UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Taxonomy for Web-programming technologies

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Tobias Zimmer

Erstgutachter:	Ralf Lämmel Institut für Informatik
Zweitgutachter:	Andrei Varanovich Institut für Informatik

Koblenz, im Februar 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . .

(Ort, Datum) (Unterschrift)

Zusammenfassung

Zur Entwicklung von Webanwendungen und Webseiten existieren viele verschiedene Technologien und Konzepte. Jede dieser Technologien implementiert bestimmte Anforderungen, wie z.B. das Erzeugen von Inhalten oder die Kommunikation zwischen Client und Server. Verschiedene Konzepte helfen, diese Technologien innerhalb einer Webanwendung zusammenzufügen. Nicht zuletzt Architekturstile und -muster gehören zu diesen Konzepten. Die Diplomarbeit beschreibt einen Ansatz zur Erstellung einer Taxonomie dieser Technologien und Konzepte unter Zuhilfenahme der freien Enzyclopädie Wikipedia, im speziellen der Kategorie "Web-Application Framework". Unser 101companies Projekt benutzt Implementationen, um die einem Web-Application-Framework zugrunde liegenden Technologien zu identifizieren und zu klassifizieren. Innerhalb des Projekts werden Taxonomien und Ontologien mit Hilfe dieser Klassifikationen erstellt. Zusätzlich beschreibt die Ausarbeitung, wie nützliche Web-Application-Frameworks mit der Hilfe von Wikipedia priorisiert werden. Abschlie̋send enthält die Diplomarbeit auch die Dokumentation der betreffenden Implementationen.

Abstract

Web-programming is a huge field of different technologies and concepts. Each technology implements a web-application requirement like content generation or client-server communication. Different technologies within one application are organized by concepts, for example architectural patterns. The thesis describes an approach for creating a taxonomy about these web-programming components using the free encyclopaedia Wikipedia. Our 101companies project uses implementations to identify and classify the different technology sets and concepts behind a web-application framework. These classifications can be used to create taxonomies and ontologies within the project. The thesis also describes, how we priorize useful web-application frameworks with the help of Wikipedia. Finally, the created implementations concerning web-programming are documented.

Acknowledgements

It is a pleasure to thank Ralf Lämmel (Software Languages Team, Koblenz) and Andrei Varanovich (Software Languages Team, Koblenz) for great support and guidance during the development of the thesis. I also would like to thank Thomas Schmorleiz and Sebastian Jackel for the help and the useful feedback. Finally, I want to thank my family and friends for their patience.

Contents

Chapter 1

Introduction

1.1 Motivation

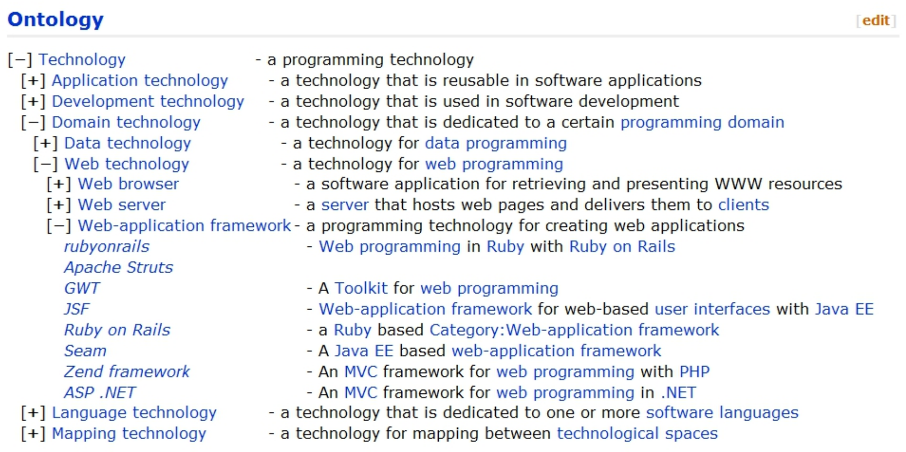
The evolution of web application development over the past years leads to a huge amount of web-programming technologies related to different programming languages and different use cases. There are countless approaches for data access, client-server communication, GUI development and the improvement of the development process itself. The summary and classification of the web-programming technologies leads to several challenges. It is difficult to classify a technology if the vendor of the technology changes its strategy or simple technological evolution leads to different approaches. And it is nearly impossible to get a complete overview of all related technologies. A survey (e.g., ”Survey of Technologies for Web Application Development” [8]) can provide a classification of a fixed technology set within a fixed domain at a fixed time. Our approach is, to gain more flexibility due to the use of a MediaWiki system [9]. The underlying project is called ”101companies project” [17]. It identifies technologies and concepts with the help of implementations, which are documented in a defined manner. The implementations shall help to classify and present included technologies and concepts. The advantages of the MediaWiki approach are adaptability and extensibility. That helps to follow technological evolution and strategy changes and to extend the system by new technologies and concepts. Furthermore, it provides a support for categorization and ontology overviews. The ”101companies project” uses well-documented implementations based on different technologies to get an empirical entry point for creating classifications and ontologies (see Fig. 1.1) of technology sets. Each implementation is based on a fixed set of use cases build upon specific requirements[1]. Since the project contains a huge field of different programming scopes, it also provides a connection between ”regular programming” and ”web programming” [8].

The web development domain includes many different web-programming technologies. Hence, it is difficult to find reasonable technologies for our ”101companies project” implementations. The first intention of the thesis is to identify such a set of technologies. The contained elements should be well definable and classifiable. We describe and evaluate two different approaches within the thesis. The first approach is programming-language centered and the second is category[2] centered. We use the free encyclopaedia Wikipedia[3]

[1] http://101companies.org/index.php/101companies:System

[2] http://en.wikipedia.org/wiki/Wikipedia:Categorization

[3] http://en.wikipedia.org/



Ontology [edit]

[−] Technology	- a programming technology
[+] Application technology	- a technology that is reusable in software applications
[+] Development technology	- a technology that is used in software development
[−] Domain technology	- a technology that is dedicated to a certain programming domain
[+] Data technology	- a technology for data programming
[−] Web technology	- a technology for web programming
[+] Web browser	- a software application for retrieving and presenting WWW resources
[+] Web server	- a server that hosts web pages and delivers them to clients
[−] Web-application framework	- a programming technology for creating web applications
rubyonrails	- Web programming in Ruby with Ruby on Rails
Apache Struts	
GWT	- A Toolkit for web programming
JSF	- Web-application framework for web-based user interfaces with Java EE
Ruby on Rails	- a Ruby based Category:Web-application framework
Seam	- A Java EE based web-application framework
Zend framework	- An MVC framework for web programming with PHP
ASP .NET	- An MVC framework for web programming in .NET
[+] Language technology	- a technology that is dedicated to one or more software languages
[+] Mapping technology	- a technology for mapping between technological spaces

Figure 1.1: 101companies ontology example

to get a key category for the identification of web-programming technologies. As we see in the following chapters, the second approachs leads to the widely used term and category ”web-application framework”. The second intention of the thesis is to introduce a first subset of the identified technologies within implementations, and document them using our MediaWiki system. We will also show, that there are intersections between regular programming and web programming.

1.2 Thesis structure

The thesis is divided into three main chapters. In the second chapter, we discribed the two language-centered and category-centered approaches. We evaluate them and show, why our decision is on the second. At the end, we present the results of the initial web-programming technology set. The following chapter points to the related work. It describes, where we get our ideas from and which work might be helpful or interesting for the future. The fourth chapter illustrates the feature coverage of the implementations. It is followed by the documentation chapter of the implementations provided by the ”101companies project”. It is strongly recommended to visit the MediaWiki system to read the documentations although they are completely contained in the thesis. At last, there is a conclusion about our measurements and achievements. We now want to further describe the structure of each implementation documentation included in the thesis.

1.2.1 Structure of implementation documentations

The implementation documentations are structured equally to each other. The guideline to the ”101companies project” [16] provides a couple of mandatory sections for every implementation page in a given order. The *Intent* alleviates the primary classification of the implementation. Both, the *Languages* and *Technologies* section are mandatory for the taxonomy since they contain the most significant relationships to supporting technologies. The *Features* list is necessary for documenting the standardization of all implementations within the 101companies project. The *Motivation* contains a short abstract about the implementation pointing to the most interesting parts. It also introduces important concepts for our taxonomy and distinguishes the specific implementation against the others. The concrete use of technologies, patterns, and concepts is described within the *Illustration* section. The *Architecture* section helps to find the relevant parts of the application in the file system. And finally the *Usage* section describes the installation, compilation, and/or opening of the developed web-application.

Chapter 2

Popular web-application frameworks

2.1 Subjects of research

As mentioned in the introduction, there are two different approaches for the assembly of web-programming technologies. The first approach uses relations between languages and technologies. The second approach uses relations between an overall category called ”web-application framework” and technologies and languages. Before we are able to talk about the two approaches, we have to define the term ”web-application framework”. After that, we describe the character of the two approaches as well as their advantages and disadvantages. We also explain our preference for the ”web-application framework”-category approach.

2.1.1 Definition: Web-application framework

The first step is to split the term into two parts: ”web application” and ”framework”. The parts are defined as follows:

Web application	”A Web application is a software system based on technologies and standards of the World Wide Web Consortium (W3C) that provides Web specific resources such as content and services through a user interface, the Web browser.” [14]
Framework	”A framework is a model of a particular domain or an important aspect therof. A framework may model any domain, be it a technical domain like distribution or garbage collection, or an application domain like banking or insurance. A framework provides a reusable design and reusable implementations to clients.”[29]
	”A software framework is a set of code or libraries which provide functionality common to a whole class of applications. While one library will usually provide one specific piece of functionality, frameworks will offer a broader range which are all often used by one type of application.”[7]

Consequentialy, a framework in the context of computer science is definable with a set of attributes. A framework

- has a domain,
- is build up of a set of code or libraries,
- is reusable,
- provides and demands reusable design,
- and supports the development of an application in its certain domain.

If web-application development is the domain, the framework is called ”web-application framework”.

2.1.2 Language-related technology setup

There are several relations between programming languages and technologies. An example in the web-programming domain is the markup language HTML. The language leads to technologies or complete technology sets such as defined in the HTML5 specification [33]. HTML5 includes many different technologies and JavaScript APIs like XMLHttpRequest, Drag and Drop or Canvas. But there are also some technologies for similar issues like Web storage and IndexedDB. XMLHttpRequest is used for client-server communication, while Drag and Drop is used to improve user interfaces and Canvas is used to draw graphics. Both, Web storage and IndexedDB help to store larger amounts of data at the client side of the application.

Mathematically spoken, there is a set L of all software languages and a set T of all software technologies. We select two subsets $W_L \subseteq L$ and $W_T \subseteq T$. W_L contains all software languages used for web programming or used with web-programming technologies. W_T contains all web-programming technologies[1]. A language $l \in W_L$ relates to a web-programming setup $S_l \subseteq W_T$. That means, that a specific requirement of a web application written in the language l is resolvable with the technology $t \in S_l$. For example, client-server communication in HTML-based web applications can be handled with the XMLHttpRequest technology. We consider the requirement dependency between a language and its technologies as relation R_l. S_l is the range of R_l. A technology t itself possibly also relates to other languages W_{L_t} including the initial language. For example, XMLHttpRequest also requires JavaScript within the application.

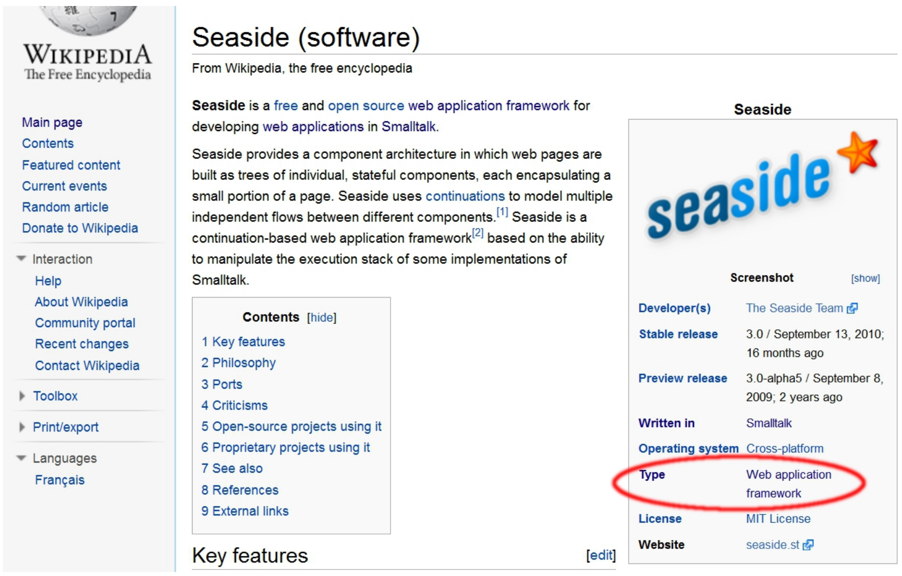
$$R_l \subseteq W_L \times W_T$$
$$R_l = \{(l, t_1), (l, t_2), ..., (l, t_n)\}$$
$$R_L = \bigcup_{l \in W_L} R_l$$
$$S_l = range(R_l)$$

2.1.3 Web-application framework setup

The internet encyclopedia Wikipedia provides a structured system for software categories[2]. The idea is to use this system for identifiing a key category of web-programming technologies. The infobox included in many software pages of Wikipedia contains a field ”Type”,

[1]We never want to claim, that we are able to collect all web-programming technologies
[2]http://en.wikipedia.org/wiki/List_of_software_categories

which refers to such a category or a set of categories (see Fig. 2.1.3). It is possible to identify the most frequently used categories for the type within a set of Wikipedia pages. The challenge is to find a reasonable set of pages related to web programming.



Seaside (software)

From Wikipedia, the free encyclopedia

Seaside is a free and open source web application framework for developing web applications in Smalltalk.

Seaside provides a component architecture in which web pages are built as trees of individual, stateful components, each encapsulating a small portion of a page. Seaside uses continuations to model multiple independent flows between different components.[1] Seaside is a continuation-based web application framework[2] based on the ability to manipulate the execution stack of some implementations of Smalltalk.

Contents [hide]

1 Key features
2 Philosophy
3 Ports
4 Criticisms
5 Open-source projects using it
6 Proprietary projects using it
7 See also
8 References
9 External links

Key features [edit]

Seaside

Screenshot	[show]
Developer(s)	The Seaside Team
Stable release	3.0 / September 13, 2010; 16 months ago
Preview release	3.0-alpha5 / September 8, 2009; 2 years ago
Written in	Smalltalk
Operating system	Cross-platform
Type	Web application framework
License	MIT License
Website	seaside.st

Figure 2.1: Type annotation of the Seaside Wikipedia page

Our research domain is "web programming". The Wikipedia page for "web programming" is redirected to the "web development" page. Hence, we use the page for "web development"[3] as initial point for the identification of the set of pages. The Wikipedia API helps us to get a set of all pages within Wikipedia, which are referenced by "web development". We extend the set by all pages referenced by an item within the set. Hence, the maximum distance d between the "web development" page and the collected pages is $d = 2$. The result contains 9076 items[4]. An infobox is included by 551 of them.

The highest ranked categories are "web browser" and "web-application framework" (see Tab. 2.1 and Fig. 2.2 for the spreading[5]). Web browser are dedicated to present web content [14]. They are definitely the essential component of all web applications. But since we want to focus on different web-programming technologies, we choose "web-application framework" as the key category for further research.

We now are able to collect all web-application frameworks contained in Wikipedia and to rank them. Mathematically spoken, we keep the sets of technologies T and web-programming technologies W_T of the first approach. The set of web-application frameworks W_F is a subset of W_T.

[3]http://en.wikipedia.org/wiki/Web_development
[4]last calculated at 20. Nov. 2011 with Wikipedia API
[5]last calculated at 17. Jan. 2012 using the content of the 9076 Wikipedia pages

Table 2.1: Top 10 results of the type identification

Rank	Type	Frequency
1	web browser	89
2	web application framework	74
3	text editor	29
4	integrated development environment	28
5	html editor	25
6	web server	22
7	rdbms	19
8	application server	18
9	content management system	17
10	ide	16
⋮	⋮	⋮

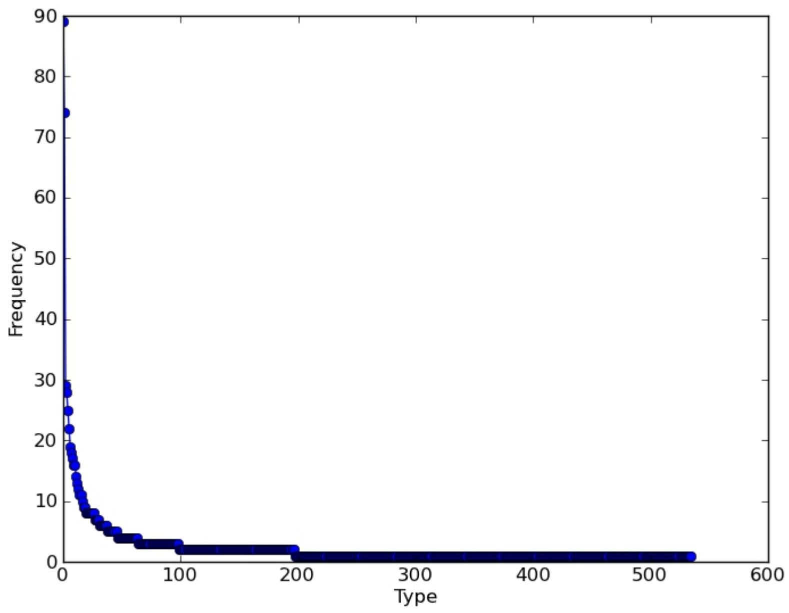


Figure 2.2: Spreading of the type identification

2.1.4 Comparison

There are many differences between the two approaches. The first, language-centered approach leads to a set of incoherent technologies. Apart from the language, there are no criteria connecting the items within the set. We have to identify the characteristics for each technology, before we are able to implement a new web-application with the help of the specific technology. We used HTML, specifically the standard subset HTML5, as an example language for the approach. The advantage of the language is, that it is intrinsically tied to web-programming. Many other web-related languages like Java are also connected to regular programming. If we collect a set of technologies for such a language, it contains a huge amount of items which do not fit to the web theme.

