongrel
Server Hostname: floorplanner.local
Server Port: 3000
Document Path: /
Document Length: 7557 bytes Concurrency Level: 50
Time taken for tests: 18.86923 seconds
Complete requests: 10000
Failed requests: 0
Write errors: 0
Total transferred: 77550000 bytes
HTML transferred: 75570000 bytes
Requests per second: 552.89 req per sec (mean)
Time per request: 90.435 [ms] (mean)
Time per request: 1.809 [ms] (mean, across all concurrent requests)
Transfer rate: 4187.11 [Kbytes/sec] received

A.2 RESTlet(Java)

RESTlet is a web server, which implements the REST approach. It is written in JAVA.

```
Server Software: AsyncWeb
Server Hostname: floorplanner.local
Server Port: 8182
Document Path: /
Document Length: 12 bytes
Concurrency Level: 50
Time taken for tests: 10.426277 seconds
Complete requests: 10000
Failed requests: 0
Write errors: 0
Total transferred: 2950885 bytes
HTML transferred: 120036 bytes
Requests per second: 959.12 req per sec (mean)
Time per request: 52.131 [ms] (mean)
Time per request: 1.043 [ms] (mean, across all concurrent requests)
Transfer rate: 276.32 [Kbytes/sec] received
```

Appendix B

Dataformat specification

Name	Datatype
Photo	(url)
- brand	(id)
- description	(string)
- model	(string)
- price	(float)
- family (furniture, luminary, dishes...)	(string)
- category (sofa, chair...)	(integer)
- room of the house dedicated (kitchen, living-room...)	(reference)
- height	(int)
- width	(int)
- depth	(int)
- diameter	(int)
- style (zen, moroccan...)	(string)
- color	(int)
- material (leather, textile...)	(string)
- address (URL + address + phone)	(string)

Appendix C

Webservice specification v1.0

C.0.1 Actors

FP: Floorplanner
WOLF: Wolf
PUBLIC: Everyone who wants to use the API

C.0.2 Resources

- placed_elements
- articles
- perspectives

C.0.3 Specification

UC1
Method: add placed elements
Call: POST / placed_elements
Parameters:
elements - list of placed elements in a Floorplanner design
context - list of parameters: language
Return: -
Access: FP

UC3 + UC4
Method: add article

Call: POST /articles
Parameters:
article - the article to be added to the den
certainty - how sure are we this data is valid
Return: -
Access: WOLF
Notes: Maybe certainty will be on attribute level instead of overall

UC5:
Method: get article
Call: GET /articles/id
Parameters: id - id
Return: single article
Access: FP, PUBLIC, DEN

Method: get article floorplan element
Call: GET /articles/id/fpelement
Parameters: id - id
Return: floorplaner element
Access: FP

Method: search articles
Call: GET /articles/search
Parameters: TODO
maxitems - maximum number of returned articles
Return: list of articles
Access: FP, PUBLIC, DEN

UC6
Method: give suggestions based on set of articles
Call: GET /articles/suggest
Parameters: articles - a set of articles
selection - style, popularity, room in the house, language
Return: articles
Access: FP, PUBLIC, DEN

UC7
Method: get perspective
Call: GET /perspective/
Parameters:
Return:

Bibliography

[1] "Python vs Perl vs Java: A comparison of Java vs C++", Mike Connel, 2004 Online at http://furryland.org/~mikec/bench/

[2] "Performance of Java versus C++", J.P.Lewis and Ulrich Neumann, 2004 Online at http://www.idiom.com/~zilla/Computer/javaCbenchmark.html

[3] "A Hybrid Layout Algorithm for Sub-Quadratic Multidimensional Scaling", Alistair Morrison et al, 2004 Online at http://www.dcs.gla.ac.uk/~matthew/papers/SubQuadMDS.pdf

[4] "MySQL' Online at http://www.mysql.com/

[5] "SimpleDB' Online at http://www.amazon.com/SimpleDB-AWS-Service-Pricing/b?ie=UTF8&node=342335011

[6] "W3C HTML Dom Online at http://www.w3.org/DOM/

[7] JUnit Online at http://www.junit.org/

[8] Apache Benchmarking tool Online at http://httpd.apache.org/docs/2.0/programs/ab.html