Applying Mashups to a Digital Journal

Jörg Dissauer

Applying Mashups to a Digital Journal

Diploma Thesis

at

Graz University of Technology

submitted by

Jörg Dissauer

March 2008

Institute of Information Systems and Computer Media (IICM)

Graz University of Technology

8010 Graz, Austria

Assessor/Advisor:	Dipl.-Ing. Dr.techn. Univ.-Doz. Denis Helic
Advisor:	Narayanan Kulathuramaiyer, M.Sc.

Anwendung von Mashups auf ein digitales Journal

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Jörg Dissauer

März 2008

Diese Arbeit ist in englischer Sprache verfasst.

Institut für Informationssysteme und Computer Medien (IICM)

Technische Universität Graz

8010 Graz, Österreich

Begutachter/Betreuer:	Dipl.-Ing. Dr.techn. Univ.-Doz. Denis Helic
Betreuer:	Narayanan Kulathuramaiyer, M.Sc.

Abstract

The term "Web 2.0" summarizes recent developments on the World Wide Web (WWW) which revolutionized the way we use the Internet today. Among these developments is a design paradigm called "Mashup" which is used to mix data sources from all over the web to derive new value from this data. This thesis explores the use of this emerging application developing method applied to a particular digital journal, the Journal of Universal Computer Science (J.UCS). This serves as a novel way of publishing scientific work and a convenient mechanism for retrieving publications. A study of Web 2.0 and Mashups was carried out to illustrate the underlying technologies to be used.

A prototype system has been built based on the completed study. The Mashup employs an application programming interface (API) from an external source to embed data of authors and editors of J.UCS into a geographical map, producing a tool that serves both as an enhanced J.UCS data administration tool and a decision making tool.

The evaluation of the prototype Mashup revealed an incompleteness of the J.UCS data sets with regards to geographical mapping information. A method was then applied to acquire that data automatically with a further data collection via community inputs. A significant increase in the successful mappings of J.UCS information was achieved. Moreover concerns about the misuse of a social computing environment were also dealt with.

Kurzfassung

Der Begriff “Web 2.0” ist eine Zusammenfassung von neuen Entwicklungen im World Wide Web (WWW), welche die Art, in der wir das Internet heute benutzen, revolutionierten. Unter diesen Entwicklungen befindet sich ein Design-Paradigma, genannt “Mashup”, das verwendet wird, um Datenquellen aus dem ganzen Web zu vermischen, um daraus neue Information abzuleiten. Diese Arbeit erforscht den Nutzen dieser aufkeimenden Entwicklungsmethode angewandt auf ein spezielles digitales Journal, das Journal of Universal Computer Science (J.UCS). Dieses stellt einen neuartigen Weg des Publizierens wissenschaftlicher Arbeit und einen bequemen Mechanismus des Beschaffens dieser Publikationen dar. Es wurde dazu eine Untersuchung von Web 2.0 und Mashups durchgeführt, um die Technologien, auf denen diese Konzepte basieren, zu illustrieren.

Ein Prototyp-System wurde auf Basis dieser Untersuchungen entwickelt. Das Mashup nutzt die Applikations-Programmierschnittstelle (API) einer externen Quelle um Daten von Autoren und Editoren von J.UCS in eine geographische Karte abzubilden und kann damit als ein Werkzeug sowohl zur verbesserten J.UCS Datenadministration als auch zur Entscheidungsfindung genutzt werden.

Die Evaluierung des Prototypen-Mashups brachte die Unvollständigkeit der J.UCS Datensätze in Bezug auf geographische Abbildungsinformation ans Licht. Daraufhin wurde eine Methode zur automatischen Akquirierung dieser Daten appliziert und um eine Datensammlung durch die Eingabe der Benutzergemeinschaft erweitert. Dadurch wurde eine signifikante Steigerung in der Zahl der erfolgreichen Abbildungen von J.UCS Information erreicht. Desweiteren wurden Bedenken über den Missbrauch einer “Social Computing”-Umgebung behandelt.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Acknowledgements

Most of all I would like to thank my girlfriend Doris for her persistent belief in me and the support she gave me throughout the time of writing this work.

I would also like to thank my parents for giving me the possibility of getting a higher education and funding my studies without restrictions resulting in the completion of the thesis.

Furthermore, I have to thank my supervisors Narayanan Kulathuramaiyer and Denis Helic, who gave me the chance to write this thesis, guiding an inexperienced writer on the way to its accomplishment and for always having an open ear for my concerns and giving me input with their expert knowledge and professional attitude.

Last but not least, I would also like to say thanks to the whole team developing and administrating the Journal of Universal Computer Science, where always provided me with a friendly atmosphere to work in.

Table of Contents

List of Abbreviations

ACM	Association for Computing Machinery
AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
ASP	Active Server Pages
CMS	Content Management System
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management System
DOM	Document Object Model
DTD	Document Type Definition
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IICM	Institute of Information Systems and Computer Media
J.UCS	Journal of Universal Computer Science
JDBC	Java Database Connectivity
JSP	Java Server Pages
JVM	Java Virtual Machine
NF	Normal Form
ODBC	Open Database Connectivity
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
PS	Postscript
REST	Representational State Transfer
RSS	Really Simple Syndication
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
URI	Uniform Resource Identifier
UTF	Unicode Transformation Format
VBScript	Visual Basic Scripting Edition
W3C	World Wide Web Consortium
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformation

1. Introduction

1.1. The Programmable Web

Recent developments on the World Wide Web lead to new web usage paradigms, which handover the power of shaping the internet from information providing experts (like media corporations or professional web engineers) to the end-user. Nowadays, information on the web is often created using community driven activities, e.g. blogs, wikis, podcasts or social networks. Additionally, web services emerge, which resemble more and more old-fashioned desktop applications in their functionality and even their speed.

A network like the internet consists of many nodes and to gain benefit from such a network nodes should join forces because in most cases combined knowledge values more than the sum of individual knowledge. On the WWW each node is reachable from every other node but co-operations with other nodes hardly take place. But wouldn't it be nice if an Internet user could combine his activities like writing a blog and hosting photos to write a blog entry on a web album stored on a Web node at a completely different place? Although not realized by the majority yet, this is already possible.

Nowadays, many companies like Google[1] make their application programming interfaces (APIs) of their web services publically available. Often these APIs are quite simple to use and deeper insight into web technologies like server-client communication knowledge is not needed. Thus, even inexperienced programmers can use them to incorporate external information in their web sites.

It can be said that people mash up data from different sources. Consequently, the services that emerge are called "Mashups". With the possibility of creating Mashups, the World Wide Web has become programmable.

[1] http://www.google.com/

1.2. Computer Science in a Digital Journal

The internet plays an important role in research and development these days. The publishing process of scientific publications in classically printed journals can take years in worst case. Furthermore, the access to these journals is quite difficult, as they are often too expensive to subscribe to for the average reader like a student, and accessing these journals at public libraries can also take a long time. With the internet at hand, journals took their chance on publishing in an electronic way and harness the power of computers in their business processes nowadays, leading to shorter paper publishing cycles and easier access to them. Journals represented on the Internet are therefore labeled as digital journals or digital libraries.