Another way dealing with the first approach is to find a technology for the specific language, which fits to a criterion or requirement, that is previously defined. An example for HTML is "XMLHttpRequest", which fits to the client-server communication requirement. The illustration of the technology leads to other issues. In the first case, we have to code pure HTML and JavaScript. The effort for such an implementation is consequentially high, because all other requirements of 101companies have to be implemented handcrafted. In the second case, we have to search for supporting technologies which are suitable for our example. It is a matter of experience and knowing the right concepts to find good solutions. We need significant manpower to solve the respective issue.

The second web-application framework centered approach leads to a set of coherent technologies. They are connected by a well defined web-related category (see Sec. 2.1.1). A web-application framework combines technologies to solve the basic requirements of web applications. In terms of 101companies, the center requirement is the feature "Web UI"[6]: The application has a user interface presented by the web browser. A web-application framework provides support for generating the presentable content by definition. It also provides a reusable design. The defined design helps to find other supporting technologies to create an implementation. The quality of the implementation definetly also depends on the experience of the developers. But it is easier to search for and learn about new technologies if the developer knows about the names and characteristics.

Concluding, there is also a mathematical difference between the two sets S_l (language-related technologies) and W_F (web-application frameworks). While S_l is an abstract construct, W_F is well defined. That is why we continue with the second approach searching for web-application frameworks. We want to priorize the set W_F to get a sequence of frameworks, ordered by popularity.

[6]`http://101companies.org/index.php/101feature:Web_UI`

2.2 Research question

After choosing the ”web-application framework” category for the identification of web-programming technologies, we face the question, how we are able to priorize these technologies. Our approach is to focus on the popularity of web-application frameworks. It leads to the following questions, which have to be answered in this chapter:

1. What are reasonable methods and tools to get informations about the popularity of web-application frameworks?
2. What are reasonable web-application frameworks for the 101companies project?

2.2.1 Definition: popular

Before we are able to talk about ”popular web-application frameworks”, we have to define the meaning of the quality assessment ”popular”. We consider a ”popular” technology as a widely used technology in the specific domain of programming. Hence, it is proven, used and accepted by developers. In our case, a ”popular web-application framework” is widely used in the domain of web programming.

2.3 Research Method

The initial idea is to use group intelligence to determine the most popular web-application frameworks. As we have illustrated in the motivation (see Sec. 1.1), Wikipedia provides a large amount of articles concerning web development and web-application frameworks with the benefit of a categorization, created and maintained by many different people. The assumption is, that if there is a well developed article about a web-application framework, the technology is highly used by many people. The result defines the important web-application frameworks for the 101companies project.

We use the Wikipedia categorization to get a full list of the web-application frameworks contained in the encyclopedia. The frameworks are extracted directly from the category[7] and template pages[8]. The category ”web-application framework” includes some useful subcategories: ajax-, python-, psf- and rich-internet application frameworks. The result of all the pages contains 146 different framework names with 15 different main categories[9] (see Tab. 2.2). We assigned the main category to each framework manually. If the Wikipedia page and the official page of the specific framework confirms that the framework is a web-application framework or any of its subcategories, we pick that category.

We made the decision to filter out all categories except *web-application frameworks* and *Ajax frameworks*. We include *Ajax frameworks* because the domain is nearly identical with *web-application frameworks*, except that the communication between client and server is performed via *Ajax*. We exclude categories like electronic commerce, because such frameworks do not fit to our 101companies use cases. The focus of content-management systems

[7]`http://en.wikipedia.org/wiki/Category:Web_application_frameworks`

[8]`http://en.wikipedia.org/wiki/Template:Application_frameworks`

[9]last calculated at 20. Nov. 2011 with category and template page for web-application frameworks

Table 2.2: Main categories of the web-application framework list elements

Rank	Main category	Frequency
1	web application framework	93
2	javascript library	14
3	content management system	12
4	ajax framework	11
5	application server	3
6	multiple phone web based application framework	2
	content management framework	2
	software development platform	2
10	rich internet application framework	1
	electronic commerce	1
	content repository	1
	webserver	1
	development environment	1
	library	1

(CMS) often leads to social communities, although they are web-application frameworks, too. We decided to exclude content-management systems because it leads to another field of research. There are some interesting CMS, which should definitely be analyzed in the future, for example TYPO3. We also excluded JavaScript libraries because they only support small specific parts concerning web-application development.

The result contains 103 remaining framework names (see Tab. 2.3, 2.4). They are presented using three columns. The first column shows the name of the framework, the second one shows the main category and the third one shows the mainly used languages for this particular framework. We extracted the languages manually for each framework by using the Wikipedia page for ”comparison of web application frameworks”[10] and the official framework pages for each framework.

There are two different approaches, how to continue with the list to get informations about the popularity of these frameworks:

counting occurences	count all occurences within a given set of webpages
counting backlinks	count all backlinks of the framework-page within all wikipedia pages

The result of both approaches is presented as descending lists dependent on the number of occurence of the framework name.

2.3.1 Counting occurences

The idea about *counting occurences* is to make a conclusion about the relevancy of a framework based on a *superior concept*. We collect a set of pages S and count the occurences of

[10]http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

Table 2.3: Framework-List extracted from Wikipedia (Part 1)

Language	Category	Language
AIDA/Web	Web-application framework	Smalltalk
Ajax4jsf	Ajax framework	Java
Apache MyFaces	Web-application framework	Java
Apache Struts	Web-application framework	Java
Apache Tapestry	Web-application framework	Java
Apache Wicket	Web-application framework	Java
Appcelerator Titanium	Web-application framework	JavaScript
AppFlower	Web-application framework	PHP
AppFuse	Web-application framework	Java
Apple iAd Producer	Web-application framework	JavaScript
ASP.NET	Web-application framework	.NET
ASP.NET Dynamic Data	Web-application framework	.NET
ASP.NET MVC	Web-application framework	.NET
Base One Foundation Component Library	Web-application framework	.NET
Bindows	Ajax framework	XML, JavaScript
CakePHP	Web-application framework	PHP
Camping	Web-application framework	Ruby
Cappuccino	Web-application framework	Objective-J
Catalyst	Web-application framework	Perl
CherryPy	Web-application framework	Python
CL-HTTP	Web-application framework	Common Lisp
CodeIgniter	Web-application framework	PHP
ColdFusion on Wheels	Web-application framework	CFML
ColdSpring Framework	Web-application framework	CFML
CppCMS	Web-application framework	C++
Dancer	Web-application framework	Perl
Django	Web-application framework	Python
DWR	Ajax framework	Java
Flask	Web-application framework	Python
FormEngine	Web-application framework	Java
Fusebox	Web-application framework	CFML, PHP
Genesis Smart Client Framework	Web-application framework	.NET, C Sharp
Gianduia	Web-application framework	JavaScript
Google Closure Tools	Ajax framework	JavaScript
Google Web Toolkit	Ajax framework	Java
Grails	Web-application framework	Groovy
Grok	Web-application framework	Python
Hobo	Web-application framework	DRYML, XML, Ruby
Horde	Web-application framework	PHP
ICEfaces	Ajax framework	Java
ItsNat	Ajax framework	Java
JavaServer Faces	Web-application framework	Java
JBoss Seam	Web-application framework	Java
Jspx-bay	Web-application framework	Java
Kepler	Web-application framework	Lua
Lift	Web-application framework	Scala
Lithium	Web-application framework	PHP
Mach-II	Web-application framework	CFML
Makumba	Web-application framework	Java
Mason	Web-application framework	Perl
Maypole framework	Web-application framework	Perl
Merb	Web-application framework	Ruby

Table 2.4: Framework-List extracted from Wikipedia (Part 2)

Language	**Category**	**Language**
Microsoft Silverlight	Web-application framework	C Sharp
Model-Glue	Web-application framework	CFML
MonoRail	Web-application framework	.NET
Moonlight	Web-application framework	C Sharp
MyFaces Trinidad	Web-application framework	Java
Nagare	Web-application framework	Python
Nevow	Web-application framework	Python
Nitro	Web-application framework	Ruby
onTap	Web-application framework	CFML
OpenACS	Web-application framework	Tcl
OpenRasta	Web-application framework	.NET
OpenXava	Web-application framework	Java
Oracle ADF	Web-application framework	Java
Play Framework	Web-application framework	Java, Scala, Groovy, Python
Pyjamas	Web-application framework	Python
Pylons	Web-application framework	Python
Pyramid	Web-application framework	Python
Python Paste	Web-application framework	Python
Qcodo	Web-application framework	PHP
Quicknet	Ajax framework	JavaScript
Quixote	Web-application framework	Python
Ramaze	Web-application framework	Ruby
RapidSMS	Web-application framework	Python
Reasonable Server Faces	Web-application framework	Java
Rich Ajax Platform	Web-application framework	Java
RichFaces	Ajax framework	Java
Ruby on Rails	Web-application framework	Ruby
Sajax	Ajax framework	JavaScript
Seaside	Web-application framework	Smalltalk
Shale Framework	Web-application framework	Java
Sinatra	Web-application framework	Ruby
Spring Framework	Web-application framework	Java
SproutCore	Web-application framework	JavaScript
Spry framework	Ajax framework	JavaScript
Stripes	Web-application framework	Java
SymbolicWeb	Web-application framework	Common Lisp
Symfony	Web-application framework	PHP
TurboGears	Web-application framework	Python
UnCommon Web	Web-application framework	Common Lisp
Vaadin	Web-application framework	Java
web2py	Web-application framework	Python
Weblocks	Web-application framework	Common Lisp
Webware for Python	Web-application framework	Python
WebWork	Web-application framework	Java
Wt	Web-application framework	C++
xajax	Ajax framework	PHP, JavaScript
Yii	Web-application framework	PHP
Zend Framework	Web-application framework	PHP
Zeta Components	Web-application framework	PHP
ZK	Web-application framework	Java, ZUML

the framework names once per page. The characteristics of the set of pages depends on the initial page, which is identified by this *superior concept*. It depends on the maximum distance m of the method, if the other pages in the set are referenced directly or indirectly by the initial page. If the maximum distance m is 1, the set contains only directly referenced pages. The $distance$ function calculates the value for each $p \in S$:

$$\forall p \in S : distance(p) \leq m$$

In our case, the *superior concept* is WEB DEVELOPMENT. The algorithm collects all wikipedia pages referenced by the WEB DEVELOPMENT wikipedia-page[11]. The result is sorted by quantity. The set S of the concerning web pages is collected via the Wikipedia API[12]. We selected distance 2 for the algorithm.

2.3.2 Counting backlinks

The idea about *counting backlinks* is to make a conclusion about the relevancy of a framework based on its occurences within all wikipedia pages. It means, in comparison to the first approach, that all pages within Wikipedia represent the set of pages S. The Wikipedia API helps to determine all backlinks for a given page, so that we do not need to collect all wikipedia pages. Our scripts run through the web-application framework pages and counts the backlinks leading to the specific pages.

2.4 Results

The tables[13] 2.5 and 2.6 show the ranked lists for each of the two approaches. The results are presented as tables with the following three columns:

rank	the rank of the observed framework
framework name	the name of the observed framework
frequency	the occurences of the framework name within the specific set of Wikipedia pages dependent on the approach

In addition, we have created tag clouds (see Fig. 2.3 and 2.4) based on the frequency of each framework.

2.4.1 Measurements

There is a certain amount of concordance between the two results. Most of the highly frequent framework names occur on both tables in similar intervals. Google Web Toolkit and ASP.NET take the lead. Apache Struts, Pyjamas and Ruby on Rails also occur on both tables in more or less different positions between the top 10. Some of them manifest highly different results like Microsoft silverlight. The issue is discussed in the following interpretation. The following tables present the comparison between the first ten results of each approach:

[11] http://en.wikipedia.org/wiki/Web_Development
[12] http://www.mediawiki.org/wiki/API:Main_page
[13] last calculated at 20. October 2011 with the Wikipedia API

Table 2.5: Framework frequency based on web development

Rank	Freq.	Framework	Rank	Freq.	Framework
1	20	ASP.NET	40	11	OpenXava
	20	Google Web Toolkit		11	SymbolicWeb
3	18	SproutCore		11	ColdSpring Framework
4	16	Pyjamas		11	Mach-II
5	15	Ruby on Rails		11	UnCommon Web
	15	Apache Struts		11	MonoRail
7	13	ASP.NET Dynamic Data		11	Model-Glue
	13	Dancer		11	OpenRasta
	13	Catalyst		11	ICEfaces
	13	Seaside		11	ColdFusion on Wheels
	13	Grails		11	OpenACS
	13	Apache Wicket		11	CL-HTTP
	13	AIDA/Web		11	Reasonable Server Faces
	13	Apache Tapestry		11	Lift
15	12	Horde		11	ItsNat
	12	Camping		11	Fusebox
	12	Nevow		11	Zend Framework
	12	Mason		11	WebWork
	12	Sinatra		11	Wt
	12	Lithium		11	CppCMS
	12	JBoss Seam		11	Makumba
	12	TurboGears		11	BFC
	12	CodeIgniter		11	Stripes
	12	Microsoft Silverlight		11	Weblocks
	12	Maypole framework		11	Play Framework
	12	Ramaze		11	Jspx-bay
	12	Flask		11	Vaadin
	12	Hobo		11	AppFlower
	12	Qcodo	68	10	ZK
	12	JavaServer Faces		10	Cappuccino
	12	Kepler	70	8	Moonlight
	12	CakePHP	71	5	Spry framework
	12	Nitro	72	2	Google Closure Tools
	12	Symfony		2	Apache MyFaces
	12	Spring Framework		2	Django
	12	Merb	75	1	DWR
37	11	Zeta Components		1	CherryPy
	11	AppFuse		1	Apple iAd Producer
	11	Yii		1	Quixote

Table 2.6: Framework frequency based on backlinks

Rank	Freq.	Framework	Rank	Freq.	Framework
1	651	Microsoft Silverlight	52	147	Zeta Components
2	545	ASP.NET	53	146	WebWork
3	524	Google Web Toolkit		146	Vaadin
4	401	Ruby on Rails	55	145	Lift
5	328	Apache Struts		145	Stripes
6	263	Pyjamas		145	Play Framework
7	258	Apache Wicket	58	144	ICEfaces
8	247	Apache Tapestry		144	OpenXava
9	221	Spring Framework	60	143	Nevow
10	215	ASP.NET Dynamic Data	61	142	AIDA/Web
11	212	Google Closure Tools		142	Wt
12	209	SproutCore	63	141	ItsNat
13	194	JavaServer Faces		141	AppFlower
14	191	Horde		141	MonoRail
15	188	Catalyst	66	139	Makumba
16	181	UnCommon Web		139	CppCMS
	181	Zend Framework		139	AppFuse
18	180	CL-HTTP		139	OpenRasta
	180	Weblocks		139	Jspx-bay
	180	SymbolicWeb	71	138	Reasonable Server Faces
21	176	Sinatra	72	118	Moonlight
22	174	Merb	73	114	Apache MyFaces
23	172	Camping	74	93	Cappuccino
	172	Symfony	75	68	Spry framework
25	171	CakePHP	76	55	Apple iAd Producer
	171	Nitro	77	27	CherryPy
27	169	Mason	78	21	Django
	169	Ramaze	79	17	Quixote
29	168	Hobo	80	12	RichFaces
30	167	TurboGears	81	9	ASP.NET MVC
31	165	Dancer	82	8	Appcelerator Titanium
	165	Maypole framework	83	6	Pyramid
33	164	web2py	84	5	Grok
	164	Kepler		5	Shale Framework
35	159	CodeIgniter		5	Pylons
36	158	Grails		5	Python Paste
37	157	Flask	88	4	DWR
	157	Seaside	89	3	FormEngine
	157	JBoss Seam		3	RapidSMS
40	155	Yii		3	Webware for Python
41	153	Fusebox		3	xajax
42	152	Mach-II	93	2	Ajax4jsf
	152	ColdFusion on Wheels		2	MyFaces Trinidad
44	151	Qcodo		2	Nagare
45	150	OpenACS		2	Quicknet
46	149	Model-Glue		2	Sajax
	149	ZK	98	1	Gianduia
	149	Lithium		1	Bindows
49	148	onTap		1	Rich Ajax Platform
	148	BFC	101	0	Genesis Smart Client Framework
51	147	ColdSpring Framework			