Among them is the Journal of Universal Computer Science[2] (J.UCS), a digital journal freely available on the WWW which publishes articles of experts on computer science subjects and provides browsing methods of digital journals or libraries and a web service where users can search for articles and retrieve them by means of standard search. The creators of J.UCS aim at extending J.UCS with innovative features in its web service to stand out from the mass of electronic journals:

"As J.UCS develops it is clear that additional demands will be placed on it. … including interfaces with other applications, dynamically updated glossaries or bibliographies and the like." [Cal94]

[2] http://www.jucs.org/

1.3. Motivation and Objective of the Work

Following this philosophy, the host of J.UCS, the Institute for Information Systems and Computer Media[3] (IICM) of the Graz University of Technology, wants to take advantage of the fresh web development approaches and accepts the challenge to incorporate them into a useful application. Therefore this thesis tries to find third party APIs suitable to enhance the journal and it fathoms the benefits of Mashups using them in connection with the information stored in J.UCS. For these studies and also as a base for further developments a prototype Mashup has been built, which is the subject of discussion in this thesis.

The resulting Mashup should assist in the management and administration process of the journal. An attempt to realize this is by adding value through visualization of the digital library's data, which is usually only shown in static text-based views. Moreover, the prototype should utilize community driven activities to find out new ways of enhancing data of J.UCS and harness knowledge distributed on a large number of individuals.

1.4. Structure of the Work

Chapter 2 introduces Web 2.0 as a concept that provides the average internet user with the ability to create and share content and elaborate with others. It discusses the tools that are used and the philosophy that stands behind it.

The remixing of different data sources into applications called Mashups which give the mixed data a new meaning are explained in chapter 3. Furthermore, examples of Mashups and available data sources are given, the role of Mashups on the web are presented, and emerging tools that assist in Mashup development are introduced.

While chapter 2 and 3 present the philosophical and creative idea behind Mashups, chapter 4 focuses on the necessary technologies for Mashup or other Web 2.0

[3] http://www.iicm.edu/

applications. XML and databases are introduced as ways of structuring and storing data and the architectures to incorporate this data into web applications are discussed.

The description of digital journals especially J.UCS and the requirements that such a journal has and the kind of requirements that are demanded are explored in chapter 5. J.UCS' operating environment are examined and its strengths and weaknesses are listed. Finally, the chapter discusses the motives for enhancing J.UCS with external data sources.

Chapter 6 presents the implementation of a J.UCS Mashup prototype that visualizes authors and editors of the journal in a geographical map. It starts with a situation analysis in which the requirements of the prototype are determined and first problems are detected and coped with. Afterwards, architectural design is given and described both on a system and a functional level.

An evaluation of the implemented application is carried out in chapter 7. Problems are identified, discussed and solution are provided.

Finally, chapter 8 and 9 conclude the thesis, whereby chapter 8 summarizes, draws conclusions, and gives an outlook on how the evolution of J.UCS could continue and which sustaining effect Mashups have on the WWW. Chapter 9 deals with personal subjects the author has drawn from the work on this thesis.

2. Web 2.0

The World Wide Web in its current form has recently undergone a massive transformation and the way how people use the internet and how the Web presents itself to the user has changed significantly. Web 2.0 is a catchphrase people encounter in the media today, which often entitles it “the people’s web” as well. However, the majority still has no clue what Web 2.0 means and how the public can profit from it, even though a high percentage already uses it without knowing.

The application that was built for this thesis deals with a part of Web 2.0 and therefore this section introduces the reader to the subject. It discusses how the WWW was used before Web 2.0 and how it transformed to the Web as we know it today. Finally, an overview of the main characteristics of “the people’s web” is given, the kind of information and content production facilities that are existent are listed, and the ways web users consume the web are discussed.

2.1. The pre-Web 2.0 Era

The “old” Web was based on a rather centralized model, where one served as the information source for many. [Kol06] states that „*Previously, most information was, by and large, offered by professional information providers such as companies advertising their products and services, organizations or news services. In addition, private users on the web had the option of establishing personal homepages as well.*” People relied on this information and although users already took part in discussions on networks like the Usenet by using newsreaders, it was still very hard to interact with other users on the web or influence the information which was published. There were no means of combining knowledge from different users all over the world so information got more accurate, complete, and reliable due to the much too complicated tools and lack of infrastructure and technical background knowledge. [Kol06]

The internet was simply a mapping of traditional media or places of commerce to web services, where new features based on the power of the net were missing. Conclusively, the so called dot-com bubble – the first relevant occurrence of web companies on the business stage and the introduction of them to the stock exchange - burst and many internet companies lost a fortune or even vanished completely.

2.2. Web 2.0 - The People's Web

"Many people concluded that the web was overhyped, when in fact bubbles and consequent shakeouts appear to be a common feature of all technological revolutions. Shakeouts typically mark the point at which an ascendant technology is ready to take its place at center stage. Leading this ascension were companies that had survived the collapse, which seemed to have some things in common." [ORe07]

Companies like Amazon[4] or Yahoo[5] survived the crash by providing features, which many of the successful web sites and web services that emerged from the ruins of the dot-com bubble copied and evolved: they gave the user the ability to actively take part in dealing with information (e.g. Amazon allows the user to rate products and to create recommendations like reading lists etc. or Yahoo's directory service for the WWW was mainly built by users).

Following the invention of giving the user the ability to get more involved in creating content and through recent technical refinements and advancements more and more systems arose, which share the common characteristic that the architectural approach is "bottom-up" rather than "top-down". This approach yields that the content and the appearance of this content through styling means is done by people of the user community instead of being defined by professional, corporate information providers. As a consequence, self-organization emerges inside these systems resulting in the appearance of advanced structures. Such systems are profiting from the fact that the knowledge of a community is always larger than the sum of knowledge and experience of the single members of this community. [Kol06]

[4] http://www.amazon.com/
[5] http://www.yahoo.com/

Due to this evolution all of these community driven systems, which embrace the principle of “the web as a platform” and where content was provided by the mass and not by the media were combined under the term Web 2.0, a term made popular mainly by the O’Reilly Media corporation[6].

Figure 2.1 shows a meme map of how [ORe07] visualizes the concept of Web 2.0 with its core principles (in the box). An additional set of practices and examples of sites or services that follow some or all of these practices, which are more or less tightly connected to the core, show which strengths could be derived from this core.

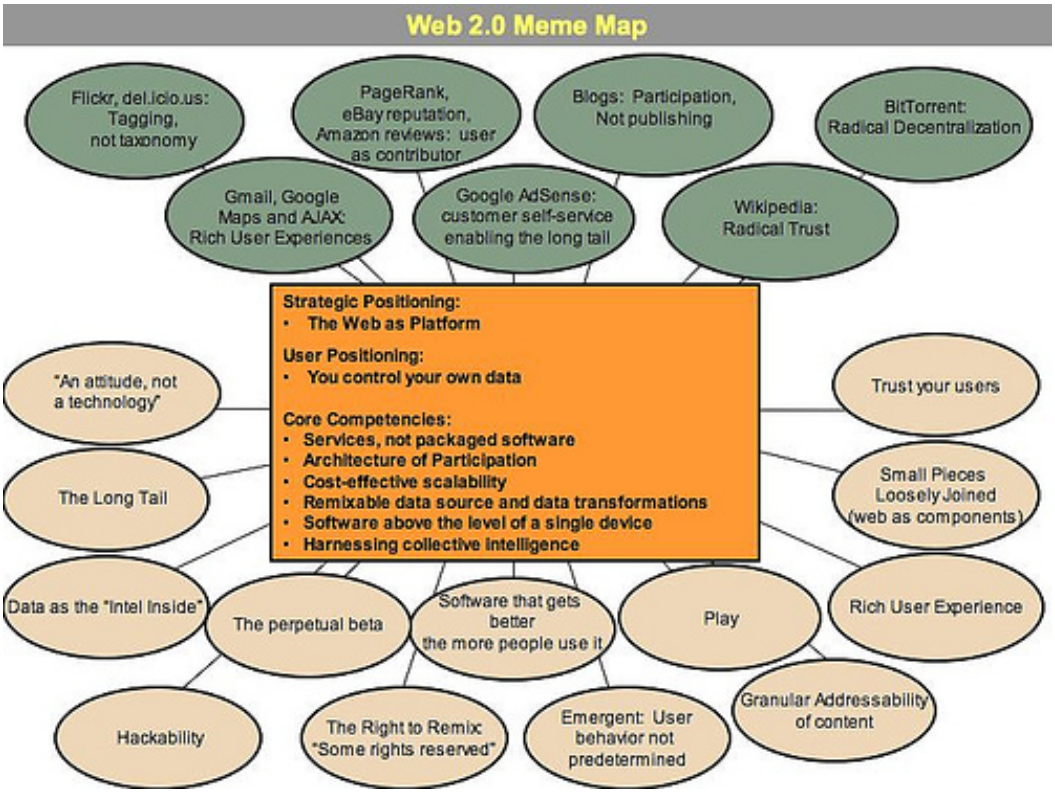


Figure 2.1: Meme map of Web 2.0. Source: [ORe07]

In the following subsections community driven systems like blogs, wikis, podcasts, file sharing systems, and social networks as well as the most important characteristics should explain the concept of Web 2.0 as shown in Figure 2.1.

[6] http://www.oreilly.com/

2.2.1. Services, not Packaged Software

As already indicated by the title of this section, web applications in the Web 2.0 style are services which are never sold or packaged. The services don't have scheduled new releases, but are steadily improved. "*Neither are these services for sale nor licensed; customers pay directly or indirectly"* for the use of a service. [ORe07]

"As a consequence operations must become a core competency and the system must be maintained 24 hours a day, e.g. Google must continuously crawl the web to update its indices and filter out link spam and other attempts to influence its result." [ORe07]

The continuous improvement introduces the users as co-developers and testers. They give input, which features they wish to use in the service and also test and either accept or decline improvements. The service gets into a state of being a permanent beta version, which [ORe07] denominates "the perpetual beta".

2.2.2. Harnessing Collective Intelligence

Although sharing information and thoughts has already been possible on the World Wide Web in the pre-Web 2.0 Era, content is now far easier to create and share among users by providing tools which are aggregated under this Web 2.0 called new concept. In [Wei05] the opinion is found that this opening of production means to non-programmers introduced *"that highly combustible fuel – critical mass"*.

This critical mass is the threshold of the number of participating users in a process where the success of this process emerges. The need for this critical mass as the success factor occurs on the internet in various kinds of processes. [ORe07] lists eBay[7] as an example that grows through the activities of users and eBay uses a critical mass of users whom it reached long ago to make the occurrence of competition besides them almost impossible.

In the production process of information the collaboration of many people lead to higher quality information on a topic than it would have been if a single person had produced

[7] http://www.ebay.com/

and edited it. Such a piece of knowledge is created by the use of collective intelligence. As described in section 2.2.7 Wikipedia is a Web site harnessing this method by letting the users build an encyclopedia.

2.2.3. Data as the "Intel Inside"

"Database management is a core competency of Web 2.0 companies, so much so that we have sometimes referred to these applications as 'infoware' rather than merely software." [ORe07]

These databases can turn out to be high-value sources (in terms of information quality and monetary), because with the other characteristic of Web 2.0, the right to remix, new web services appear using exactly this data in combination with other sources resulting in innovative Mashups (see chapter 3).

2.2.4. Rich User Experiences

Following [ORe07], a rich user experience means that a web based application should have interactivity like desktop applications and a rich user interface. Although it was already proposed in the early years of the World Wide Web, web developers are finally able to build web services with these strengths using the AJAX paradigm (as described in section 4.4).

2.2.5. Remixable Data Sources and Data Transformations

With the credo "the web as a platform" Web 2.0 sites should be modeled in a way that their web service and data interface could easily be reused by others to support a loose coupling of systems.

As a consequence, the policy of "*the Right to Remix*" [ORe07] is derived: Interesting new systems could emerge from collective adoption, so developers of web services should keep access barriers to their APIs low and therefore giving others the possibility to "mash" their initial system up with other services. Thus, following existing standards in web service development must be seen as a commandment in Web 2.0 systems.

2.2.6. Blogs

Blogs (short for Weblogs) are the Web 2.0 crossing between diaries, newsgroups, newspaper editorials, and hotlists where owners (one author or a small group of authors) write down information important to them on a regular base. Thus in blogs people express their point of view and publish their opinion on certain topics. [Kol06]

These features do not differ from the diaries or the "What's new" part on personal homepages from the early internet. However, blogs stand out by giving readers certain power that turns blogs into an innovative idea and accessing information by using blogs with the features mentioned below is therefore sometimes called "*the incremental web*" [skr05] or "*live web*" [ORe07].

First of all there is the advantage that weblogs are easier to set up and maintain by using several freely available tools than static content on personal homepages. By using RSS feeds one can subscribe to blogs and can therefore always be kept up to date. The advantage of a link to weblog compared to i.e. a bookmark or a link to an old-fashioned website is that it is "*expected to point to a perennially changing page, with 'permalinks' for any individual entry, and notification for each change"* [ORe07]. It is therefore a much tighter bond to the actual entry compared with other types of links. [Kol06], [ORe07]

"Blogs contribute to Web content by linking and filtering evolving content in a structured way and by establishing interlinked communities - the blogosphere - connecting people through shared interests. Bloggers can link to news feeds, personal journals, and topic-specific blogs of almost every sort". [Lin03]

With the additional function of trackbacking (bloggers get notified when someone else is referring to his article in another blog), the blogosphere could be seen as a "*peer-to-peer equivalent to the Usenet or to bulletin-boards"*. [ORe07]

2.2.7. Wikis

In [Kol06] wikis are described as "*[…] self-organizing web-sites where everyone can add and edit content"*, which means that every user also becomes an author. By giving users the possibility to edit content added by other users, wikis rely on the concept that a great number of authors and a sufficient number of changes guarantee error correction and completeness of content stored in the wiki.

Wikis can be seen as "*web based content management systems (CMS) for generating web-pages that contain text, sound and similar media objects as well as hyperlinks to internal and external resources"* [Kol06]. Content is added using a wiki-specific markup language, which also determines which styling and content features the wiki supports. Additionally, the articles are under version control because even by editing or completing articles semantic errors can be made (often on purpose) and therefore fallbacks to older versions are possible.

Wikis are used in various areas, for instance as a way of managing projects, whereby project stakeholders add and edit content, or as learning environments, where in contrast to previous learning environments, where content was given by teachers or professional information providers, now also students and pupils themselves contribute information. Therefore information wiki users provide does not need to be complex. Even small chunks of data and little fragments of knowledge become a whole at a time.

Probably the largest and best known wiki is Wikipedia[8], which calls itself the "free encyclopedia", which is a wiki-based encyclopedia. Everybody has the right to read, write or edit articles. By February 2008, Wikipedia claims to keep more than 2 million articles in English and more than 700.000 articles in German language. Wikipedia exists on the belief that "*with sufficient critical mass, truth would arise from consensus*" [Wei05].