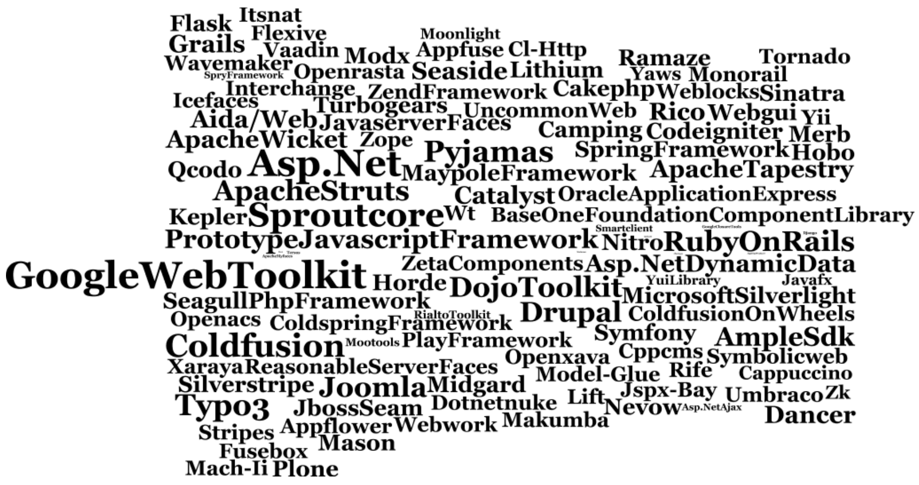


Figure 2.3: tag cloud of the web development based frequency

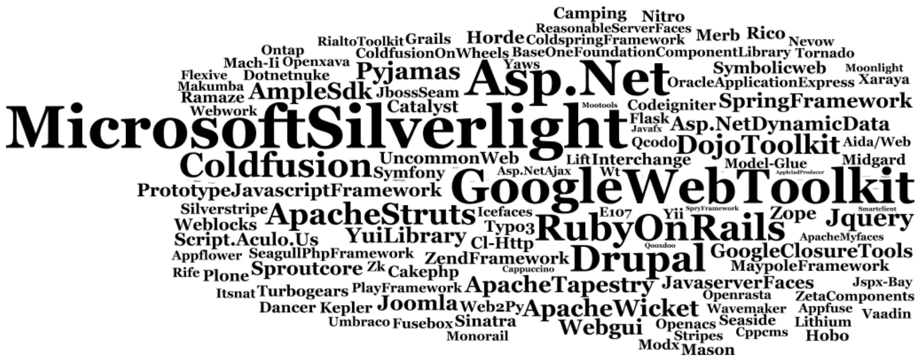


Figure 2.4: tag clound of the backlinks frequency

Table 2.7: Top 10 web development based frequency compared to backlink frequency

	Web development based frequency		Backlink frequency	
	Framework	Frequency	Position	Frequency
1	ASP.NET	20	2	545
	Google Web Toolkit	20	3	524
3	SproutCore	18	12	209
4	Pyjamas	16	6	263
5	Ruby on Rails	15	4	401
	Apache Struts	15	5	328
7	ASP.NET Dynamic Data	13	10	215
	Dancer	13	31	165
	Catalyst	13	15	188
	Seaside	13	37	157

2.4.2 Interpretation

As we have mentioned before, there are several results out of alignment. The best example is Microsoft Silverlight. It is on top of the backlink frequency-list but it is only on the fifteens rank at the web-development based frequency-list. The distance between the first and the following ranks is significant in both frequency lists. In this particular case, the comparison between the first and the fourteenth rank leads to following result:

- *Web development based frequency:*

$$\frac{f_{15}}{f_1} = \frac{12}{20} = 0.6 = 60\%$$

- *Backlink frequency:*

$$\frac{f_{15}}{f_1} = \frac{188}{651} = 0.29 = 29\%$$

The fourteenth rank of the web development frequency list is at 60% of the first rank. The fourteenth rank of the backlink frequency list is at 29% of the first rank. The percentage shows, that the frequencies of the web-development based frequency list are considerably closer to each other. The values itself are also very small. The conclusion is, that the frequency of the list is less significant for us because small changes within the values cause great change in ranks. That is why a framework is not mandatory on the first rank of the web development based frequency list if it is on the first rank of the backlink frequency list.

There is also an expectable growth in frequency of the web-development based list if we raise the recursion depth of the corresponding algorithm. At some point, the collected set of pages is mostly equal to the complete page set of Wikipedia. The only pages not in that list are those, which do not refer to or are not refered by other pages within Wikipedia.

Table 2.8: Top 10 backlink frequency compared to web development based frequency

	Backlink frequency		Web development based frequency	
	Framework name	Frequency	Position	Frequency
1	Microsoft Silverlight	651	15	12
2	ASP.NET	545	1	20
3	Google Web Toolkit	524	1	20
4	Ruby on Rails	401	5	15
5	Apache Struts	328	5	15
6	Pyjamas	263	4	16
7	Apache Wicket	258	7	13
8	Apache Tapestry	247	7	13
9	Spring Framework	221	15	12
10	ASP.NET Dynamic Data	215	7	13

Our results lead to the following idea: The high ranked web-application frameworks of the backlink frequency-list are picked up and compared to the web development based frequency-list. If the selected framework has its place in the web-development related list, the framework is considered as important. Since we are not able to implement every listed framework, we selected the following seven:

Table 2.9: Selected frameworks

Framework name	Backlink rank	Web development rank
Microsoft Silverlight	1	15
Google Web Toolkit	3	1
Apache Struts	5	5
Pyjamas	6	4
JavaServer Faces	13	15
Zend Framework	16	41
JBoss Seam	37	15

Silverlight, Google Web Toolkit, Apache Struts and Pyjamas are chosen because they are top 10 frameworks. JavaServer Faces is chosen because the framework is used for many other frameworks like Spring, JBoss Seam or Apache Struts as a basic view-controller technology. Zend is one of the two highest rankend PHP frameworks. And JBoss Seam is a good example for an application server related framework.

2.5 Threats to validity

We use Wikipedia for the identification of useful web-programming technologies. Since it is involved in many parts of the technology-identification process, we have to discuss its trustworthyness. The term ”web-application framework” or ”web framework” is very common in scientific literature. It is described as a reliable concept of technology for rapid web-application development[8]. The concept is subject to scientific comparisons [31, 6, 37] as well as it is an important background for new development approaches [21, 5]. It is also a well definable (see Sec. 2.1.1) term. The conclusion is, that Wikipedia leads to a research subject, which is useful and well known.

Nevertheless, it is difficult to guarantee, that every technology page in Wikipedia related to ”web development” has a type or is categorized in a suitable manner. In fact, there are even examples for web-application frameworks, which are undercategorized[14] or overcategorized[15]. Both issues complicate the assignment of a technology to its main categories, especially when the only applied category is weak defined. The same problem occurs with the type contained in the infobox of software technologies. The threat applies to both processes, is it the research subject identification or the web-application framework identification.

We also do not claim, that every existing high frequently used web-application framework is documented in Wikipedia. Our process of identification is just an instrument to simplify the search for suitable web-application frameworks. Other interesting new web-programming technologies like Web storage [34] or IndexedDB [36] are also not included in our framework implementations. Hence, our approach will never highlight all useful concepts and technologies.

2.5.1 Tools

The tools we have used for Wikipedia-page analysis are mostly selfmade. During the development, we faced several problems indentifiing categories and types:

- Different links in Wikipedia redirect to the same category page.
- There are different capitalizations for same category names and different uses of hyphens.
- Types in the infobox are seperated by either comma or by HTML tag (”$< br >$”).
- Types are sometimes links to other Wikipedia pages, sometimes normal text and sometimes marked with citation links.

The list of problems is verified with the assistance of samples. We randomly chose items out of the web-development related web-pages and examine their content. The problems listed above are handled by our scripts.

2.5.2 Implementations

Every web-application framework we have included in the 101companies project is additionally verified by using the official pages and documentations. We used tutorials found in the internet to get familiar with the specific technology. A major part of the tutorials does not include every aspect of the described technology. One reason is technological evolution. JSF, for example, is historically connected to JSP, but today the JSP technology

[14] `http://en.wikipedia.org/wiki/Oracle_Application_Express`
[15] `http://en.wikipedia.org/wiki/Wavemaker`

has been replaced by Facelets [30]. Another reason is, that JSF also covers a very broad list of features like Ajax support or different scopes of Java Beans. It would overwhelm the developer to put all the possibilities of a web-application framework into a tutorial dedicated to the basics.

Chapter 3

Related work

3.1 Taxonomies and classifications

There are some reasonable alternatives to gain taxonomies and classifications of web-programming technologies. An approach is to survey web-technologies covering different factors of web programming. The factors include global distribution [8] of the web itself, interactivity of a web application and at last cultural and social environment concerning users and developers. Each factor leads to different objectives like quality attributes, architectural styles or user-interface design. Web-programming technologies can be classified and compared by using these objectives. Since web programming is a huge field of countless technologies and concepts, the ”Survey of Technologies for Web Application Development”, written by Barry Doyle and Cristina Videira Lopez, concludes, that there is still a lack of a solid domain model for web programming. Our approach is to priorize web-programming technologies in order to get an overview of the most used and most important components related to the domain.

Many approaches compare and classify web-application frameworks. The diploma thesis of Andreas Wende, ”Klassifikation und Bewertung von Frameworks für die Entwicklung von Web-Anwendungen” [37] describes different basic concepts for web-application frameworks to get a strong classification system. The system introduces criteria such as architecture, application control, user-interface components, interface generation, data binding and data exchange, dialog control, validation and session management. The approach illustrates the classification of the frameworks using implementations. We flip that approach and use implementations to get classifications for web-programming technologies.

One of the most common languages used in the web-programming domain is Java. Hence, there are many web-application frameworks based on that language. The ”Taxonomy for Java Wep-application frameworks” [31], written by Tony C. Shan and Winnie W. Hua defines five schools for Java EE based web-application frameworks. Each Java framework is related to a category like *request-based*, *component-based*, *hybrid*, *meta* and *RIA-based*. The schools describe the communication characteristics of each framework in particular. For example, a *request-based* framework like Apache Struts communicates directly via controllers and actions while a *component-based* framework like JSF encapsulates the request handling into reusable components.

Another example of a web-application framework taxonomy related to Java is the thesis of Ian F. Darwin about ”Java Web MVC Frameworks: Background, Taxonomy, and Examples” [6]. It uses the components of MVC architecture for classification: model, view and controller. Each regarded framework can be assigned to one or more of these components.

All the approaches illustrate, that there are several classification possibilities about web-programming technologies, web-application frameworks in particular. We want to merge the approaches with the help of the 101companies project. Our collected web-application frameworks lead to different languages, architectural characteristics, communication styles, interface-generation methods and also to many different web-programming technologies like libraries, APIs and other frameworks. The approaches also illustrate, that our focus on web-application frameworks is reasonable.

3.2 Building the taxonomy

Building a taxonomy is a challenging process. The major task is to identify reasonable objects and characteristics for the taxonomy. The approach of Robert C. Nickerson, ”Taxonomy development in information systems: Developing a taxonomy of mobile applications” [25], illustrates a defined method for generating taxonomies. The method is build up on a cyclic process divided into three iterations. Objects are empirically identified and distinguished in the first iteration. The result is extended by new characteristics and dimensions in the second iteration. Finally, missing objects are created to complete the taxonomy and close possible gaps.

Our approach describes the empirical identification of web-programming technologies and their concepts. Hence, it is highly related to the first iteration of the process. The result is a set of technologies and concepts building the first taxonomy. Each concept leads to further objects and classifications. A good example is client-server communication. Some of our first implementations illustrate *Ajax* as a concept for data exchange between client and server. We used a pull-based approach, but there is also a push-based approach called *Reverse-Ajax* or *Comet* [2]. In addition, there are interesting conceptual alternatives for data exchange like *REST* and *SOAP* [27]. Last but not least, Ajax leads to new architectural patterns, for example *SPIAR* [20]. All the new concepts, technologies and architectures build up new classifications and dimensions for our taxonomy, although we do not have implementations for each of the categories. Searching for alternative concepts and technologies is related to the second iteration of the taxonomy-building process.

Another approach for creating taxonomies is to use a previously developed taxonomy as source. Such sources can be verified, changed and extended. As we have seen, Wikipedia offers a huge amount of classifications for many domains and, as follows, provides a good initial point for searching such sources. For example, there is an approach to get large scale taxonomies by using all categories of Wikipedia named ”Taxonomy induction based on a collaboratively built knowledge repository” by Simone Ponzetto and Michael Strube [28]. It considers the category system as a *thematically organized thesaurus*. Since Wikipedia includes many software categories[1], the Wikipedia thesaurus helps us to gain access to the web-programming domain.

[1] http://en.wikipedia.org/wiki/List_of_software_categories

3.3 Web-application frameworks

There are also other more exotic but interesting approaches for web-application frameworks. Flapjax [21] and Links [5] are language-framework systems directly designed to create web-applictions. While Flapjax is an event based programming language for generating JavaScript-Ajax web appilcations, Links simplifies three-tier architecture based web-applications by using only one language for every tier. Both approaches illustrate reasonable architectural and structural alternatives for web programming and therefore should be included into the 101companies system.

3.4 Architecture

Since we have used the *model view controller* (MVC) pattern in many implementations, it is interesting to take a deeper look into the concept. MVC is one of the most commonly used architectures in web development, but its basics are related to regular programming. The following references help to understand the pattern and its historical development.

Trygve Reenskaug invented MVC in 1979. His first description of the style showed up in a Xerox PARC technical node [32] related to the work of Reenskaug with the object-oriented programming language Smalltalk. Originally, the name was formed by four terms: *thing*, *model*, *view* and *controller*. The *thing* describes the underlying problem, which is solved by the use of the MVC pattern.

Steve Burbeck wrote a revision in the years 1987 and 1992 with the title "Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)" [3]. It is also based on Smalltalk. The architecture is designed to support the development of large scaled, interactive applications with a possibly huge data set in the background. These characteristics fit well to web-applications and applicable frameworks. Espacially web-application frameworks are dedicated to develop such large scale applications with the additional requirement, that the application is web based.

Chapter 4

101companies features

4.1 Features of the 101companies project

Before we want to present the implementation documentations, it is necessary to talk about the features and the feature coverage of each implementation. The 101companies feature model [15] offers defined requirements for the implementations. It is divided into functional, non-functional and user-interface requirements. The mandatory functional requirements specified in the category 101minimum are company, cut and total. These three features provide the basic data structure and basic functionalities for each implementation. Since the main character of web-applications is a user interface presentable by the browser, the web UI is also mandatory for web-programming. The feature is accompanied by the feature navigation. Web applications often fulfil other interesting requirements including client-server communication, data persistence or access control. The following table illustrates the feature coverage of the implementations included in the thesis.

Table 4.1: Feature coverage of the implementations

		html5local	html5XMLHttpRequest	html5indexedDatabase	html5ajax	html5tree	jsf	gwtTree	pyjamas	zend	seam	strutsAnnotation	silverlight
ui	**Web UI**	●	●	●	●	●	●	●	●	●	●	●	●
	Voice control												
	Undo/Redo												
	Touch control												
	Structural editing					●		●					
	Navigation	●	●	●	●	●	●	●	●	●	●	●	●
	Localization												
	Intelligent UI												
	Attribute editing	●			●	●	●	●	●	●			
quality	**Scalability**												
	Reliability												
	Persistence					●	●	●		●	●		
	Data mapping												
	Code generation							●	●				
	Client-server		●		●	●	●	●		●	●	●	●
	Access Control										●		
minimum	**Total**	●	●	●	●	●	●	●	●	●	●	●	●
	Cut	●	●	●	●	●	●	●	●	●	●	●	●
	Company	●	●	●	●	●	●	●	●	●	●	●	●
extra	**Precedence**					●		●					
	Mentoring												
	Logging												
	Import												
	Export												
	Depth												

Chapter 5

Implementations

5.1 *101implementation* html5local

5.1.1 Intent

Web programming based on the HTML5 ecosystem with local Web storage

5.1.2 Languages

- HTML5
- JavaScript
- JSON

5.1.3 Technologies

- Web storage
- Web browser

5.1.4 Features

- Company
- Cut
- Total
- Navigation
- Attribute editing
- Web UI

5.1.5 Motivation

This implementation illustrates the use of client side data storage within a web application. While most other web applications only store the requested data for the view in the web browsers' cache, the HTML5 integrated JavaScript API for Web storage offers the possibility to permanently store more complex data as *key-value pairs* on the client-side. While this implementation offers session independent storage, there is a corresponding implementation *101implementation* html5session providing session storage.

5.1.6 Illustration

The illustration section is devided into two parts. The first section shows the method of data storage, the second section shows the implementation of the Total. Most of the web application is written in JavaScript.

Web storage

The Web storage API provides the `localStorage` and the `sessionStorage` objects. This application uses `localStorage`, at which the use of `sessionStorage` in *101implementation* html5session is completely the same. We initialize our own variable `storageObject` in order to keep the possibility for a simple exchange of the storage principe:

```
var storageObject = localStorage;
```

The value of the key-value pairs provided by web storage have to be primitive data types or a string. Since JavaScript provides a great JSON support, we store the company data as JSON encoded string within the web storage. But first we have to create the company objects. Since JavaScript does not provide classes, there is a possibility to emulate classes with functions. The example shows one of these emulated classes for the company:

Listing 5.1: company.js

```
function Company(id, name) {
    this.id = id;
    this.name = name;
    this.departments = new Array();
}
```

After providing the functions for each company, department and employee, we are now able to instantiate the company structure:

Listing 5.2: company.js

```
function loadData(reset) {
    // If the company does not exist, it must be created
        locally.
    if (storageObject.company == null || reset == true)
        {
        // create company ...
        var company = new Company(0, "Meganalysis");
        ...

        // save company
        storageObject.setItem('company', JSON.stringify(
            company));

    }
    return JSON.parse(storageObject.getItem('company'));
}
```

The `setItem('company', JSON.stringify(company))` call encodes the company as JSON string and stores it to a variable `company` within our `storageObject`. If this `company` variable already exists, the method simply returns the stored company. To get the company string, the `getItem('company')` method has to be invoked on the `storageObject`.