[8] http://www.wikipedia.org/

One of the biggest problems wikis like Wikipedia have to deal with is the legitimate critic that it is almost impossible to guarantee high quality standards, due to the reason that almost everybody can change information and the danger exists that people take this information for granted and propose false knowledge in their own contributions (i.e. students cite Wikipedia articles in master theses). Quality can be enhanced by applying user hierarchies to a wiki as suggested by [Kol06], where users on a certain hierarchical level can only edit articles from authors on the same or on lower levels. Wikipedia is planning to tag an article as stable once it has reached a certain quality and is also in the evaluation of a system where users can rate articles. [Gil05]

2.2.8. Podcasts

The artificial word podcast, a combination of Apple's popular MP3 Player iPod and the term broadcasting, is an audio equivalent to blogs. Single persons or a group of people produce podcasts on a regular basis, whereby they reflect certain issues by means of audio. This content is provided in the MP3 format and comes with additional metadata like the title or an abstract of the topic. Like in blogs, people can retrieve a single podcast or subscribe to the regularly produced podcast, thus making it an "audio-on-demand" system. [Kol06]

Examples for podcasts are entertainment or news programmes. Universities are even starting to provide podcasts, where students can re-listen to lectures or prepare for them (i.e. [WCL07]).

Analogue to podcasts, services are emerging, which provide "photocasting" or "vodcasting", which is a synonym for videocasting.

2.2.9. File Sharing Tools

Following [Kol06], people usually associate file sharing with distributing and downloading pirated digital content using applications like BitTorrent or EDonkey, which are used to share mostly illegal content over the Internet indeed.

In recent times though, also peer-to-peer file sharing tools appeared, whereby people share private documents like photographs or videos. Best examples for that are Flickr[9] (for photographs) and YouTube [10] (for Videos). These tools are web-based systems, where people have their own private space where they can upload their videos and photos. One of the most important features is the possibility to attach tags to the content, which means that certain keywords are associated to the pictures and videos, giving the system a user-steered categorizing feature. This yields that they are "*self-organizing communities where the system doesn't tell a user how to organize and tag his content"* [Kol06]. Unfortunately, caused by this missing clear defined taxonomy, tags can be ambiguous, e.g. if a picture is described with the tag "rock" it could be a mineral as well as related to the music style. Sometimes this can be handled though by adding additional keywords to a search. [Kol06]

2.2.10. Social Networking

It is in human's nature to have needs that want to be satisfied. Among these are the social needs esteem and affiliation. Therefore people established communities reaching back thousands of years. The Internet now gives mankind a new way of building these communities.

Social networking services on the web offer internet users a fresh new possibility to maintain and build up relationships and share their life in form of information with others. For that purpose users are required to create a personal profile, which contains personal information like name, date of birth, hobbies, and a photo. This profile is made public on the network, thus "*other users can identify them as friends and add them to their contact list*." [Kol06]

One of the ways to establish new friend- or relationships leads through relationship-chains where people get to know each other because they are a friend of a friend. Another way is to participate in the activities these services offer. Among these activities is the participation in discussion groups on certain topics.

[9] http://www.flickr.com/
[10] http://www.youtube.com/

Although social networks seem to revolute the way of social interaction, concerns regarding the privacy of people's data exist because service providers can utilize the personal data. [Kol06] list threats like

- Dissemination of personalized fraudulent advertisements,
- Automatic sign up to services matching a user's profile or
- Selling of personal data to third parties

Examples for social networking services are MySpace [11], Facebook [12], or the recently under German-speaking students becoming popular StudiVZ [13]. All of these networks host several million people (MySpace is already succeeding the 100 million user threshold). Apart from contacting users and taking part in discussions, users can also add multi-media content to their personal space, like photos or as in the case of MySpace even audio and video content.

Other kinds of social networks are incorporated by eBay or Amazon. On eBay, people's auction behavior is influenced by ratings of sellers and buyers because everybody that had to deal with another user can rate this interaction and therefore a network of weighted connection evolves, where the weight depends on positive, neutral or negative ratings between buyers and sellers. Amazon gives users the possibility to review and rate articles they bought and to create a list of similar articles they recommend, resulting in clusters of users with similar interests. [Kol06]

[11] http://www.myspace.com/
[12] http://www.facebook.com/
[13] http://www.studivz.net/

3. Mashups

According to the demand of “remixable data source and data transformation” for web sites that want to comply to the Web 2.0 paradigm as introduced in section 2.2.5, this section discusses the mechanisms of this remixing process and the possibilities emerging from it. The resulting products are denominated as “Mashups”.

3.1. The Usage of Mashups

Originally, the term Mashup is a construct from the music scene used when tracks of different artists were crossed and mixed together, but the term has recently appeared among the community of web service developers. In short, a Mashup is the remix of online data sources and services, ending in a totally new application featuring a rich user experience – richer than the single services or data sources alone.

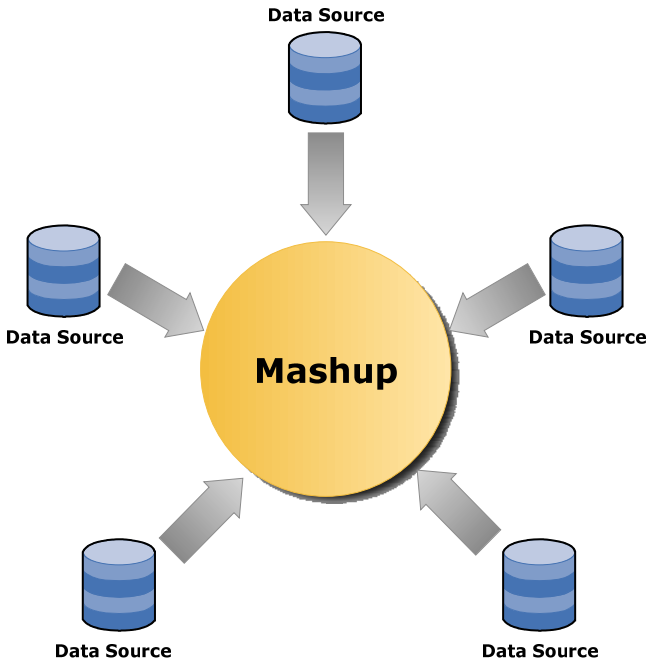


Figure 3.1: Data is remixed in a Mashup application

Data for Mashups is usually provided using an application programming interface (API) from a third party, but still it is the creativity of the Mashup creator that gives the combined provided services new value and semantic. Other forms of data sources are RSS feeds, XML messages (i.e. retrieved in a SOAP envelope) or local databases. Additionally, there are situations where data is not available but instead "*user inputs are used to dynamically acquire data*" [Kul07]. Obvious benefits gained by Mashups are well described in [Kul07]:

"*Mashups provide an added dimension by incorporating facilities for user-driven design and development of new applications. Mashups stimulate web development by allowing anyone to combine existing data from sources like eBay, Amazon, Google, Windows Live and Yahoo in innovative ways and therefore support a structuring capability for the integration of web content and application.*"

[Wes07] states that the age of mass media and mass production is nowadays followed by customizations for single consumers, which follow the principle of Web 2.0's "long tail" (which means that due to the large distribution of the internet, even niche products find their customers). This process was driven through the immense simplification of

producing, designing, and publishing of content. Nearly everything can be disassembled, personalized, and cross-linked. Assembling Mashups is such a do-it-yourself process, representing in a strong way the technical and creative possibilities on the internet. However, due to the fact that various external sources are combined, availability becomes a crucial factor. So if one of the services used is out of operation, the whole Mashup can fail to work. For this reason it is often a temporary problem due to maintenance work at the company providing a part of the data but services could also shut down their system at all which could also lead to the death of the Mashup if no similar source can be found (otherwise at least reengineering efforts need to be accomplished). A Mashup developer should have in mind as well that programming interfaces could change.

3.2. Chances for Data Providers

The Mashup approach frees web service developers of the burden of building their own data repositories by delegating this to professional data providers (e.g. Google Maps[14] as the provider of geographical mapping data). The question arises, which profits data providers could gain from offering access to their repositories. Following [Wes07], every new Mashup is free marketing for third party web services used in the Mashup. Every user increases the volume of usage and adds additional traffic, which can be converted to money.

For example Google Maps would only be one mapping service of many, if the access API did not exist. Instead it is a platform to all kinds of geographical applications, which gathers money through advertising income due to the traffic to and from the Mashups. As another instance, e-commerce platforms like eBay or Amazon profit from the usage of Mashups incorporating their web services, because the offering of their data and products add business to them [Wes07]. Other types of data from third parties apart from mapping and e-commerce data include search results, blog entries, media objects like video or photos, newscasts or medical repositories.

[14] http://maps.google.com/

3.3. Examples of Mashups and their Sources

As mentioned earlier, providing an API to their web services marks an excellent chance for companies to increase commercial coverage and hardly any of the big players on the web refuse to open their services for remixes. As a consequence, a lot of sources have already been tried to cross and this section should give a brief overview of what is possible.

As seen in figure 3.3, Google Maps is the API that is almost used in every second Mashup. One of the first Mashups developed using Google Maps, which was also one of the sparks that ignited the hype on Mashups, is HousingMaps[15]. It combines Google Maps with real estate adverts from Craigslist[16] - a web service where adverts are listed to help people find "*just about anything*" [Cra07]. It is one of the best examples to show how the combination of two web services can create a useful new application or enrich the data from one service with information from the other.

Although meta-search engines such as Amazon's A9[17] are aggregating information from various multiple existing search engines, their interface could be reused to build Mashups with a custom interface to suit individual needs and that serve as a more personalized portal [Kul07].

Mashups dealing with music are great examples, where much more than 2 data sources can be used. A great example for this is FoxyTunes Planet[18], which serves – contrary to using the API of a meta-search engine - as a kind of meta-search engine for music related content itself. Users can browse or search for artists and FoxyTunes aggregates information about this artist like music videos, lyrics, photos or shopping items using data sources like YouTube, LyricWiki[19], Flickr, Amazon, Google, Last.fm[20] and more.

[15] http://www.housingmaps.com/
[16] http://www.craigslist.org/
[17] http://www.a9.com/
[18] http://www.foxytunes.com/planet
[19] http://lyricwiki.org/
[20] http://www.last.fm/

Another Mashup dealing with music – but also with other types of content from the entertainment business like movies or actors - is LivePlasma[21]. Although it is not showing links to videos or photos, instead it shows the possibility that gathered data (in this case through the Amazon API) can be used to visualize relationships, e.g. by building meaningful information maps.

A lot of Mashups use a photo storing and displaying web service like Flickr. Obvious Mashup ideas like attaching pictures associated with a certain location to a map have been realized quite often. However, people also create innovative Mashups like FlickrFling[22], an RSS Reader that parses a feed and picks certain words, which it sends to Flickr to retrieve pictures tagged with these words. It then assembles the content of the feed as a sequence of pictures instead of displaying text.

Not only APIs are used as an interface to data, but also RSS feeds serve as a source of information of Mashups. Mashups can for example easily overpower the functionality of simple RSS readers by providing a robust RSS management (e.g. fwicki[23]) tool, which offers high customization and additional features like information analysis of feeds, i.e. to build relationships between topics.

3.4. The Role of Mashups on the Web

One of the best resources on the web on Mashups is ProgrammableWeb[24], which contains links to Mashups and APIs as well as statistical data and Mashup-creating tutorials. ProgrammableWeb tracks more than 2800 Mashups and more than 650 APIs. At the moment, the number of Mashups increases by an average of 3.14 per day.

[21] http://www.musicplasma.com/
[22] http://www.nastypixel.com/prototype/cms/myfiles/pages/flickrfling/
[23] http://www.fwicki.com/
[24] http://www.programmableweb.com/

Figure 3.2: Top 10 Tags used for Mashups at ProgrammableWeb [Pro07]

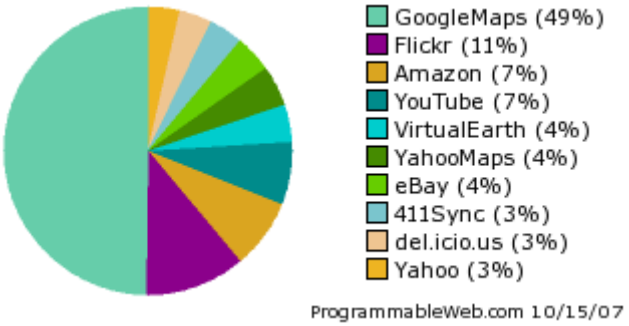


Figure 3.3: Top 10 API'S used for Mashups at ProgrammableWeb [Pro07]

ProgrammableWeb also hosts the so-called “Mashup Matrix”, a two-dimensional grid where both X- and Y-axis indicate an API. Thus each point in the grid is a representation of a tuple of APIs, which keeps a list of Mashups combining these two APIs (see [Pro071]).

Figures 3.2 and 3.3 show the distribution of the top categories of Mashups and the most frequently used APIs. They illustrate the importance of Google Maps for the emerging of Mashups as a new web development paradigm.

However, compared to the daily increase rate of other Web 2.0 concepts like blogs (175.000 according to [Tec07]), an average of 3.45 new Mashups a day as tracked by ProgrammableWeb is disillusioning low. One of the main problems why the mass of web users shies of creating Mashups is that there is a lack of robust development tools that make life for developers easier (see [Kul07]). Not only is this fact keeping more users to create Mashups, but also the mixing of services by means of hacking Javascript and

server-side scripts seems to be cumbersome for the ones who dare to mash. Due to this fact, Mashups mostly incorporate only one external data source, while in theory multiple ones can be combined. Nevertheless will the remixing of different web services play an important role on the WWW in the near future and smart companies will soon offer the tools to make life for Mashup developers easier. A glimpse into this future is given in the next section.

3.5. Mashups in the Future Web

The World Wide Web contains an immense amount of information, thus it is nowadays often thought of as a huge database, like [Isk071] does. However, like for relational databases, a database management system (DBMS) is needed to combine data from different sources and give the information a new meaning. In above sections API driven Mashup building was introduced as a way of mixing up data from different Web sources just like combining data from different tables in a relational database, which provided a way of managing information stored in the database we call the World Wide Web.

Building Mashups using API's require high programming skills though and so they are quite useless for a regular person, who wants to mix up data sources from all over the web. Another point is that most information on the Web is not accessible over an API, so only a small part of the WWW is remixable.

In [Isk07], the vision of gathering data for Mashups easier in the future is stated. "Web 3.0" is the keyword and its meaning is that "*major web sites are going to be transformed into web services – and will effectively expose their information to the world*" [Isk07].

Information which is not accessible over an API will be available using tools which are already in use but still have a long way to go until they have reached a mature level. Among this tools are online services called Dapper[25] or Yahoo! Pipes[26] as well as desktop environments like Open Kapow[27] and the already mentioned RSS managing tool

[25] http://www.dapper.net/
[26] http://pipes.yahoo.com/
[27] http://openkapow.com/

fwicki (see section 3.3). These tools are guided by the aim to convert unstructured information into structured information and to make queries and remixes of the data possible.

3.5.1. Dapper

A way to convert unstructured to structured data is by taking advantage of a technique called “Web Scraping” , which is done by Dapper. Scrapers analyze HTML pages and their markup to determine relevant data and its relations and return this data in a structured data format like XML. In case of using Dapper, the user is provided with a GUI that assists in gathering data from pages the user defined.