Feature implementation

The data-structure is a tree, which can be traversed to cut salaries or determine the total of the whole company or individual departments. The following example shows the method for totalling companies with the help of totalling departments:

Listing 5.3: companyModel.js

```
// This method calculates the total for a company.
function totalCompany(company) {
    var total = 0;
    var len = company.departments.length;
    for (var i = 0; i < len; i++) {
        // To get the total for the company, this method
            calls the total method for departments and
            adds the results.
        total += totalDepartment(company.departments[i])
            ;
    }
    return total;
}

// This method calculates the total value for
    departments.
function totalDepartment(department) {
    var total = 0;
    var len = department.subdepartments.length;
    // Here, the total values of all subdepartments are
        added recursively.
    for (var i = 0; i < len; i++) {
        total += totalDepartment(department.
            subdepartments[i]);
    }
    var lenEmp = department.employees.length;
    // Here, the salaries of all contained employees are
        added.
    for (var i = 0; i < lenEmp; i++) {
        total += department.employees[i].salary;
    }
    return total;
}
```

The function `totalCompany` uses the function `totalDepartment`, to get the values of its subdepartments. This total values are calculated recursively.

GUI generation

The HTML code for the GUI is completely generated with JavaScript. The following code generates the name textfield for the company view:

Listing 5.4: companyView.js

```
...

// Create a form for the company view.
var content = "<form name=\"company\">";

// Create a table within the company view.
content += "<table border=0>";

// Create a table row with the name textfield and a save
    button.
content += "<tr><td align=\"right\">Name: </td>";
content += "<td><input type=\"text\""
    + " class=\"text\" name=\"name\" value='"
    + model.headline
    + "'>";
content += " <input type=\"button\" name=\"save\""
    + " class=\"button\" value=\"save\""
    + " onclick=\"controller.changeName(this.form.name.
        value)\">"
    + "</td></tr>";
...

// Close table and form tags.
content += "</table>";
content += "</form>";

// Move the content to the 'content' division of the
    company.html file.
document.querySelector('#content').innerHTML = content;

...
```

The first step is to create an HTML form (`<form name="company">`) including the table as a grid for proper alignment of the GUI components. The table row starting with the `<tr>` tag contains the name label, the name textfield and the save button. The textfield (the first `<input ... />`) is initialized with the value `model.headline`, which contains the name of the company. The save button (second `<input ... />`) invokes the method `changeName`, where the `this.form.name.value` parameter contains the current content of the name textfield. Finally, the last allocation (`... = content;`) moves the generated HTML code to the *content* division of the document company.html.

5.1.7 Architecture

All files of the application are contained in one folder:

- There is an initial HTML page for each company entity of the company: company.html, department.html and employee.html. The content is generated by JavaScript. Both, the HTML pages and companyView.js, departmentView.js and employeeView.js form the view of this MVC based implementation.
- The controller (the files companyController.js, departmentController.js and employeeController.js) passes the data to the model and refreshes the GUI. It is very lightweight, based on the character and limited use of the features and on the pure client side implementation.
- The model (companyModel.js, departmentModel.js, employeeModel.js) handles the access to the Web storage.
- The company is initialized in company.js.

5.1.8 Usage

- Download the complete folder content.
- Open the local index.html with your web browser.

5.2 *101implementation* html5indexedDatabase

5.2.1 Intent

Web programming with indexedDB based on the HTML5 ecosystem

5.2.2 Languages

- HTML5
- JavaScript

5.2.3 Technologies

- IndexedDB
- Web browser

5.2.4 Features

- Company
- Cut
- Total
- Navigation
- Web UI

5.2.5 Motivation

The IndexedDB API is a good alternative solution for persisting data on the client side of a web application. While the Web storage API uses *key-value pairs*, the IndexedDB API uses indexed tables represented by a *B*-tree structure for data storage. This is an applicable approach for larger amounts of data. Since the synchronous API is not fully implemented yet, we used the asynchronous API.

5.2.6 Illustration

The communication between the application and the indexedDB is transaction based. But before using any transactions to gain access to the data we have to create the database and the data within. This section will start with the database initialization and data generation and will finish with illustrating a transaction based cut request for the whole company. Please visit *101implementation* html5local for the aspect of GUI generation.

Database initialization

The implementation contains an initial open function for the database creation. If the database initialization is successful, the function also generates tables and data.

Listing 5.5: company.js

```
companies.indexedDB.open = function(f) {

    // opens a connection to the 101Companies database
    // - if it does not exist, create new database
    // - if it exists, use database
    var request = indexedDB.open("101Companies");

    // the database connection is successfully
       established
    request.onsuccess = function(e) {

        ...

    };

    // the database connection is unavailable
    request.onfailure = companies.indexedDB.onerror;
}
```

The `indexedDB.open("101Companies");` function opens the connection to an existing database named *101Companies* or creates a new database with this name, if it does not exist. It returns a request status object, as well. This object allows an asynchronous callback for the two possible results of the *open* function: success or failure. On failure, the `companies.indexedDB.onerror` function returns some user notification. On success, we are able to continue with the table generation:

Listing 5.6: company.js

```
...

// creates a data table for Companies
var companiesStore = db.createObjectStore("Company", {
    keyPath: "id"});

...
```

This simple call creates a table named *Company* with the key *id*. This table is able to store arrays with any fields, except that the field called *id* is used as the key for the b-tree of the table. It returns the table object for further use, as well.

Data generation

Data manipulations are handled with transactions. This is why we have to create transactions to generate the table contents. Each data manipulation consists of three steps:

- create transaction
- retrieve table object
- put data into the table as an array

Listing 5.7: company.js

```javascript
companies.indexedDB.addData = function() {

    // get database
    var db = companies.indexedDB.db;
    // create transaction
    var transComp = db.transaction(["Company"],
        IDBTransaction.READ_WRITE, 0);
    // get company table
    var compStore = transComp.objectStore("Company");

    ...

    // create data object with id = 0
    var compData = {
        "company": "Meganalysis",
        "id": 0
    };

    ...

    // store data object
    compStore.put(compData);

    ...

};
```

We first retrieve the database using the field `companies.indexedDB.db;`. After that, a new transaction to the *Company* table with `READ_WRITE` access has to be created. The table is retrieved with `transComp.objectStore("Company");`. We now are able to read and write data from and into the table.

Feature implementation

The manipulation of data corresponds to its creation except that we need a cursor to get the required arrays out of the b-tree. In case of cutting the company we need all employees:

Listing 5.8: companyModel.js

```javascript
companies.indexedDB.cut = function() {
    // get database
    var db = companies.indexedDB.db;

    // create transaction
    var transEmp = db.transaction(["Employee"],
        IDBTransaction.READ_WRITE, 0);
    // get employee table
    var empStore = transEmp.objectStore("Employee");

    // Key range: Get every single employee in the store
    var keyRange = IDBKeyRange.lowerBound(0);
    // create Cursor with key range
    var cursorRequest = empStore.openCursor(keyRange);

    // cursor runs through the result-set
    cursorRequest.onsuccess = function(e) {
        var result = e.target.result;
        if(!!result == false)
            return;
        // cut the salary
        result.value.salary = result.value.salary / 2;
        // store the employee
        empStore.put(result.value);
        // next employee ...
        result.continue();
    };

    // error handling
    cursorRequest.onerror = companies.indexedDB.onerror;
}
```

The `IDBKeyRange.lowerBound(0);` means, that all keys are greater than 0. That is, in case of employees, every element within the employee b-tree. If the cursor is successfully created (`var cursorRequest = empStore.openCursor(keyRange);`), we are able to run through the results within the `onsuccess` function. The function `e.target.result` delivers the next element. If it exists, we cut the salary and restore it into the database. After that, we continue with the next element at `result.continue();`.

5.2.7 Architecture

The architecture is equal to *101implementation* html5local, except that the model organizes the connection to the indexed database.

5.2.8 Usage

- Check out all files from the repository.
- Open the index.html with your Web browser. This application currently does work only with google chrome.

5.3 *101implementation* html5XMLHttpRequest

5.3.1 Intent

Basic use of XMLHttpRequest with HTML5

5.3.2 Languages

- HTML5
- JavaScript
- PHP
- XML

5.3.3 Technologies

- DOM
- XMLHttpRequest
- Web browser

5.3.4 Features

- Company
- Cut
- Total
- Client-server
- Navigation
- Web UI

5.3.5 Motivation

This implementation provides simple server side XML based data storage. Therefor, it introduces XMLHttpRequest in a very simple way of use. This helps to understand the asynchronous mechanisms of the XMLHttpRequest API. In order to keep it simple, there is no greater Ajax support in this implementation. If you want to see an Ajax based implementation, please visit *101implementation* html5ajax.

5.3.6 Illustration

This section illustrates, how the data moves from an XML file to a DOM based Web application and back. The first section shows the structure of the initial XML file, the second part shows the load mechanism with XMLHttpRequest, the third part shows some data manipulation according to the Cut and the last part shows the save mechanism with XMLHttpRequest. Please visit *101implementation* html5local for the aspect of GUI generation.

XML document structure

The company.xml file represents the company. It models the company structure in typical XML manner: Each entity is represented by a node, at which its parameters or appended entities are represented by subnodes. Each department and employee contains an aditional parameter node for the *id*.

Listing 5.9: company.xml

```
<Company>
    <name>Meganalysis</name>
    <departments>

        ...

    </departments>
</company>
```

This example shows the `company` node with the two subnodes `name` and `departments`. The `departments` node contains all direct subdepartments of the company as a single subnode.

Load company

We are able to access this company.xml file by using a simple XMLHttpRequest. The request itself needs three informations:

- The request method is *GET*, because we only want to load the file,
- the filename is *company.xml*,
- and we want to perform an asynchronous request, announced by the last boolean parameter *true*.

Listing 5.10: company.js

```
company.loadData = function() {

    var xhr = new XMLHttpRequest();

    // This statement creates a new request with the
       parameters:
    // - "GET": only load
    // - "company.xml": filename of the requested xml
        doc
    // - "true": asynchronous request
    xhr.open('GET', 'company.xml', true);

    // This method is triggered after the response
        reaches the client.
    xhr.onload = function(e) {
        if (this.status == 200) {
            // This line guarantess, that the result has
                xml format.
            company.response = xhr.responseXML;
            controller.loadInner();
        }
    };

    // This call starts the request.
    xhr.send();

}
```

The `loadData` function first creates a new XMLHttpRequest object. The three necessary values mentioned before are used as parameters for the `open` function. The `onload` function defines the reaction after finish the file load. Status 200 means, that the file is successfully transfered to the client. The advantage of XML is, that it is extremely easy to load with XMLHttpRequest. The `xhr.responseXML` function returns a complete data structure for the company traversable with DOM.

Feature implementation

We use the DOM API to retrieve all salary nodes of the company:

Listing 5.11: companyModel.js

```
model.cut = function() {
    // This call retrieves all salary nodes.
    var salaryNodes = company.response.documentElement.
        getElementsByTagName("Salary");

    // This loop cuts the salary values by two and saves
        the value to the specific nodes.
    for (var i = 0; i < salaryNodes.length; i++) {
        salaryNodes[i].childNodes[0].nodeValue =
            parseFloat(salaryNodes[i].childNodes[0].
            nodeValue) / 2;
    }

    // This function saves the company to the xml file.
    company.saveData(company.response);
    // The new total value has to be determined after
        the cut.
    model.total();
}
```

The `getElementsByTagName("Salary")` returns all salary nodes for the company. The return value is a simple array. The *for* loop traverses this array and cuts all the salaries. After cutting the salaries, all new values have to be saved and the new total value has to be determined. In our example we will save the complete company with the new data into the company.xml file.

Save company

The save mechanism is as simple as the load mechanism. The difference is, that some parameters of the `open` function have to be changed:

- The request method is now *POST*, because we want to have write access
- and the filename is *update.php*, refering to the PHP script, which accepts the changed content for the company.xml.

Listing 5.12: company.js

```
company.saveData = function(data) {
    var serializer = new XMLSerializer();
    var xml = serializer.serializeToString(data);

    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'upload.php', true);
    xhr.setRequestHeader("X-Requested-With", "
        XMLHttpRequest");
    xhr.setRequestHeader("X-File-Name", "company.xml");
    xhr.setRequestHeader("Content-Type", "application/
        octet-stream");
    xhr.send(xml);
}
```

There are some additional parameters, which are used to define the following proceeding on the server sides PHP script. The three elements of the request header show, that the request is an XMLHttpRequest, that the concerning file has its relative path *company.xml* and the content is a stream. The upload.php script handles the stream and saves it as XML file with the given name:

Listing 5.13: upload.php

```
<?php
    $uploaddir = "";

    if($_SERVER['HTTP_X_FILE_NAME']!="") {

        $nomefile=$_SERVER['HTTP_X_FILE_NAME'];

        $fh = fopen($uploaddir.$nomefile, 'w') or die("<
           h1 style='color:red;'>Upload failed</h1>");

        fwrite($fh, $HTTP_RAW_POST_DATA);

        fclose($fh);

        echo "<h1>success uploaded</h1>.\n";
    }
?>
```

The first *if* control structure proofs, that the filename is not empty. After that, the script opens the file with the given filename and write access. If it is successfully opened, the stream can be written to the file handled by the function `fwrite` The parameters for this function are the opened file (`$fh`) and the delivered content (`$HTTP_RAW_POST_DATA`).

5.3.7 Architecture

All necessary files are located in the base folder. The architecture is based on MVC:

- All HTML files in combination with the JavaScript (.js) files with the suffix *View* represent the view.
- The controller is implemented within the JavaScript (.js) files with the suffix *Controller*.
- The model files have the suffix *Model*.

There is an additional upload.php file, which is necessary to upload new content for the XML file.

5.3.8 Usage

- Please check out all files in the repository.
- Open the index.html with your web-browser (check HTML5 for the HTML5-support of your browser).

This HTML5-program does not work over file-protocol when using Chrome. In this case, you need access over http. To gain access over http, you can use XAMPP, for example, to create a webserver.

- Download XAMPP from http://www.apachefriends.org/en/xampp.html.
- Install XAMPP.
- Deploy all files to your htdocs-directory (for example: E:/xampp/htdocs/xhr/).
- Start the XAMPP-Control Panel and activate Apache.
- Start your web-browser.
- Call http://localhost/xhr/index.html.

5.4 *101implementation* html5ajax

5.4.1 Intent

Web programming based on the HTML5 ecosystem using Ajax style

5.4.2 Languages

- HTML5
- JavaScript
- PHP
- SQL
- JSON

5.4.3 Technologies

- MySQL
- XAMPP
- Apache HTTP Server
- XMLHttpRequest
- Web browser

5.4.4 Features

- Company
- Cut
- Total
- Client-server
- Navigation
- Attribute editing
- Web UI

5.4.5 Motivation

This web application provides an optimized data exchange between the client and the server. It is achieved by the use of the Ajax principle. Optimized data exchange with Ajax means, that only necessary parts of the current page are reloaded and only necessary data are transmitted from the server to the client. This web application uses the XML-HttpRequest API included in HTML5 and is developed without any supporting JavaScript framework.

5.4.6 Illustration

This implementation is MVC and client-server based. In order to create a reasonable illustration of the different layers of this application, this section is geared to this layers. The view, controller and some parts of the model are located on the client side.

Client

The view is mainly HTML based, even though there are JavaScript parts. The following example shows a table row in the HTML5 company.html document, which creates the textfield for the companies' total output and the cut button:

Listing 5.14: company.html

```html
<table>

    ...

    <tr>
        <td>Total:</td>
        <td><input type="text" name="total" class="text"
             readonly="readonly"/></td>
        <td><input type="button" class="button" value="
           cut" onClick="controller.cut()"/></td>
    </tr>

    ...

</table>
```

If the user presses the cut button, the corresponding JavaScript method located in the client side model companyModel.js is invoked through the controller.js. The *initCompany* method, used in *cut*, initializes an object with all necessary data to identify the company on the server side:

Listing 5.15: companyModel.js

```javascript
// cut company
model.cut = function(strategy) {
    model.initCompany();
    model.company.action = "cut";

    model.sendRequest(strategy, model.company);
}
```

The sent request contains a strategy, which defines the reaction after receiving the response. In case of a non error response the strategy refreshes the total field of the company.
The use of JSON instead of XML for request and response messages has a major advantage: The messages are created in an object oriented style and can easily be transformed into JSON strings. There is no additional effort for creating and interpreting complex XML messages. The JSON message for the specific cut request contains the necessary informa-

tion about the action and the entity:

```
{
    "id":1,
    "table":"company",
    "action":"cut"
}
```

The message is received by the server, which cuts the company with the identifier 1.

Server

After receiving the cut request, the server performs the corresponding action within the PHP script companyServer.php. After that, it returns the new information for the company to the client:

Listing 5.16: companyServer.php

```
// --------------------------------------- cut company
function cut($jsonObject) {
    $id = $jsonObject->id;
    $request = "UPDATE employee SET salary = salary / 2
       WHERE cid = $id";
    mysql_query($request);

    return loadCompany($jsonObject);
}
```

The answer is a stringified company object containing all necessary information about the company (and nothing more):

```
{
    "name":"meganalysis",
    "departments":["Research","Development"],
    "total":199873.5
}
```

This application implements the attribute editing feature. Hence, there is also validation. The client side validation is performed within the client side part of the model, while the server side validation is performed within the PHP scripts. If there is a client side input error, no request will be created and the error is displayed directly. If there is a server side error, the PHP script responds with an appropriate JSON message.

5.4.7 Architecture

The architecture is based upon the MVC pattern. While the view (example: company.html, companyView.js) and the controller (controller.js) are parts of the client, the model is part of the server and the client.

- The view is based on pure HTML (see [this!!client/]) and JavaScript. The corresponding JavaScript files (see views) are used to fill the fields of the user interface.

- The model on the server side (see [this!!server/]) is a PHP script and receives the requests and responds in JSON. The model on the client-side (see model) receives the JSON message and refreshes its data (example: companyServer.php, companyModel.js).
- All requests and responses via the XMLHttpRequest API are handled by the function defined in XMLHttpRequest.js.
- The JavaScript based controller (see controller.js) handles the actions invoked by the user and refreshes the GUI at the client side.

5.4.8 Usage

You need a web and sql server to use this application. In this tutorial both will be taken care of by XAMPP: http://www.apachefriends.org/en/xampp.html
This tutorial adopts some parts of *101implementation* mySql. The company.sql and sampleCompany.sql are modified for this project. They are located in the ”sqlScripts” folder.

- Download and install XAMPP
- Open the ”XAMPP Control Panel” and start ”Apache” and ”MySQL”
- Use the guideline of *101implementation* mySql up to ”Populate tables...” with the modified sql scripts.

Once the database is running, follow the next steps:

- To start the application, you need to download all project files
- Put the files into the htdocs directory of your XAMPP (a new sub-directory in ”htdocs” is recommended)
- Run index.html

The project is provided as a netbeans project. If you want to change the code, you have to:

- Download (http://netbeans.org/) and install NetBeans
- ”Open project” and select the html5ajax folder

5.5 *101implementation* html5tree

5.5.1 Intent

Web programming based on the HTML5 ecosystem using Ajax style and a tree view

5.5.2 Languages

- HTML5
- JavaScript
- PHP
- SQL
- JSON

5.5.3 Technologies

- MySQL
- XAMPP
- Apache HTTP Server
- XMLHttpRequest
- jQuery
- Web browser

5.5.4 Features

- Company
- Total
- Cut
- Precedence
- Persistence
- Navigation
- Client-server
- Attribute editing
- Structural editing
- Web UI

5.5.5 Motivation

This web application extends the *html5ajax* implementation with a tree view, thereby improving the navigation. This is necessary to guarantee a clear overview while deleting, moving and creating departments and employees due to structural editing. JavaScript is very suitable for trees, because it allows client side DOM manipulation. This is necessary for creating fast, expandable tree structures. We have used the jQuery library to gain proper DOM manipulation.

5.5.6 Illustration

The first part of this illustration shows, how the tree view is created. The second part illustrates the deletion of a department as it is part of the structural editing feature.

Tree creation

As shown in *101implementation* html5ajax, the data is stored within a MySQL database. After the client creates the initial request, a server side PHP script called treeServer.php prepares the data for a JSON response. The initial XMLHttpRequest is rather simple:

Listing 5.17: treeModel.js

```
...

treeModel.load = function(strategy, id) {
    treeModel.initCompany(id);
    treeModel.company.action = "load";

    requestUnit.sendRequest(strategy, treeModel.url,
        treeModel.company);
}

...
```

The `requestUnit` (file: XMLHttpRequest.js) is developed to create generic requests. The given strategy updates the tree view with the retrieved data.

Server The server side PHP script treeServer.php creates the objects necessary for the tree. Therefore, it loads the entities from the database with the ids and names, bacause there is no need for total oder address values within the tree display. The following function illustrates the initialization of the company:

Listing 5.18: treeServer.php

```
...

function perform($jsonObject) {
    $action = $jsonObject->action;

    switch ($action) {
        case "load":
            return loadCompany($jsonObject);
    }
}

function loadCompany($jsonObject) {
    // The $jsonObject contains the request of the
       client.
    $id = $jsonObject->id;

    // This is the SQL statement to get the company with
        a given id.
    $request = "SELECT * FROM company WHERE id = $id";
    $result = mysql_query($request);
    $row = mysql_fetch_object($result);

    // These few commands create a new company and set
       the id and the name.
    $company = new Company();
    $company->setId($row->id);
    $company->setName($row->name);

    // The subdepartments are added to the company.
    $company->setDepartments(loadDepartmentsForCompany(
       $id));

    return $company;
}

...
```

The `perform($jsonObject)` method interprets the request of the client. After the server retrieves the company id, it is able to select the company out of the MySQL DBMS. After that, a new company object is filled with id and name. The subdepartments are added to the company within a nested tree structure. The `$company` is transformed to JSON before the response is returned to the client:

Listing 5.19: connection.php

```
...

echo json_encode(perform($jsonObject));

...
```

Client In the introduction of the tree creation section we have introduced the parameter `strategy`, which is used for the views callback initialization after the response is returned asynchronously. This strategy is initialized in the controller.js and simply invokes the method `refresh` of the treeView.js:

Listing 5.20: treeView.js

```
...

// This method refreshs the tree view.
treeView.refresh = function() {
    // The content variable contains the generated html string for a nested unordered list.
    content = "<ul>";

    // If there are subdepartments, create a list item with a ’plus’ symbol
    if (treeModel.response.departments.length > 0) {
        // The ’plus’ symbol is the button for expanding the tree.
        content += "<li> <input id=\"0\" "
            + "type=\"image\" src=\"symbols/plus.gif\" "
            + "onclick=\"treeNavigation.toggleList(this)\">";
        // The name of the company is the button for company load.
        content += "<input type=\"button\" class=\"companyButton\" value=\""
            + treeModel.response.name
            + "\" onclick=\"controller.loadCompany("
            + treeModel.response.id
            + ")\">";
        // The subdepartments are added as sublists of the company list item.
        content += treeView.showDepartments(treeModel.response.departments);
        content += "</li>";
    // If there are no subdepartments, create a list item with a ’dot’ symbol
    } else {
        content += "<li> <img src=\"symbols/leaf.gif\"><b>"
            + treeModel.response.name
            + "</b></li>";
    }

    content += "</ul>";

    // This replaces the content of the tree division with the generated html code in ’content’.
    document.querySelector(’#tree’).innerHTML = content;
}

...
```

Every tree item is a list item of a nested unordered list and consists of two visible parts, a symbol and a name. The symbol refers to the position of the item within the tree. There are *dots* for leafs and *plus* and *minus* symbols for unexpanded and expanded tree items. The name is a link, which refers to a load function. This function returns a complete data set for the requested entity.

Delete department

The following part illustrates an example for the deletion of a department according to the Structural editing. Please visit *101implementation* html5ajax to get an overview over loading company entities with XMLHttpRequest and JSON. Keep in mind, that there is no need to load subdepartments or employees for a specific department, since the tree provides the structure to pick these subelements.
After a department is selected, the user can press the *delete* button to invoke the corresponding client side delete method implemented in departmentModel.js:

Listing 5.21: departmentModel.js

```
departmentModel.deleteEntity = function(strategy) {
    departmentModel.initDepartment(departmentModel.
        response.id);
    departmentModel.department.action = "delete";
    requestUnit.sendRequest(strategy, departmentModel.
        url, departmentModel.department);
}
```

This method will send a request in the JSON format, which contains all necessary informations to delete a department with a given id. Most of the JSON messages in the web application are simplified versions of the messages in *101implementation* html5ajax.

```
{
    "id":1,
    "table":"department",
    "action":"delete"
}
```

This request is received by the server side PHP script, which deletes the department with the id 1. The cascading delete anchored in the database provides a recursive deletion for all containing subdepartments. Hence, the following shows the simple delete request:

Listing 5.22: departmentServer.php

```
$request = "DELETE FROM department WHERE id = " . $id;
mysql_query($request);
```

5.5.7 Architecture

The basic architecture is similar to the architecture of *101implementation* html5ajax. It is based on the MVC architectural pattern in combination with a client-server architecture. The key difference is the encapsulation of all views within one HTML file named index.html, controlled by different JavaScript files, which are located in view and the controller.js.

5.5.8 Usage

You need a web and MySQL server to run this application. In this tutorial both will be taken care of by XAMPP: http://www.apachefriends.org/en/xampp.html
This tutorial adopts some parts of *101implementation* mySql. The company.sql and sampleCompany.sql are modified for this project. They are located in the "sqlScripts" folder.

- Download and install XAMPP
- Open the "XAMPP Control Panel" and start "Apache" and "MySQL"
- Use the guideline of *101implementation* mySql up to "Populate tables..." with the modified sql scripts.

Once the database is running, follow the next steps:

- To start the application, you need to download all project files except the README
- Put the files into the htdocs directory of your XAMPP (a new subdirectory in "htdocs" is recommended)
- Run index.html

The project is provided as a netbeans project. If you want to change the code, you have to:

- Download (http://netbeans.org/) and install NetBeans
- "Open project" and select the html5tree folder

5.6 *101implementation* jsf

5.6.1 Intent

Web programming with JSF

5.6.2 Languages

- Java
- XHTML
- CSS
- XML
- JavaScript (generated)

5.6.3 Technologies

- JSF
- Hibernate
- Java EE
- NetBeans
- GlassFish
- Web browser

5.6.4 Features

- Company
- Total
- Cut
- Client-server
- Persistence
- Navigation
- Attribute editing
- Web UI

5.6.5 Motivation

This implementation covers the popular approach for web programming with JSF. JSF has a great support for the development of user interfaces in the MVC context. This implementation is considered as a typical Java based implementation with an amount of related and commonly used technologies like the Hibernate persistence API and the GlassFish application server.

5.6.6 Illustration

The main architecture is based on the MVC pattern. JSF itself is focussed on the view and the controller. The user interface (view) is provided by facelets, which is based on the XHTML dialect. A facelet contains the necessary GUI components for the specific view and connects them to the corresponding methods of the backend Java Beans (model). The data provided by the Java Beans is stored within a MySQL database, accessible through Hibernate with the help of DAOs (data access objects). The following sections provide a specific description of the involved parts.

GUI development

Facelets provide HTML page generation. Since the XML tags for the JSF components can not be displayed by the browser, they have to be changed to corresponding HTML tags via component aliasing. The following example shows the JSF components needed for total and cut of a company:

Listing 5.23: company.xhtml

```
<h:outputLabel for="total" value="Total:"/>
<h:outputText id="total" value="#{companyBean.total}"/>
<h:commandButton value="cut" actionListener="#{
    companyBean.cut()}"/>
```

The three elements model the GUI components for the label ”Total”, the textfield for the Total and the button for the Cut. For example, the `<h:commandButton .../>` is transformed into the HTML tag `<input .../>`. The communication between client and server is provided by XMLHttpRequest. The needed JavaScript files are automatically generated by the framework.
There is no need to implement the controller since it is provided by the Servlet API used within the JSF framework.

Managed Beans

The previously introduced command button ”cut” invokes the corresponding method `cut` in the Java Bean `CompanyBean`. In context of JSF, such beans are called Managed Beans:

Listing 5.24: CompanyBean.java

```
@ManagedBean(name = "companyBean")
@RequestScoped
public class CompanyBean {

```

```java
    ...

    // This is the set of employees for the whole company (loaded previously by the method "loadCompany(int id))".
    private Set<Employee> employees;

    ...

    // The method returns the current value for total of the loaded company.
    public double getTotal() {
        return total;
    }

    // The method cuts all employees of the loaded company.
    public void cut() {
        // Here we retrieve the session and begin the transaction with Hibernate.
        HibernateUtil.getSessionFactory().getCurrentSession().beginTransaction();
        DAOFactory daoFactory = DAOFactory.instance(DAOFactory.HIBERNATE);
        // The employeeDAO manages the database interaction.
        EmployeeDAO employeeDAO = daoFactory.getEmployeeDAO();

        // This loop iterates over the previously loaded employees and persists the new salary values.
        for (Employee employee : employees) {
            employee.setSalary(employee.getSalary() / 2);
            employeeDAO.makePersistent(employee);
        }

        total = total / 2;

        // Finally, we commit and close the transaction.
        HibernateUtil.getSessionFactory().getCurrentSession().getTransaction().commit();
    }

    ...

}
```

The class `CompanyBean` encapsulates the server-side business methods for the application. Responses for *GET* requests are provided by simple Java *getters* (e. g. `getTotal()`). *POST* requests are handled by corresponding methods (e. g. `cut()`) or by Java *setters*. The two annotations for the class provide the following features:

- `@ManagedBean` annotates, that the bean is a managed bean in the context of JSF. The attribute `name` provides the *connection point* useable within the facelets.
- `@RequestScope` annotates, that every new request affecting the `CompanyBean` will create a new instance.

The application server GlassFish provides the necessary container for the beans and manages the application.

Persistence

The principle of the DAO pattern is the exchangeability of the persistence layer. This is provided by a `DAOFactory`, which instantiates the specific DAOs for the used persistence technology. In our case, the technology is Hibernate. According to the interface GenericDAO.java, every implemented DAO provides the methods to load an entity either by id or by example, to load all entities corresponding to a specific class, or to persist a given entity.

Listing 5.25: GenericDAO.java

```java
T findById(ID id, boolean lock);
List<T> findAll();
List<T> findByExample(T exampleInstance);
T makePersistent(T entity);
void makeTransient(T entity);
```

The `T` stands for either the class `Company`, or the class `Department`, or the class `Employee`. The concrete methods for Hibernate are implemented in GenericDAOHibernate.java. This structure enables the Java Beans to perform data-affecting actions mostly independent from the persistence implementation:

Listing 5.26: CompanyBean.java

```java
@ManagedBean(name = "companyBean")
@RequestScoped
public class CompanyBean {

    ...

    public void cut() {
        // There has to be a transaction before creating
            hibernate requests.
        HibernateUtil.getSessionFactory().
            getCurrentSession().beginTransaction();

        // Retrieve the Hibernate DAOFactory for
            creating the employee DAO.
        DAOFactory daoFactory = DAOFactory.instance(
           DAOFactory.HIBERNATE);
        EmployeeDAO employeeDAO = daoFactory.
           getEmployeeDAO();

        // Cut all employees and save them.
        for (Employee employee : employees) {
            employee.setSalary(employee.getSalary() / 2)
               ;
            employeeDAO.makePersistent(employee);
        }

        // Calculate new total value.
        total = total / 2;

        // The transaction commits the data and ends.
        HibernateUtil.getSessionFactory().
           getCurrentSession().getTransaction().commit()
           ;
    }

    ...

}
```

There is an issue to be solved in the future: The transaction should be invoked by annotations or any automated transaction management instead of using corresponding methods with the help of `HibernateUtil`. If there is no Hibernate persistence, the calls will lead to exceptions.

5.6.7 Architecture

The facelets (company.xhtml, department.xhtml, employee.xhtml) are located in the folder *jsf/web/*. There is a template.xhtml as well, which arranges the main UI components of each view. The folder *jsf/web/resources/css/* contains the corresponding CSS files.
The navigation between the different facelets is managed with the help of the file *faces-config.xml*, while the starting page and the class for the controller servlet is defined in the web.xml.
The *jsf/src/java/* folder contains all relevant code for the realization of the main features (except navigation):

- The *hibernate.cfg.xml* defines the necessary data for the database connection and the classes for hibernate mapping.
- The folder *src/java/company/beans/jsf/* contains the three beans (CompanyBean.java, DepartmentBean.java, EmployeeBean.java) necessary for handling the companies, departments and employees.
- The classes folder contains the relevant classes for the instantiation of the company system.

- The dao folder contains all necessary classes and factories for the exchangablility of the data model corresponding to the DAO design pattern.

5.6.8 Usage

You need an sql-server to use this application. In this tutorial both will be handled by XAMPP (http://www.apachefriends.org/en/xampp.html).
You can use the *company.sql* and *sampleCompany.sql* of *101implementation* html5tree for the jsf project.

- Download and install XAMPP
- Open the "XAMPP Control Panel" and start "Apache" and "MySQL"
- Use the guideline of *101implementation* mySql up to "Populate tables..."

After the database is running, follow the next steps:

- To start the application, you have to download the sources from github
- Open the project with NetBeans (http://netbeans.org/)
- Select the project, right click and *run*
- The glassfish-server (and a browser window) with the application will start automatically

The project is implemented with NetBeans 7.0.1. You will need the full version with an installed GlassFish Application server.

5.7 *101implementation* zend

5.7.1 Intent

Web programming in PHP with the Zend framework

5.7.2 Languages

- PHP
- HTML
- CSS

5.7.3 Technologies

- Zend framework
- Web browser

5.7.4 Features

- Company
- Total
- Cut
- Client-server
- Navigation
- Persistence
- Attribute editing
- Web UI

5.7.5 Motivation

PHP is one of the most commonly used languages for web programming. To get a more structured and valuable implementation, it is reasonable to choose a PHP based web-application framework. The Zend framework provides a good infrastructure and a large amount of pre-assembled components and supports form-based web development. The use of PHP suggests an SQL DBMS like MySQL, to allow persistence. The connection to the DBMS is provided by the database adapter included in the Zend framework.