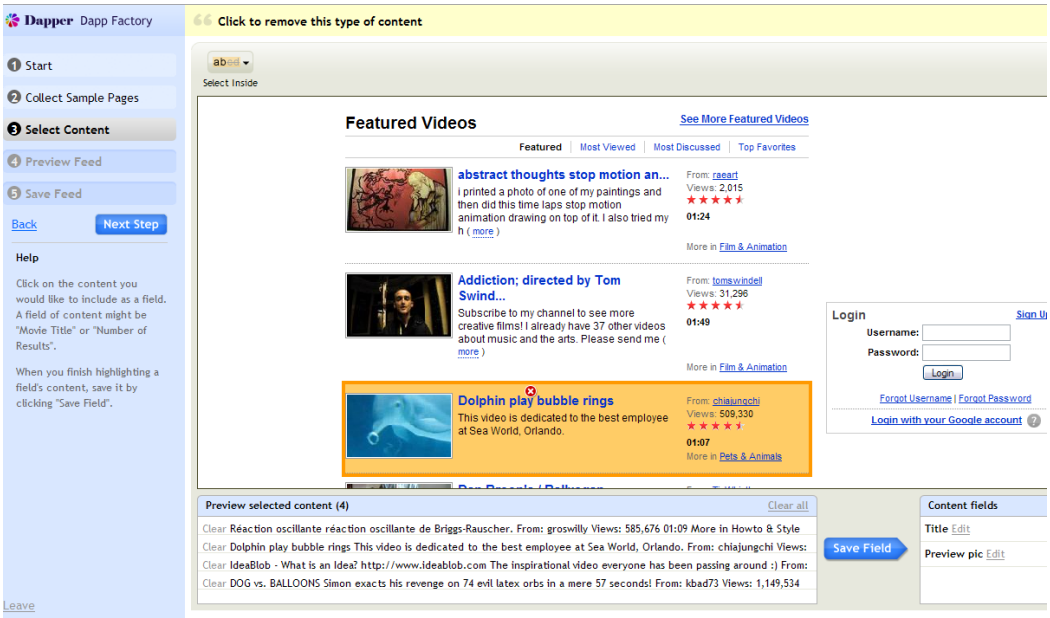


Figure 3.4: Scraping YouTube video data using Dapper

Scraping is not the best way of making information on the Web transparent and remixable as, although the algorithms that work in the background of tools like Dapper are highly evolved, they cannot extract structured data at all times, because the HTML pages are written in a too confusing way for parsing. However *“[…] it is a top-down, unintrusive*

and perhaps the fastest way of turning any web site into a data service. As such, its power and potential exceeds its drawbacks". [Isk072]

3.5.2. Yahoo! Pipes

Although web sites often offer a way of getting data in a sort of structured form like RSS, Atom or XML feeds, remixing them would still require high level programming skills. But since Yahoo! Pipes and similar tools appeared, the task got much easier. [ORe072] describes Yahoo! Pipes as "*a service that generalizes the idea of the mashup, providing a drag and drop editor that allows you to connect internet data sources, process them, and redirect the output*" and [Isk071] states that "*it is the first GUI builder for the biggest database in the world, the Web itself*". Yahoo! Pipes empowers developers with the ability to fetch data not only from feeds but also from certain web services like Yahoo! Search[28], Yahoo! Maps[29]or Flickr. A set of operators, including filtering processes, date-extraction, string operations etc., can be applied to these data sources and developers are able to interconnect the sources with operations by simple drag and drop operations resulting in a pipe shaped structure (see Figure 3.5). With the ability to combine data sources and involve operators in this process "*interesting, non-obvious information is fetched from the web"* [Isk071].

[28] http://search.yahoo.com/
[29] http://maps.yahoo.com/

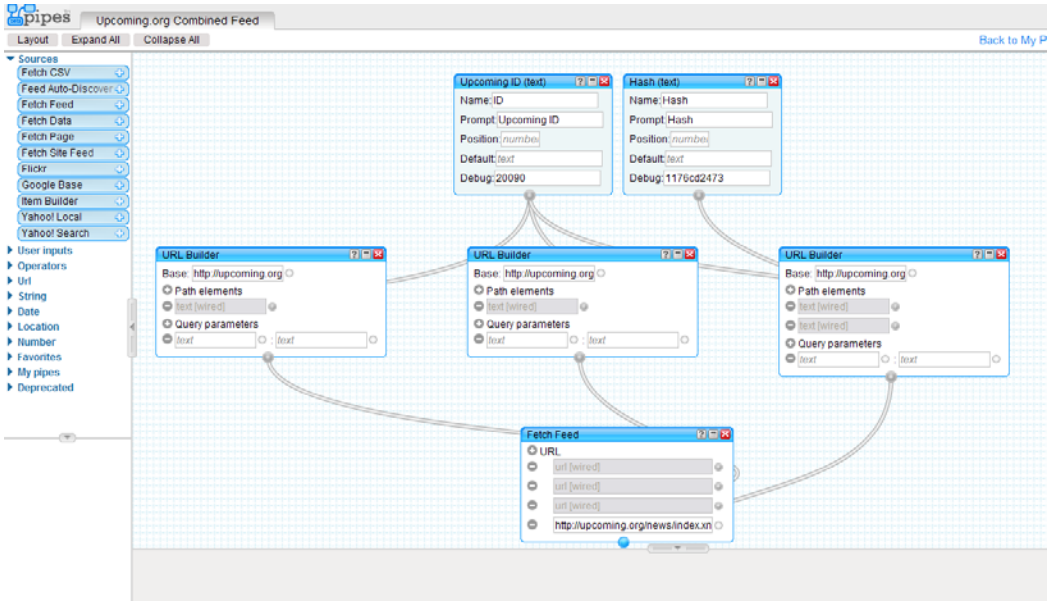


Figure 3.5: The Yahoo! Pipes editor

Building a pipe sounds very simple because it is not necessary to write code. However, it still needs a software building trained brain to stick the pieces together and apply the right operators at the right place and in the correct order to gather the wanted result. Additional data sources will also be required for Yahoo! Pipes to become a successful Mashup development tool. For example, most web services provided are part of Yahoo's own product palette, while other popular services like the ones from Yahoo's biggest challenger Google are omitted. Nevertheless "*Yahoo! Pipes is a first step towards creating a programmable web for everyone*" [ORe072] and should be considered as an interesting concept to see how data mixing could be achieved in the future.

3.5.3. Open Kapow

A mighty tool in building Mashups is the service building platform Open Kapow. It is used if data from web sites are needed for remixing but there is no way of getting this data over a feed or an API. In this case, Open Kapow gives Mashup developers the possibility to create so-called robots (which are actually Web services) "*that automate what a person can do in a browser*" [Ope07]. This means that a robot has to be trained on

the business process of a web site that is needed to be automated, i.e. performing a keyword search in a search engine and retrieve the best matching result. Developers have the possibility to configure the process flow by including loops, conditions, and input handling during the training stage of the robot. Once the robot is trained it can be called to return results as an RSS or Atom feed or the robot can work as a REST API and can be called within a program written in a common Web programming language like PHP. With Open Kapow it is even possible to take functionality snippets out of a web site and move it to another site, thus mixing the GUI's of multiple web sites easily. Such a snippet is called a Web Clip robot. [Ope071]

Robots can even be used within Yahoo! Pipes, but the robot building IDE is still not easy to use and therefore Open Kapow is another tool suitable for persons with software development skills only. But in combination with Yahoo! Pipes it is another step forward to making Mashup development so easy that an average computer user is able to personalize data in the way he wants it.