5.7.6 Illustration

This section is divided into three parts. The first part describes the mapping of the entities, the second part describes the implementation of Total and Cut and the third part describes the development of the GUI.

Object/Relational mapping

Persistence is provided by a MySQL database (see *101implementation* mySql) and the *zend adapter* for this specific database. The adapter is generated automatically according to the corresponding entries in the file application.ini:

```
resources.db.adapter = "PDO_MYSQL"
resources.db.params.host = "localhost"
resources.db.params.username = "root"
resources.db.params.password = ""
resources.db.params.dbname = "test"
```

The entities of the company are mapped by specific classes, which are extended by the class *Zend_Db_Table_Abstract*. They create simple associative arrays out of the declared tables. If we assume, that associative arrays are objects, we can call this process Object/Relational mapping. To map a table, it is only necessary to specify the name of the designated table within the corresponding class. The class `Application_Model_DbTable_Employee` in Employee.php illustrates the mapping of the employee table:

Listing 5.27: Employee.php

```
class Application_Model_DbTable_Employee extends
    Zend_Db_Table_Abstract
{

    protected $_name = 'employee';

    ...

}
```

Feature implementation

The features total and cut are both implemented in the model Employee.php in the sense of MVC. The reason is, that these methods affect only employees, depend on the corresponding company or department. The following example shows the calculation of the total value for a company with a given *$id*.

Listing 5.28: Employee.php

```
class Application_Model_DbTable_Employee extends
    Zend_Db_Table_Abstract
{

    ...

    public function getTotalForCompany($id) {
        // cast the id to integer
        $id = (int)$id;
        // get all employees from the database
        // - with the given company cid
        // - as an array composed of associative arrays
           for each employee
        $rows = $this->fetchAll('cid = ' . $id);

        $total = 0;

        // walk through the array and add every salary
        foreach ($rows as $row) {
            $total += $row->salary;
        }
        return $total;
    }

}
```

GUI

It is possible to create the GUI using a native HTML-PHP web page. The other way is to create simple forms by using the extension Zend_Form. Zend framework offers a great support for form based websites. That means, that it is possible to generate forms with almost no HTML code, because the web-application framework automatically generates the website out of the pre-defined PHP objects. The objects are initialized and added to the specific form in a developer-defined order. The following example shows the company form. The different GUI components are explained in the comments:

Listing 5.29: Company.php

```
class Application_Form_Company extends Zend_Form
{
    var $departmentList;

    public function init()
    {

        // The field helps to identify the form within
           the application.
        $this->setName('company');

        // The hidden field keeps the company id for
           further use.
        $id = new Zend_Form_Element_Hidden('id');
        $id->addFilter('Int');

        // The name field contains the name of the
           company.
        $name = new Zend_Form_Element_Text('name');
        $name ->setLabel('Name')
                ->setRequired(true)
                ->addFilter('StripTags')
                ->addFilter('StringTrim')
                ->addValidator('NotEmpty');

        // The button invokes the function to save a new name.
        $submit = new Zend_Form_Element_Submit('save');
        $submit ->setAttrib('id', 'submitbutton')
                ->setOptions(array('class' => 'button'));

        // The following list shows all subdepartments of the company.
        $this->departmentList = new Zend_Form_Element_Select('departments');
        $this->departmentList ->setLabel('Departments');
        $this->departmentList->setRegisterInArrayValidator(false);

        // The button is pressed to select a department out of the list.
        $select = new Zend_Form_Element_Submit('select');
        $select ->setAttrib('id', 'submitbutton');

        // The textfield shows the total value.
        $total = new Zend_Form_Element_Text('total', array("readonly" => "readonly"));
        $total  ->setLabel('Total');

        // The button is used to cut the company.
        $cut = new Zend_Form_Element_Submit('cut');
        $cut->setAttrib('id', 'submitbutton');

        // There is a block created in the GUI for each array added in the following way:
        $this->addElements(array($id, $name, $submit));
        $this->addElements(array($this->departmentList, $select));
        $this->addElements(array($total, $cut));
    }
    ...
}
```

If the controller wants to fill in some data into the total (`$total`) field, it must call the method `populate` of the previously initialized form. If the array parameter of the method contains a field `total`, the textfield with the identifier ”total” is filled automatically with the value contained in the array:

Listing 5.30: Company.php

```
...

$c = $company->getCompany($id);
$c[total] = $employee->getTotalForCompany($id);

$form->populate($c);

...
```

5.7.7 Architecture

- The entry point for the application is the generated index.php in the public folder. This folder contains the style sheets, too.
- All relevant code for the functionality is located in the application folder.
- The architecture is strictly based on the MVC-pattern. There are different folders for the models (models), the views (views) and the controllers (controllers). This is required, because the framework automatically identifies the classes and its concerns with the help of these namespaces.
- The forms are located in the forms folder.

5.7.8 Usage

This project needs a running MySQL-database. We recommend XAMPP. You can use the *company.sql* and *sampleCompany.sql* of *101implementation* mySql for this project.

- Download and install XAMPP.
- Open the ”XAMPP Control Panel” and start ”Apache” and ”MySQL”.
- Use the guideline of *101implementation* mySql up to ”Populate tables...”.

After the database has started:

- Download the sources.
- Copy the complete zend folder to the htdocs-directory of your XAMPP-installation.
- Start your Web browser and go to http://localhost/zend/public/.

If you want to continue with the development of this 101implementation, please download NetBeans and import the project into the IDE.

5.8 *101implementation* pyjamas

5.8.1 Intent

Web programming in Python with Pyjamas

5.8.2 Languages

- Python
- JavaScript (generated)
- HTML (generated)

5.8.3 Technologies

- Pyjamas
- Web browser

5.8.4 Features

- Company
- Total
- Cut
- Code generation
- Navigation
- Attribute editing
- Web UI

5.8.5 Motivation

Pyjamas offers the possibility to generate pure JavaScript-Code out of Python source code. One advantage of pyjamas is, that it is very simple to understand. Apart from that, there is no need for complicated HTML or JavaScript programming. Pyjamas is combined with CSS. It also can be considered as a ”spin off” of GWT.

5.8.6 Illustration

Although pyjamas demands no specific architecture, the main parts of the application are located in a simple Python file 101Companies.py. Nevertheless, we have devided the code into two concerns encapsulated into different classes. One class contains the GUI generation, another class contains the company data and manages the major functionalities of the app. There are further classes for each company, department and employee. The complete Python code illustrated in this section is translated directly to JavaScript.

GUI implementation

The GUI is implemented with the help of pre defined Pyjamas classes. There is a grid with all necessary components like labels, textfields, listboxes and buttons. The following method of the class `101CompaniesAppGUI` shows the initialization of the employee part of the GUI:

Listing 5.31: 101Companies.py

```python
# initializes the GUI for the employee view
def initEmployeeGUI(self):
    self.grid.clear()
    self.grid.resize(4, 3)

    # row 1
    self.grid.setWidget(0, 0, Label("Name:"))    # column
         1 = name
    self.grid.setWidget(1, 0, Label("Address:"))     #
        column 2 = address
    self.grid.setWidget(2, 0, Label("Salary:")) # column
         3 = salary

    # row 2
    self.grid.setWidget(0, 1, self.name)
    self.grid.setWidget(1, 1, self.address)
    self.grid.setWidget(2, 1, self.total)

    # row 3
    self.grid.setWidget(0, 2, self.save)
    self.grid.setWidget(2, 2, self.cut)
    self.grid.setWidget(3, 2, self.back)

    # initialize content for current employee
    self.name.setText(self.current.name)
    self.address.setText(self.current.address)
    self.total.setText(self.current.salary)
```

The three textfields `self.name`, `self.address` and `self.total` and the buttons `self.save`, `self.cut` and `self.back` are initialized in the constructor of the class `101CompaniesAppGUI`. The `self` parameter represents the current instance of the class.

Feature implementation

There is only one handler for each button located in the class `101CompaniesAppGUI`. The handler is implemented by the function `onClick`. It contains a control structure, which determines the clicked button:

Listing 5.32: 101Companies.py

```python
def onClick(self, sender):
    self.errors.clear()
    if sender == self.cut:
        self.current.cut()
        self.total.setText(self.current.total())
    else if sender == ...
```

In case of cut, this method cuts the current entity and refreshes the the total textfield. Every class, be it the company, department or employee, implements such a cut method. In case of employee, the specific implementation looks like:

Listing 5.33: 101Companies.py

```
class Employee:
    ...

    def cut(self):
    self.salary = self.salary / 2

    ...
```

5.8.7 Architecture

There are three files, which are not generated:

- The file 101Companies.py contains the main functionalities of the application.
- 101Companies.html is the index page for the application.
- style.css defines the CSS attributes for the elements.

The file 101Companies.py is divided into three parts:

- The company structure and most of the attribute editing is provided by the classes *company*, *department* and *employee*.
- The GUI is provided by the class *101CompaniesAppGUI*.
- The company initialization and the low-level management of the departments and employees is provided by the class *101CompaniesApp*.

5.8.8 Usage

- install pyjamas (getting started)
- download the sources
- open a terminal and move to your local pyjamas implementation-folder
- type *pyjsbuild 101companies.py*
- open 101Companies.html with your Web browser

If you only want to watch the HTML/javaScript-result of the compile process, please:

- download the sources
- open 101Companies.html with your Web browser

5.9 *101implementation* gwtTree

5.9.1 Intent

Tree-based web programming with GWT

5.9.2 Languages

- Java
- JavaScript
- HTML
- XML
- CSS

5.9.3 Technologies

- GWT (Version 2.5)
- Web browser

5.9.4 Features

- Company
- Cut
- Total
- Code generation
- Precedence
- Client-server
- Navigation
- Attribute editing
- Structural editing
- Web UI

5.9.5 Motivation

GWT is a framework for creating JavaScript web applications with Ajax support in Java. Thereby, the complete JavaScript code is generated out of the Java code. The result is a complete client-server-based web application with a tree based GUI. Additionaly, the implementation extends the basic *101implementation* gwt adding the Attribute editing and Structural editing.

5.9.6 Illustration

The main difference to the *101implementation* gwt implementation is the improvement of the navigation. Hence, the focus of this section is the development of the GUI and the implementation of the new features.

GUI implementation

The supervising component of the GUI is the parent class implemented in GwtTree.java. This component manages the visibility of the three panels CompanyPanel.java, DepartmentPanel.java and EmployeePanel.java and, additionaly, contains the panel TreePanel.java. The method `onModuleLoad()` initializes all these panels on application start.

Tree The class `Tree`, included in GWT, provides great support for a tree based view. Every tree item contains a user object with the identifier of the specific entity. The tree generation starts with the company as root:

Listing 5.34: TreePanel.java

```java
...

for (CompanyItem item : info.getCompanies()) {
    TreeItem root = new TreeItem(item.getName());
    root.setUserObject(item);
    appendDepsAndEmps(root, item.getDepartments());
    addItem(root);
}

...
```

First, the tree item for the company is generated with the name of the company. The `setUserObject` method adds the `CompanyItem` with the identifier of the company. The method `appendDepsAndEmps` appends all departments and employees to the company root. At last, this root is added to the tree.
The selection handler of the tree invokes the displaying of the additional informations for each entity included in the company:

Listing 5.35: TreePanel.java

```java
...

this.addSelectionHandler(new SelectionHandler<TreeItem
    >() {

    @Override
    public void onSelection(SelectionEvent<TreeItem>
        event) {
        Object obj = event.getSelectedItem().
            getUserObject();
        if (obj instanceof CompanyItem) {
            TreePanel.this.main.showCompany(((
                CompanyItem)obj).getId());
        } else if (obj instanceof DepartmentItem) {
            TreePanel.this.main.showDepartment(((
                DepartmentItem)obj).getId());
        } else if (obj instanceof EmployeeItem) {
            TreePanel.this.main.showEmployee(((
                EmployeeItem)obj).getId());
        }
    }
});

...
```

The `TreePanel.this.main` member provides the main class GwtTree.java. There is a different method `TreePanel.this.main.show...` for each different entity class. This method includes the specific company, department or employee panel into the GUI and initializes it with the corresponding data.

Panel The following section illustrates the main functionalities of the department panel. This panel containes two textfields for the name and the total value and two listboxes for the parent department and the manager. These elements are encapsulated within a `grid`:

Listing 5.36: DepartmentPanel.java

```
...

Grid grid = new Grid(4, 3);

...

Label lname = new Label("Name:");
lname.setWidth("60px");

// add labels
grid.setWidget(0, 0, lname);
grid.setWidget(1, 0, new Label("Total:"));
grid.setWidget(2, 0, new Label("Manager:"));
grid.setWidget(3, 0, new Label("Parent:"));

// add textboxes
grid.setWidget(0, 1, name);
grid.setWidget(1, 1, total);
grid.setWidget(2, 1, manager);
grid.setWidget(3, 1, parent);

...

add(grid);

```

```
HorizontalPanel buttons = new HorizontalPanel();

...

buttons.add(save);
buttons.add(cut);
buttons.add(delete);

add(buttons);

...
```

The Grid constructor has two parameters for the table rows and columns. First, the labels are added and second, the textfields and listboxes are added. The buttons cut, save and delete are added within a seperate `HorizontalPanel buttons`.

Feature implementation

The following part shows the implementation of the Cut for departments. We assume, that a user has clicked on the ”Cut” button to create a corresponding request. All client side requests are handled via asynchronous communication. First we need a connection to the server-side service:

Listing 5.37: DepartmentPanel.java

```
private final DepartmentServiceAsync departmentService =
     GWT.create(DepartmentService.class);
```

This simple initialisation provides an easy possibility to connect the client to the server. The next step is to create a Cut request. We have implemented this feature with the interfaces interface. By calling this method, the request is created and the corresponding server side method is invoked. The two parameters of the method are in first the department identifier (”*department*”) and in second an object for the asynchronous callback.

Listing 5.38: DepartmentPanel.java

```
departmentService.cut(department, new AsyncCallback<
    Double>() {

    // If the server responds an error, this method is
        invoked.
    @Override
    public void onFailure(Throwable caught) {
        Window.alert(caught.getMessage());
    }

    // If the request is successfully executed on server
         side, this method is invoked with a new total
        value as parameter.
    @Override
    public void onSuccess(Double result) {
        DepartmentPanel.this.total.setText(Double.
            toString(result));
    }
});
```

If the request fails, the *onFailure* method simply returns an adequate error message. If the request succeeds, the server returns the new total value for the department. The server side request handler is implemented as a method:

Listing 5.39: DepartmentServiceImpl.java

```
@Override
    public double cut(Integer id) {
        // This statement loads the department with the
            given id.
        Department department = CompanyApp.getInstance()
            .getDepartments().get(id);

        // The department gets cutted.
        department.cut();

        // The new total value for the department is
            returned.
        return department.total();
    }
```

This method first calls amethod cut for the specific department and then returns the new total value to the client. The method for one department is implemented the following way:

Listing 5.40: Department.java

```
public class Department implements Parent {

    // Members
    private int id;
    private String name;
    private List<Department> departments;
    private List<Employee> employees;
    private Parent parent;

    ...

    // This method cuts the salary of all contained
        employees and all contained subdepartments.
    public void cut() {
        for (Employee employee : employees) {
            // The employees salary is devided by 2.
            employee.cut();
        }
        for (Department department : departments) {
            // The subdepartments are cutted recursively
                .
            department.cut();
        }
    }

    ...

}
```

5.9.7 Architecture

- The application architecture is devided into client and server packages: client and server. While the server package contains all relevant data for the representation of the company, the client package contains all relevant informations to generate a user interface.
- The interfaces for asynchronous communication between client and server are defined in interfaces. These interfaces are also implemented by the service classes of the server.
- The communication between server and client is provided with serialized classes, which can be interpreted by the client implementation. Therefore, we have two classes for each company, department and employee to submit the data from the server to the client. The first class, provided in guiinfo, delivers the data needed for the whole set of information for an entity like name, address, salary and parent for a given employee. The second class, provided in tree, delivers the data needed for generating a tree-based GUI.
- The GUI is divided into panels for each concern. They are located in client. The panels are initialized in the class GwtTree.java. The panels CompanyPanel.java, DepartmentPanel.java and EmployeePanel.java are exchangeable with each other, while the panels TreePanel.java and the ButtonPanel.java are always visible.

5.9.8 Usage

The implementation is created with Eclipse (3.7/Indigo) and the GWT-plugin (Version 2.5). If you want to compile the code, you need these versions to run it safely. You can get them at:

- Eclipse
- GWT

After installation and start, simply open this web application as project in Eclipse:

- Please clean up (Project → Clean...) the project before running.
- Right-click on the project, left-click on ”Run As” and left-click on ”Web-application”.
- Visit http://127.0.0.1:8888/GwtTree.html?gwt.codesvr=127.0.0.1:9997

If you want to recompile it, please end the old server-process by terminating the process in the Eclipse console first.

5.10 *101implementation* seam

5.10.1 Intent

Web application development with Java and the Seam framework

5.10.2 Languages

- Java
- HQL
- XHTML
- SQL
- XML
- JavaScript (generated)

5.10.3 Technologies

- Seam
- JBoss Application Server
- Eclipse

5.10.4 Features

- Company
- Total
- Cut
- Client-server
- Navigation
- Persistence
- Access control
- Web UI

5.10.5 Motivation

Seam provides a great support for the development of web applications in Java with easy access control and an integrated persistence layer. It combines the popular JSF (please visit *101implementation* jsf for more information) approach for UI programming with Hibernate and JPA. Both persistence technologies are directly integrated into the web-application framework.