4. Web 2.0 technology basics

This section discusses fundamental technologies and standards used in the creation of Web 2.0 sites, especially Mashups. The importance of understanding these underlying concepts lies in the lack of strong Mashup building tools and the concluding demand on Mashups assemblers to have web service development skills.

The first subsection, XML, deals with the problem of how data can be structured to be exchanged over the internet in an efficient way. Afterwards, the problem of storing data in a database is treated because there needs to be a place where the information to exchange is kept. Then an introduction to client- and server-side programming and architectural styles in web development is given. Finally, section 4.4 enhances the discussed web engineering mechanisms with a novel paradigm called AJAX.

4.1. XML

4.1.1. XML - A Document Shaping Standard

The World Wide Web can be seen as a huge document storage repository that leads to the urge of setting standards on the physical and logical shape of these documents. An important international standard is the Standard Generalized Markup Language (SGML), a meta-language for the description of a markup language.

Markup languages define "*[…] a set of markup conventions used together for encoding text. A markup language must specify what markup is allowed, what markup is required, how markup is to be distinguished from text, and what the markup means.*" [Spe07]

Following [Kuc98], implementing a full SGML parser is difficult though. Thus the Extensible Markup Language, abbreviated XML, was specified, which is a slim subset of

SGML, having less power than SGML, but providing an easier understanding and implementation. [W3C06] notes that "*XML describes a class of data object called XML documents and partially describes the behavior of computer programs which process them*".

The following list of goals for XML is quoted from [W3C06], the latest W3C recommendation for XML:

- XML should be straightforwardly usable over the internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

4.1.2. Structure of XML Documents

[W3C06] differs between a logical and a physical structure. Physically a document consists of units called entities, which can refer other entities to include them into the document. For the logical structure, [W3C06] divides XML documents into the following parts: declarations, elements, comments, character references, and processing instructions. These parts are indicated by the use of markup.

A typical XML document could look like the following example, which will be used to explain the most important parts of the logical structure, the declaration and the elements:

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE cinema SYSTEM "cinema.dtd">

<cinema>
```

```
  <movie>
    <title lang="english">Shrek</title>
    <time>16:00</time>
    <time>18:00</time>
    <time>20:00</time>
    <rating>PG</rating>
    <description>
      Shrek, the Oger, tries to regain his swamp
    </description>
  </movie>
  <movie>
    <title lang="german">Silentium</title>
    <time>20:00</time>
    <time>22:00</time>
    <rating>R</rating>
    <description>
      Jetzt ist schon wieder was passiert…
    </description>
  </movie>
</cinema>
```

Elements

Elements compose the main part of an XML document by structuring data hierarchically. XML demands a single top-level element, called the "root" or document entity, which is a container for more elements. These sub-elements can have sub-elements themselves again. Elements are indicated by tags, which are either empty or contain content. Content can be simple text, other elements or a combination of both. [Kuc98]

In the above example, cinema is the root of the document and some content is nested between the `<cinema>`- start-tag and the `</cinema>`-end-tag.

Additionally, start-tags or empty tags can contain attributes, specifying special values associated with an element. [Kuc98] E.g. `<title lang="english">` contains the attribute `lang`, indicating, which language is spoken in the movie title.

Declarations

The W3C Recommendation [W3C06] notes that XML documents should, but need not, begin with an XML declaration and consists of two parts, a prolog in the beginning and the document type declaration afterwards.

The prolog contains information including the XML version number used and the character encoding scheme that is used in the document (e.g. UTF-16 Unicode characters).

The document type declaration has references or contains itself rules for a class of documents, which are called markup declarations. These rules build a grammar known as the Document Type Definition (DTD). Therefore the DTD defines exactly the allowed elements with their possible attributes in a document, their default values, and how they must be nested. [Kuc98]

In the given example the line

```
<!DOCTYPE cinema SYSTEM “cinema.dtd”>
```

contains the document type declaration which references a DTD in the file `“cinema.dtd”,` which could contain the following rules:

```
<!ELEMENT cinema (movie*)>
<!ELEMENT movie (title, time+, rating, description?)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST title lang ( english | german | french ) english>
<!ELEMENT time (#PCDATA)>
<!ELEMENT rating (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

This example demonstrates how elements, their nesting rules (allowed sub elements are defined in brackets), and their attributes (with allowed and default values) are defined. For a complete syntactical review, see [W3C06]. With such DTD’s it is possible to check if the structure of a XML document is valid by using a validating parser. More advanced ways in defining the structure of XML documents are XSD (XML Schema Definition) or Relax NG, which are not explained in this thesis.

4.1.3. Usage of XML

The goals defined by the W3C constitute a strong base to use XML and its scheme definition standards as a format for exchanging structured data in distributed environments. The most important reason for this interoperability is probably the fact that XML is totally text-based and Unicode supporting. Therefore it does not matter which 2

systems communicate as both of them can incorporate their own XML parsing mechanisms for processing. A well known example of XML as a data exchanging standard is SOAP[30].

In most cases structured data provides additional information about a resource, so-called metadata.

"*Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information.*" [NIS04]

However, not only can XML serve as a standard for exchanging data, it is also an ideal way to provide metadata to resources in closed systems, like CMSs. [NIS04] defines three types of metadata, whereby XML is a suitable format for all of them:

- **Descriptive metadata:** Describes a resource for purposes such as discovery and identification (e.g. libraries, where the author's name, the title or keywords is important metadata).
- **Structural metadata:** Giving information on how documents consisting of various part are assembled (e.g. the page number where a certain image can be found in a paper of a digital library).
- **Administrative metadata:** Information for managing resources, like user rights, modifying date, etc.

The simplicity of XML as seen in section 4.1.2 lead to the success of XML as the most used standard in describing metadata of resources. And due to the reason that any kind of content on the Web can be seen as a resource, the metadata to such a resource often decides how a resource is mapped to a representation (see [Fie00]). So it is essential for Web developers to have a profound knowledge of XML and it also plays an important role in the development of Mashups, especially in the Mashup developed under this thesis.

[30] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

4.2. Relational Databases

In data driven applications accessing information is requiring an effective mechanism. Often such applications have to deal with several million sets of data and this sets can be interrelated too. These data sets are likely to contain textual data describing resources like metadata in XML does. Keeping all the data in a single or a collection of files seems therefore unsuitable. The lookup of data, especially when pieces of information belonging to different sets should be combined, would require heavy moving of a file pointer and an unacceptable performance would be the consequence. As a consequence, systems relying on data store their information in a relational database system, which is known to work as a fast data managing facility.

Relational Databases are an efficient and convenient way to store access and store data from various relations (or "tables", the synonym for relations in common database management systems). E.F. Codd can be seen as the founder or inventor of the relational data model, which he claims in his paper from 1970 that it solves the problems of redundancy and consistency in data storing [Cod70]:

"*The relational view (or model) […] provides a means of describing data with its natural structure only - that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization on the other.*"

Actually when it is spoken of relation databases the term relational database system would be more appropriate. The database itself is just the physical memory on some kind of storage device. To manage and administrate the data (creating the relations, querying and manipulating the data) a so-called "Database Management System" (DBMS) is needed. Among the most popular database management systems are Oracle, Microsoft's SQL Server, and the open source DBMS MySQL.

4.2.1. Structure of a Relational Database

A database consists of a set of relations (also called tables), which are designed to store data in a 2-dimensional form (a set of information called entity), where the rows are called records or tuples and the columns describe the single fields or attributes of such a record. Therefore each field describes the type of information it holds, like the name of the field and its data type and range of possible values. Each record is then a unique instance of such an attribute tuple.