5.10.6 Illustration

Seam is based on the three-tier architecture. Since the presentation is covered by JSF, the focus of the Seam framework is the business and data access layer. This section illustrates the major components of the Seam framework and introduces its rights management.

Presentation

The presentation layer is based on the MVC architecture, because it is the main architecture of the JSF framework. From this it follows, that the view consists of Facelets. We will start our presentation illustration with the first page of the application: the company view. It contains four parts, at which there are two textfields for the name and the total of the company and two buttons for the department list and the cut of the company. In theory, the view is designed to show more than one company, but this is not demanded by the feature model.

Listing 5.41: listAllCompanies.xhtml

```
...

<!-- 'c' is mapped to the field 'allCompanies' contained
     in the CompanyAction.java. -->
<rich:dataTable value="#{allCompanies}" var="c" width="
    300px">
    <rich:column>
        <f:facet name="header">Name</f:facet>
        <!-- The field displays the name of the company.
            -->
        <h:outputText value="#{c.name}" />
    </rich:column>
    <rich:column>
        <f:facet name="header">Total salaries</f:facet>
        <!-- The field displays the total value of the
           company. -->
        <h:outputText value="#{c.total()}" />
    </rich:column>
    <rich:column>
        <!-- This button opens the view for all
           departments contained by the company. -->
        <s:button value="Show details" action="#{
           companyAction.showDetails()}" />
    </rich:column>
    <rich:column>
        <!-- This is the cut button for the company. -->
        <s:button value="Cut salaries" action="#{
           companyAction.cutSalaries()}" />
    </rich:column>
</rich:dataTable>

...
```

The `<rich:dataTable ...> ... </rich:dataTable>` tags create a new HTML table with a predefined `width="300px"`. The value `"#allCompanies"` maps the table rows to the `allCompanies` list contained in the class CompanyAction.java. Each entry has a corresponding table row. The attribute `var="c"` helps to access one object of the list and get its informations. As mentioned above, the table has four columns: name, total, detail button and cut button. Each column is created by the use of the tags `<rich:column> ... </rich:column>`. The methods of the class CompanyAction.java are directly accessed via the actions of the buttons, for example in `<s:button value="Cut salaries" action="#companyAction.cutSalaries()"/>`. CompanyAction.java is a Java Bean, which receives the requests of the facelets.
The following example illustrates the `cutSalaries()` method implemented by the class CompanyAction.java:

Listing 5.42: CompanyAction.java

```java
...

public String cutSalaries() {
    try {
        // The method cuts the salaries for the selected
           company.
        companyService.cutSalaries(selectedCompany);
        // If the call is successful, a corresponding
           message is displayed.
        facesMessages.add(FacesMessage.SEVERITY_INFO, "
           The cut salary operation was successfully
           applied.");
    }
    catch(Exception e) {
        // If an exception occures, an error message is
           displayed.
        facesMessages.add(FacesMessage.SEVERITY_ERROR, "
           Error when trying to cut salaries. " + e.
           getMessage());
        e.printStackTrace();
    }
    // The framework expects a view name for loading the
       next view. If this name is 'null', the current
       view will reloads.
    return null;
}

...
```

The action invokes the `cutSalaries(selectedCompany)` method of the CompanyService.java and additionaly manages the loading of further pages. If an exception occurs, it returns an error message to the facelet. The `null` return value simply means, that the current view has to be reloaded.

Business and Data access

The business layer manages the access control of the application. In addition, it is connected to the persistence layer, which is implemented with JPA. On cut, the following method of the service class CompanyService.java is invoked:

Listing 5.43: CompanyService.java

```java
...

// The annotation manages the access control for this
   method.
@Restrict("#{s:hasRole('admin')}")
public void cutSalaries(Company company) {
    company.cut();
    entityManager.merge(company);
}

...
```

A simple annotation `@Restrict` provides access controll for the different users. The annotated method is restricted to a user or a group of users, in this case to the "admin" user. The cut is performed within the company object directly. The `entityManager` manages the company entities and provides an easy merge mechanism for all contained entities. In this case all manipulated employees of this company are merged automatically without explicit call.

5.10.7 Architecture

- The Facelets defining the view are located in the folder *[view]*. The folder contains layout templates, images and style sheets as well, which are provided directly by the Seam web-application framework.
- The Java bean CompanyAction.java is located in web and provides the connection point to the business layer.
- The entity classes for the company are located in the model package.
- The Seam service class is located in the services package.

There are some necessary configuration files like the *build.xml* and the *build.properties*. Apart from that, everything else is generated or part of the libraries.

5.10.8 Usage

This application requires the Eclipse and JBoss Application Server (Version 4.2.x). After downloading and installing both, follow these steps:

- Import the seam project from your file system to Eclipse as a java project.
- Change the build.properties file located in the projects base folder. The *jboss.home*-property must refer to your JBoss Application Server location: *jboss.home = <your JBoss location>*.

- Run the seam project as ant build in eclipse (Right click on the *build.xml* file → Run As → Ant Script).
- Start JBoss using either *<your JBoss location>/bin/run.bat* for Windows OS or *<your JBoss location>/bin/run.sh* for Unix OS.
- Start a Web browser and go to http://localhost:8080/seam

It is possible to access the system with two different users: ”admin” (Password: ”admin”) and ”user” (Password: ”user”). If you are logged in as ”admin”, you are allowed to visit all departments and employees contained in the company and cut all salaries. If you are logged in as ”user”, you are only allowed to visit all elements of the company but not to cut salaries.

5.11 *101implementation* strutsAnnotation

5.11.1 Intent

Web programming in Java with Apache Struts configuring with annotations

5.11.2 Languages

- Java
- XML
- HTML
- CSS

5.11.3 Technologies

- Apache Struts
- Java Servlet container (for example: Apache Tomcat)
- Web browser
- Apache Maven

5.11.4 Features

- Company
- Total
- Cut
- Client-server
- Navigation
- Web UI

5.11.5 Motivation

This Java web application illustrates the use of the popular Apache Struts technology in combination with JSP and a servlet based web server. It introduces the use of Apache Maven as a good advantage for Java based applications, as well.

5.11.6 Illustration

The Apache Struts architecture is based on MVC. The view is implemented with JSP, the controller is based on Java *action* classes in combination with servlets and the model is provided by a service class and a class for each company, department and employee. The initial company data is stored in a [this!!src/main/resources/sampleCompany.ser serialization file] (compare *101implementation* javaInheritance). We will illustrate the Apache Struts implementation with an example of showing and cutting a company.

View

The JSP [this!!src/main/webapp/WEB-INF/content/list-all-companies.jsp file] for the view of the company offers two textfields and two buttons. One of the textfields shows the name, the other one shows the total value. The buttons allow the user to request for further detailed information like a department list. It also allows her to cut all salaries of the company.

Listing 5.44: list-all-companies.jsp

```
...

<s:form action="company">

        ...

        <s:iterator value="allCompanies">
            <tr>
                <!-- The name and total fields refer
                    to the corresponding getters of "
                        Company.java". -->
                <td><s:property value="name"/></td>
                <td><s:property value="total"/></td>
                <td>
                    <!-- These lines create a link,
                        which invokes the cutSalaries-
                           method of "CompanyAction.java
                            -->
                    <s:url id="cutURL" action="company.
                        cutSalaries">
                        <s:param name="id" value="%{id}"
                           />
                    </s:url>
                    <s:a href="%{cutURL}">Cut</s:a>
                </td>
                <td>
                    <!-- These lines create a link
                        to the department list of the
                           company. -->
                    <s:url id="detailURL" action="
                        company.details">
```

```
                              <s:param name="id" value="%{id}"
                                  />
                         </s:url>
                         <s:a href="%{detailURL}">Detail</s:a
                             >
                    </td>
               </tr>
          </s:iterator>

          ...
```

The iterator `<s:iterator ... />` creates a table row for each company listed in the ListAllCompaniesAction.java instance. The value `"allCompanies"` refers to the member `List<Company> allCompanies;` of the class. Each company within this list has a getter for its name and another for the total value.

Controller

The `action="company.cutSalaries"` of the cut link invokes the `cutSalaries()` method of CompanyAction.java, which is, in combination with a servlet, a controller of the application:

Listing 5.45: CompanyAction.java

```java
@Action(value = "company.cutSalaries",
               results = { @Result(name = "listAllCompanies
                   ", type="redirectAction", location="list-
                   all-companies")})
public String cutSalaries() {
     company =  CompanyService.instance().findCompany(
         Long.parseLong(RequestUtil.getRequestParameter("
         id")));
     company.cut();
     return "listAllCompanies";
}
```

The `@Action(value = "company.cutSalaries", ...)` maps this method to the action name *company.cutSalaries*. Whenever this name is called within an action of the JSP files, this method is invoked. `CompanyService.instance()` returns the instance of the model, which returns the necessary entity for the company. This object is used to perform the companies cut method. The `results` parameter redirects the application to the [this!!src/main/webapp/WEB-INF/content/list-all-companies.jsp list-all-companies.jsp], which simply means, that the page is reloaded. Every result entry refers to a returned *string* value of the `@Action(value = "company.cutSalaries", ...)`. In this case, there is only one possible return value.

Model

All data are instantiated within the Singleton CompanyService.java, which is considered as a major part of the model. The instance contains lists of the company, its departments and its employees. As we have seen in the controller description, the cut method of all of these entities is invokeable. In our case, the cut method for the company simply invokes the cut method of the departments:

Listing 5.46: Company.java

```
public void cut() {
        for (Department d : getDepts())
            d.cut();
    }
```

The lists within this CompanyService.java are initialized by loading a previously serialized company. We strongly recommend to have a look on *101implementation* javaInheritance, to get an overview over the serialization and deserialization process in Java.

5.11.7 Architecture

- The JSP files for the view are located in content.
- The index.jsp in the webapp folder represents the initial page, which redirects to the first *list-all-companies.jsp*. The webapp folder also contains the CSS files for this application.
- The resources folder contains two necessary files. The sampleCompany.ser provides the serialized company data. The struts.xml XML file defines, that the initial page of this application is the index.jsp.
- The main Java code of the application is located in softlang. We have four folders for the different concerns. actions contains the actions described in the illustration section for the controller. The basics folder contains all necessary classes for the deserialized company. The CompanyService.java is located in the services folder. The last folder util contains some helpful Java files for the deserialization.

5.11.8 Usage

Requirements:

- Apache Maven (Version 2.x) as Eclipse plugin (http://eclipse.org/m2e/download/) or standalone (http://maven.apache.org/download.html)
- Web server or application server based on the servlet technology (We recommend JBoss application server, but Apache Tomcat will also be sufficient).

Import (Eclipse only): Import the strutsAnnotation implementation into eclipse as Maven project:

- Click the ”File”-button in the menu bar and ”Import...”.
- Select the ”Maven” folder and ”Existing Maven Projects”.
- Browse to your local ”strutsAnnotation” folder and ”Finish”.

Build:

- Run ”mvn clean” and ”mvn install” in the root directory of the struts 2 implementation or (*Eclipse only*) right click on your imported project and first click ”Run As” → ”Maven clean” and second ”Run As” → ”Maven install”.

- Copy the target/struts2app.war file to the web-application folder of your Web server (JBoss application server: <JBossHome>/server/default/deploy).

Run:

- Start your Web server (JBoss application server: <JBossHome>/bin/run.bat (Windows) or <JBossHome>/bin/run.sh (Unix)).
- Start your Web browser and go to http://localhost:8080/struts2app.

5.12 *101implementation* silverlight

5.12.1 Intent

Web programming in C# with Silverlight

5.12.2 Languages

- CSharp (C#): Programming language used for all code (Version 4.0)

5.12.3 Technologies

- csc.exe: C# compiler (Version 4.0)
- .NET: Framework used to execute compiled code (Version 4.0)
- Silverlight: (Version 4.0) used for the client-side

5.12.4 Features

- Company
- Total
- Cut
- Client-server
- Navigation
- Web UI

5.12.5 Motivation

The implementation illustrates the development of a client-side user interface accessible via web browser. We used Silverlight in combination with the Navigation framework n to create an MVC based implementation. The web application is tied to a web service, which is implemented in *101implementation* wcf. An advantage of Silverlight is the out-of-the-box back button support.

5.12.6 Illustration

There have to be proxy DTOs enabling the client to receive the serialized data from the web service. These proxies are generated from a WSDL file. The namespace for the generated classes is `silverlight.CompanyServiceReference`. These proxies are CSharp classes located in [this!silverlight/Service References/CompanyServiceReference/Reference.cs]:

Listing 5.47: Reference.cs

```
namespace silverlight.CompanyServiceReference {
    using System.Runtime.Serialization;

    // The class "CompanyDto" is the proxy DTO for
       serialized company data.
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("
       System.Runtime.Serialization", "4.0.0.0")]
    [System.Runtime.Serialization.DataContractAttribute(
       Name="CompanyDto", Namespace="http://schemas.
       datacontract.org/2004/07/wcf.Dto")]
    public partial class CompanyDto : object, System.
       ComponentModel.INotifyPropertyChanged {

        ...

    }

    ...

    // The connection point to the company service
       provided by the wcf implementation.
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("
       System.ServiceModel", "4.0.0.0")]
    public partial class CompanyServiceClient : System.
       ServiceModel.ClientBase<silverlight.
       CompanyServiceReference.ICompanyService>,
       silverlight.CompanyServiceReference.
       ICompanyService {

        ...

    }

    ...

}
```

GUI

The proxy in combination with the `CompanyServiceClient` allows us to create an asynchronous communication asynchronous request.

Listing 5.48: Home.xaml.cs

```
// Create a service client for companies and ...
var client = new CompanyServiceClient();
```

```
client.GetCompanyCompleted += client_GetCompanyCompleted
    ;
// ... wait for the requested company.
client.GetCompanyAsync();
```

The illustrated code creates a service client and waits, until the company data are completely delivered to the client. The result is a DTO, which is stored in the `DataContext`:

Listing 5.49: Home.xaml.cs

```
void client_GetCompanyCompleted(object sender,
    GetCompanyCompletedEventArgs e)
{
    DataContext = e.Result;
}
```

We are now able to fill the GUI with the help of the `DataContext`. Each field of the DTO is automatically binded to the GUI defined by the XAML files by using binding properties. For example, the Home.xaml contains a field `txtCompanyName` binded to the field *Name* of the DTO.

Listing 5.50: Home.xaml

```
<TextBlock Height="23" HorizontalAlignment="Left" Margin
    ="59,46,0,0" Name="txtCompanyName" Text="{Binding
    Name}"
```

5.12.7 Architecture

The project is represented as two Visual Studio projects:

- *silverlight.Web* contains the generated bootstrap code for launching the silverlight application.
- *silverlight* contains the actual implementation. The WSDL file and the generated DTO proxies are located in the folder *silverlight/ServiceReferences/CompanyServiceReference*. The GUI is provided by XAML files located in the base folder of the project.

5.12.8 Usage

- Follow the Usage section for the *101implementation* wcf to create a WCF service.
- Build the project using Visual Studio.
- Open the silverlightTestPage.html from *silverlight.Web* folder.

5.13 *101implementation* wcf

5.13.1 Intent

WCF Web service implementation in .NET 4.0

5.13.2 Languages

- CSharp (C#)
- XML
- WSDL

5.13.3 Technologies

- csc.exe
- .NET
- WCF
- IIS
- SOAP
- ASP .NET

5.13.4 Features

- Company
- Total
- Cut
- Client-server

5.13.5 Motivation

The WCF technology supports the development of applications based on service-oriented architecture. Our implementation illustrates a stateful web service created with WCF. It also introduces the use of SOAP over HTTP. That necessitates the use of DTOs to wrap the domain model into leightweight serializable containers and pass it to the client.

5.13.6 Illustration

WCF uses contracts to generate WSDL definitions for the web service. The contract is provided by a CSharp interface. It contains all necessary methods for retreiving the company data and invoking the method cut on every entity within the company:

Listing 5.51: ICompanyService.cs

```
[ServiceContract]
public interface ICompanyService
{
    // The following methods define the interface for
       data retrieval.
    [OperationContract]
    CompanyDto GetCompany();

    [OperationContract]
    DepartmentDetailsDto GetDepartmentDetails(Guid id);

    [OperationContract]
    EmployeeDto GetEmployee(Guid id);

    // The following methods define the interface for
       performing cut on each entity.
    [OperationContract]
    decimal CutDept(DepartmentDetailsDto dept);

    [OperationContract]
    decimal CutEmpl(EmployeeDto emp);

    [OperationContract]
    decimal Cut(CompanyDto company);
}
```

The class `CompanyService` is the concrete contract implementation. The implementation uses the basic *101implementation* csharp for data generation and manipulation. Our contract class invokes the methods of the implementation and creates DTOs out of the retrieved entities. The example shows the implementation of the method `GetCompany`. The method returns all necessary data for the company view encapsulated into a DTO:

Listing 5.52: CompanyService.svc.cs

```
[AspNetCompatibilityRequirements(RequirementsMode =
    AspNetCompatibilityRequirementsMode.Required)]
[ServiceBehavior(InstanceContextMode =
    InstanceContextMode.Single)]
public class CompanyService : ICompanyService
{

    ...

    // The concrete method implementation creates the DTO for a company using the data provided in the baseline csharp implementation.
    public CompanyDto GetCompany()
    {
        var dto = new CompanyDto
        {

            // The DTO is filled with ID, Name and subdepartments.
            Id = Company.Id,
            Name = Company.Name,
            Departments = Company.Departments.Select(d => new DepartmentDto
            {

                // The data for each department are included iteratively.
                Details = new DepartmentDetailsDto
                {
                    Id = d.Id,
                    Name = d.Name,
                    Manager = new EmployeeDto
                    {
                        // Every Manager has its id, address, name and salary.
                        Address = d.Manager.Person.Address,
                        Id = d.Manager.Id,
                        Name = d.Manager.Person.Name,
                        Salary = d.Manager.Salary
                    }

                },

                // The data for each employee are included iteratively.
                Employees = d.Employees.Select(e => new EmployeeDto
                {
                    // Every Manager has its id, address, name and salary.
                    Id = e.Id,
                    Address = e.Person.Address,
                    Name = e.Person.Name,
                    Salary = e.Salary
                }).ToList(),

                SubDepartments = FillSubDepartments(d),
            }).ToList(),

            // The parameter "Total" is filled.
            Total = Company.Total
        };

        return dto;
    }

    ...

}
</syntaxhighlight >
The DTO classes <syntaxhighlight lang="csharp" enclose="
    none">CompanyDto
```

, `DepartmentDetailsDto`, and `EmployeeDto` provide the serializability for view informations. The following example illustrates the DTO of the company, which contains fields for identifier, name, subdepartments and Total.

Listing 5.53: CompanyDto.cs

```
[ServiceContract]
[DataContract]
public class CompanyDto
{
    // Identifier
    [DataMember]
    public Guid Id { get; set; }

    // Name
    [DataMember]
    public string Name { get; set; }

    // Subdepartments
    [DataMember]
    public List<DepartmentDto> Departments { get; set; }

    // Total value
    [DataMember]
    public decimal Total { get; set; }
}
```

5.13.7 Architecture

The base folder contains the contract definition ICompanyService.cs and the contract implementation *ComanyService.svc.cs*. The file CompanyService.svc links to the concrete implementation. DTO definitions are located in the *dto* folder. Both files, clientaccesspolicy.xml and crossdomain.xml, solve possible cross-domain issues. The *.config* files in the base folder are generated.

5.13.8 Usage

Requirements:

- You need Microsoft Windows (preferably Vista or 7) running in combination with .NET 4.0.
- Install IIS (Internet Information Services) 7 or 7.5 using the installation tutorials on IIS] or [http://msdn.microsoft.com/de-de/library/aa964620.aspx MSDN. Make sure, that you have installed the ”static content” feature as well.
- Download the wcf implementation from github.

Build:

- Use Visual Studio to build the project.

Create web site:

- Go to the ”Computer Management” dialog.
- Open the ”Services and Applications” root and select the ”Internet Information Services (IIS) Manager”.
- On the right, there is a panel ”Connections”. Create a new web site by right click on ”web sites”.
- Use *wcf* as ”sitename” and link to the physical folder of the wcf implementation. Use port 1212.

Make sure that http://localhost:1212/CompanyService.svc shows that you have created a service. After the web service is running, continue with your client side application, for example *101implementation* silverlight.
We also used:

- version 4.0 of WCF
- version 4.0 of CSharp
- and version 4.0 of the csc.exe compiler

to create the implementation.
Attention: When creating a service anywhere other than on the web server, that hosts your Silverlight application, cross-domain issues can arise. Cross-domain calls between Silverlight applications and services present a security vulnerability and must be specifically enabled by an appropriate cross-domain policy. For procedures that describe how to implement such a policy, see ”Making a Service Available Across Domain Boundaries” [24].

Chapter 6

Conclusion

We identified a useful method for identifiing a reasonable set of web-programming technologies by the use of the category ”web-programming framework”. The identified items contained in the set are well classifiable as web-application framework. We selected a subset of these frameworks based on a founded ranking for implementations in the ”101companies project”. The thesis therefore illustrates implementations of the frameworks JSF, Zend framework, Pyjamas, Google Web Toolkit, JBoss Seam, Apache Struts and Microsoft Silverlight (see tab.˜2.9). The frameworks introcude different architectural styles and concepts like Model View Controller or Client-server architecture, but also different technologies like Facelets or JBoss Application Server.

Although we conclude, that the language-centered approach introduced in the second chapter is not suitable for our purposes, we chosed interesting HTML5 technologies like XMLHttpRequest, Web storage and IndexedDB as additional interesting technologies for our project. These technologies are supported by further concepts and technologies like Ajax, MVC or jQuery. Nevertheless, the language-centered approach leads to technology sets without reasonable categorization and priorization. The result is difficult to overlook and we will not continue working on that approach.

We also conclude, that Wikipedia is very helpful to identify a core category for the web-programming domain. The resulting category ”web-application framework” is widely used and defineable in a strong way (see sec.˜2.1.1). We checked additional literature (see sec.˜3) to prove, that the category is a reasonable concept for web-programming (see sec.˜2.5). The list of remaining web-application frameworks is still long and there are many important and uncovered concepts and technologies left. For example, the implemented web-application frameworks do not yet cover all web-related languages like Ruby, Perl, Common Lisp or others (see tab.˜**??**). Exotic or new concepts are also not necessarily covered by the identification process. Therefore, the 101companies project needs more implementations to strengthen up the taxonomy.

Finally, web-application frameworks are very versatile in case of used programming languages, data exchange concepts and technologies. Although, the most used architecture in the implementations is MVC, the frameworks also leads to service-oriented or multi-tier architecture. The implementations are well documented within our project[1]. The documen-

[1] http://101companies.uni-koblenz.de/index.php/101companies:

tation describes, how the different technologies and concepts are organized and thereby helps to classify each used object.

`Project`

Appendix A

Terms and Technologies

A.1 Ajax

A.1.1 Intent

Concept for asynchronous requests in web applications

A.1.2 Description

Asynchronous JavaScript + XML [10] or *Asynchronous JavaScript and XMLHttpRequest* [30] (AJAX) is a concept for web programming. The core idea is to manipulate page content (HTML and CSS) instead of reloading the whole page. Hence, the advantage is a more fluent interaction between user and application. The basic language for Ajax is JavaScript, the basic technology is XMLHttpRequest. The data exchange is handled via asynchronous communication.

A.1.3 Technologies

The main parts of AJAX are:

- HTML, XHTML, CSS
- DOM
- XML, JSON, ... (depends on the message format)
- XMLHttpRequest
- JavaScript

A.2 MVC

A.2.1 Intent

An Architectural pattern dedicated to seperate concerns

A.2.2 Discussion

The MVC (Model View Controller) pattern divides a program into three major parts:

- View
- Controller
- Model

The view contains the user interface. The model contains the data. The controller handles the user input and notifies both the model and the view, if there are any changes in one of them. If there are changes in the model, the view will collect them directly from the model, when it gets notified. If there are changes in the view, the controller will delegate this to the model [3].

A.3 *Technology* JSF

A.3.1 Intent

Web-application framework for web-based user interfaces with Java EE

A.3.2 Description

"*JavaServer Faces* (JSF) is a standard Java web-application framework for building user interfaces for web applications" [30]. In terms of MVC it is focused on the view (JSP, Facelets), the model (Java Bean) and the controller (provided by the Servlet API). Ajax is included from version 2.0. *JavaServer Faces* controls the navigation between different pages and connects the pages to Java components implemented as Java Bean [30, 26].

A.3.3 Technologies

- Java Bean
- Servlet API
- JSP / Facelets

A.4 *Technology* Pyjamas

A.4.1 Intent

A Python-Web-application framework for JavaScript-web applications

A.4.2 Description

Pyjamas provides a Python-to-JavaScript compiler, and many other components including Ajax and a widget-set library. It is based on the idea of the GWT, which is creating JavaScript web applications without programming JavaScript code [18].

A.4.3 Technologies

- DOM
- XMLHttpRequest

A.5 *Technology* Zend framework

A.5.1 Intent

An MVC framework for web programming with PHP

A.5.2 Description

Zend framework is one of the most popular web-application frameworks for PHP. It offers a good infrastructure based on the MVC pattern and an amount of component libraries supporting web application development [19, 4].

A.5.3 Technologies

- REST
- SOAP

A.6 *Technology* GWT

A.6.1 Intent

A Toolkit for web programming in Java

A.6.2 Description

GWT is a web-application framework for developing JavaScript web applications in Java. It provides a Java-to-JavaScript compiler (see also: Pyjamas) as well as DOM and Ajax support in combination, and other useful components for web programming [11].

A.6.3 Technologies

- DOM
- XMLHttpRequest

A.7 *Technology* Apache Struts

A.7.1 Intent

A Java based Web-application framework with JSP and Servlets

A.7.2 Description

Apache Struts is a web-application framework highly dedicated to MVC. It affects every component of MVC, is it the view (JSP), the controller (Java actions in combination with the Servlet API) or the model (Java services). Apache Struts supports Ajax as well as REST and SOAP [1].

A.7.3 Technologies

- JSP
- Servlet API
- REST
- SOAP

A.8 *Technology* Seam

A.8.1 Intent

A Java EE based web-application framework

A.8.2 Description

JBoss Seam is an JSF based Ajax web-application framework for the development of rich web applications in Java. It is based on a three-tier architecture, where the presentation layer is handled by JSF [12].

A.8.3 Technologies

- JBoss Application Server
- EJB
- JSF
- JPA
- Hibernate

A.9 *Technology* XMLHttpRequest

A.9.1 Intent

A JavaScript API for HTTP-Requests

A.9.2 Discussion

XMLHttpRequest is an API for transfering any data over HTTP. Most commonly it is used in combination with message formats like XML or JSON [35].

A.10 *Technology* IndexedDB

A.10.1 Intent

A database management system API in the HTML5 ecosystem for client side data storage

A.10.2 Discussion

The JavaScript API IndexedDB provides client side data storage with the help of B-trees [36].

A.11 *Technology* Web storage

A.11.1 Intent

A JavaScript API for client side data storage in the HTML5 ecosystem

A.11.2 Discussion

There are four major differences in behaviour between cookies and Web storage [34, 13], which are shown within this implementation:

- the Web storage offers a largely enhanced amount of data (up to 5MB-10MB for web storage, 4KB for cookies),
- there is no need to use special parameters within the pages' URL to refer to a cookie with the required data,
- the client does not need to transmit data to the server with every request,
- and the data within Web storage never expires, if there is no explicit command to do so.

A.12 *Technology* Silverlight

A.12.1 Intent

A Web-application framework and a web browser plugin for interactive user interfaces

A.12.2 Discussion

Silverlight supports the development of rich-internet applications with focus on user interface and animations[22].

A.13 *Technology* WCF

A.13.1 Intent

A framework for asynchronous communication between web service endpoints

A.13.2 Discussion

Windows Communication Foundation (WCF) is a framework used for asynchronous communication between web services and clients. The supported message formats are single characters up to xml or binary files[23].

Bibliography

[1] Apache. *Struts*. Last visited on 17. January 2012. URL: http://struts.apache.org/.

[2] Engin Bozdag, Ali Mesbah, and Arie van Deursen. *A Comparison of Push and Pull Techniques for Ajax*. Online since 16. August 2007. http://arxiv.org/abs/0706.3984.

[3] Steve Burbeck. *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. Last visited on 06. January 2012. URL: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

[4] Federico Cargnelutti. *Zend framework architecture*. Last visited on 8. December 2011. URL: http://blog.fedecarg.com/2008/07/28/zend-framework-architecture/.

[5] Ezra Cooper et al. *Links: Web Programming Without Tiers*. In FMCO 2006r, pp. 266-296. Editor: Frank S. de Boer et al., Springe. ISBN: 978-3-540-74791-8. URL:http://dx.doi.org/10.1007/978-3-540-74792-5_12.

[6] Ian F. Darwin. *Java Web MVC Frameworks: Background, Taxonomy, and Examples*. Master thesis, Staffordshire University. Sept. 2004.

[7] DocForge. *Software Framework*. Last visited on 16. December 2011. URL: http://docforge.com/wiki/Framework.

[8] Barry Doyle and Cristina Videira Lopes. *Survey of Technologies for Web Application Development*. Online since 17. January 2008. URL: http://arxiv.org/abs/0801.2618.

[9] Wikimedia Foundation. *MediaWiki*. Last visited on 16. February 2012. URL: http://www.mediawiki.org/wiki/MediaWiki.

[10] Jesse James Garrett. *Ajax: A New Approach to Web Applications*. Last visited on 2. November 2011. URL: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.

[11] Google. *Google Web Toolkit*. Last visited on 17. January 2012. URL: http://code.google.com/intl/de-DE/webtoolkit/.

[12] Red Hat. *Seam Framework*. Last visited on 17. January 2012. URL: http://seamframework.org/.

[13] *Introduction to DOM storage*. Last visited on 27. January 2012. URL: http://msdn.microsoft.com/en-us/library/cc197062%28VS.85%29.aspx.

[14] Gerti Kappel et al., eds. *Web Engineering: The discipline of Systematic Development of Web Applications*. John Wiley and Sons, Ltd, 2006. ISBN: 0-470-01554-3.

[15] Ralf Lämmel. *101Companies features*. Last visited on 23. February 2012. URL: http://101companies.org/index.php/Category:101feature.

[16] Ralf Lämmel. *101Companies System*. Last visited on 10. February 2012. URL: http://101companies.org/index.php/101companies:System.

[17] Ralf Lämmel, Thomas Schmorleiz, and Andrei Varanovich. *101 ways to cut salaries*. 10 pages. Online since 28. April 2011. URL: http://softlang.uni-koblenz.de/101companies/cut101/.

[18] Luke Leighton. *Pyjamas: Python-based Web Application Development Framework*. Last visited on 17. January 2012. URL: http://seamframework.org/.

[19] Zend technologies Ltd. *Zend Framework*. Last visited on 17. January 2012. URL: http://framework.zend.com/.

[20] Ali Mesbah and Arie van Deursen. *An Architectural Style for Ajax*. In WICSA 2007. Publisher: IEEE Computer Society. ISBN: 978-0-7695-2744-4. URL: http://doi.ieeecomputersociety.org/10.1109/WICSA.2007.7.

[21] Leo A. Meyerovich et al. *Flapjax: a programming language for Ajax applications*. In OOPSLA 2009, pp. 1-20. Editor: Shail Arora and Gary T. Leavens, ACM. ISBN: 978-1-60558-766-0. URL: http://doi.acm.org/10.1145/1640089.1640091.

[22] Microsoft. *Silverlight*. Last visited on 25. February 2012. URL: http://www.silverlight.net/.

[23] Microsoft. *What is Windows Communication Foundation*. Last visited on 25. February 2012. URL: http://msdn.microsoft.com/en-us/library/ms731082.aspx.

[24] MSDN. *Making a Service Available Across Domain Boundaries*. Last visited on 24. January 2012. Jan. 2012. URL: http://msdn.microsoft.com/en-us/library/cc197955(v=vs.95).aspx.

[25] Robert C. Nickerson et al. *Taxonomy development in information systems: Developing a taxonomy of mobile applications*. In ECIS 2009, pp. 1138-1149. Editor: Susan Newell et al.. ISBN: 978-88-6129-391-5. URL: http://is2.lse.ac.uk/asp/aspecis/20090094.pdf.

[26] Oracle. *JavaServer Faces Technology*. Last visited on 17. January 2012. URL: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html.

[27] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. *Restful web services vs. "big"' web services: making the right architectural decision*. In WWW 2008, pp. 805-814. Editor: Jinpeng Huai et al., ACM. ISBN: 978-1-60558-085-2. URL:http://doi.acm.org/10.1145/1367497.1367606.

[28] Simone Paolo Ponzetto and Michael Strube. *Taxonomy induction based on a collaboratively built knowledge repository*. In Artificial Intelligence, June 2011, Vol. 175, pp. 1737-1756. Publisher: Elsevier Science Publishers Ltd., Essex. URL: http://dx.doi.org/10.1016/j.artint.2011.01.003.

[29] Dirk Riehle. *Framework Design: A Role Modeling Approach*. PhD thesis, ETH Zürich, number: 13509. 2000.

[30] Chris Schalk. *JavaServer faces : the complete reference*. New York, NY: Mcgraw-Hill, 2007. ISBN: 0072262400.

[31] Tony Chao Shan and Winnie W. Hua. *Taxonomy of Java Web Application Frameworks*. In ICEBE, pp. 378-385. IEEE Computer Society, 2006. ISBN: 0-7695-2645-4. URL: http://doi.ieeecomputersociety.org/10.1109/ICEBE.2006.98.

[32] *THING-MODEL-VIEW-EDITOR: an Example from a planningsystem*. Last visited on 06. January 2012. URL: http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf.

[33] W3C. *HTML5*. Last visited on 10. February 2012. URL: http://dev.w3.org/html5/spec/Overview.html.

[34] W3C. *Web storage*. Last visited on 27. January 2012. URL: http://www.w3.org/TR/webstorage/.

[35] W3C. *XMLHttpRequest*. Last visited on 20. January 2012. URL: http://www.w3.org/TR/XMLHttpRequest/.

[36] *W3C indexedDB*. Last visited on 10. January 2012. URL: http://www.w3.org/TR/IndexedDB/.

[37] Andreas Wende. *Klassifikation und Bewertung von Frameworks für die Entwicklung von Web-Anwendungen*. Diplomarbeit, Universitt Leipzig. Sept. 2005.