Identifying records with the whole tuple does not offer a sufficient way of accessing data. In relational databases records in relations are identified using its primary key.

Definition: Primary Key [Cod90]

> *Each table has exactly one primary key. This key is a combination of columns (possibly just one column) such that*
>
> - *The value of the primary key in each row of the pertinent table identifies that row uniquely (i.e., it distinguishes that row from every other row in the table);*
> - *If the primary key is composite and if one of the columns is dropped from the primary key, the first property is no longer guaranteed.*

If a field in a table is part of the primary key of another table it is called a foreign key. Thus if there is the need to refer to a certain record from another table, that table uses the primary key attributes from the table to refer to as foreign key attributes.

4.2.2. Relationships between Tables

It was mentioned above that records from one table can be referenced from another table using keys. These relationships between tables must be separated into 3 different types:

- One-to-one relationships (1:1)
- One-to-many relationships (1:n)
- Many-to-many relationships (m:n)

1:1-relationships

In a 1:1-relationship between tables each record of table A refers to exactly one record in table B and vice versa. For example in a mail-order business an invoice is associated with exactly one customer.

1:n-relationships

In this case a record of table A refers to multiple records in table B but each record in table B is only linked to one record in table A. E.g., is a wheel mounted on one car, but a car has (at least) 4 wheels.

m:n-relationships

This third type of relationship occurs when for each record in table A several records in table B exist and vice versa. For instance actors occur in many different movies, but also a movie stars a lot of actors. These relationships would produce a lot of redundant data, if established in this direct way. A better solution is to split a m:n-relationship into two 1:n-relationships by introducing a third table C. The primary key of table C is then a

composite key of the primary keys of table A and B, so instead of linking table A with table B directly, A refers to C and B refers to C in a 1:n-relationship.

Appendix A explains the process of normalizing a database to define a robust table scheme where data stays consistent over all kinds of relations when data manipulation processes take place.

4.3. Web Engineering

This section explains roughly how web applications work and how their architecture generally looks like. It also describes the communication between the architectural components, both in the traditional way and with the more modern concepts called AJAX.

4.3.1. Client/Server Architecture

A client/server architecture is a common software model for giving multiple users access to the same resources. The server part acts as the resource owner and distributor, while the client requests and uses them (see Figure 4.1).

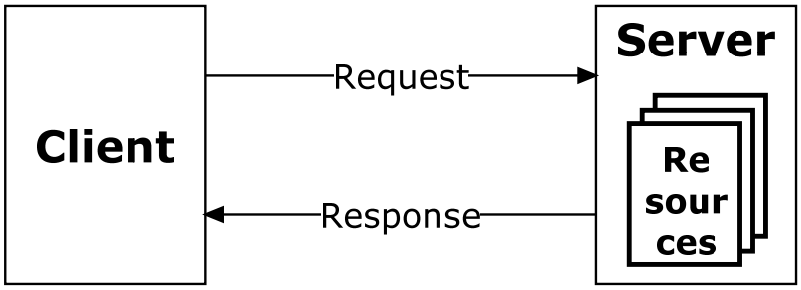


Figure 4.1: Client/Server model

Distributing information and resources is a key purpose of the World Wide Web, thus adapting the client/server architecture to the web is a powerful way of sharing these resources. In most cases, a Web browser takes the part of the client with a special Web server in the role of its counterpart. In this basic model, a Web server has no possibilities to respond dynamically (at runtime) generated content to the client. Only for the client it is possible to dynamically change the used resource by means of client-side scripts like Javascript or by running Java applets in the Java Runtime Environment. To cope with this restriction, the client/server model can be extended to a more sophisticated architecture: the multi-tiered client/server architecture.

4.3.2. Multi-Tiered Client/Server Architecture

Following [Fei00], in the multi-tiered client/server model several more servers can be added to enrich a web application using script or server-side program engines supported by a Web server, which act as an interface between the web server and other servers, like a database or other applications that process and respond data. Using this scheme a client can send data to a web server, where it invokes a server-side script using the engine. This script then sends data to the additional servers and processes the response to generate a response for the client (see Figure 4.2). With this concept, database-backed Web applications can be built where server responses are dynamically built in dependency of the user's request.

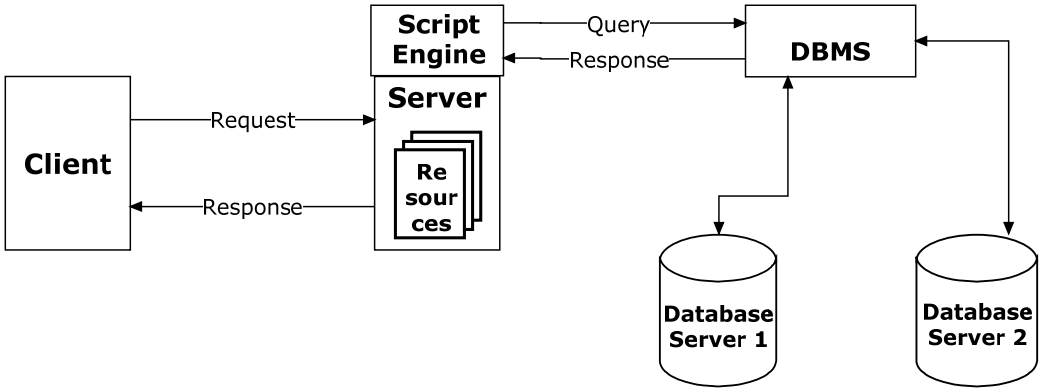


Figure 4.2: Multi-tiered Client/Server architecture

4.3.3. Client-Side Programming

As mentioned in the client/server architecture, the retrieval of resources from a web-server is not limited to static HTML-pages. Also functionality can be added to resources, which can be used on the client-side once the page is loaded. This functionality is mainly realized using the scripting languages Javascript or VBScript (the latter restricted to Internet Explorer as Web Browser) or through precompiled Java Applets.

Client-side scripts or programs are meant to be executed by the browser and not the server. They add value to HTML pages by linking HTML-elements like buttons, text-fields or displaying styles with input dependent computations. Furthermore, client side scripting can make the navigation on a page dynamical and is often used to validate the input of a form. However, according to [The07], client side programming is unfortunately limited in older browsers and it is often difficult to provide reliable scripts to the different environments and systems which are interpreting them.

4.3.4. Server-Side Programming

[Hal00] names the following capabilities of a server-side program:

- Reading data sent by a user and other information contained in an HTTP request
- Generating and formatting results, which are often gathered from databases or remote applications
- Setting the HTTP response parameters and sending the generated document to the client.

Reasons for building web-pages dynamically are [Hal00]:

- Web pages are based on the data a user submitted
- Requested web pages are dependent on frequently changing data
- The web page uses information from corporate databases or other-server side resources.

A widely used server-side programming technology is Java Servlets. Servlets can incorporate all the features mentioned above and illustrate an efficient, powerful, portable, and convenient way to create dynamic web applications:

- Efficiency: Servlets run in a Java Virtual Machine (JVM) on the server, so for each request to a Servlet only a new Java thread is used instead of starting a new heavyweight system process. Thus if the Servlet is called and processed multiple times concurrently, only a single process is used in the server, which holds a lightweight thread for each call.
- Convenience and power: Servlets are written in the Java programming language, which has a vast amount of supporting programming libraries and interfaces, so powerful Servlets can be written easily incorporating functionality that goes far beyond simple database querying and generating documents.
- Portable: Because Java programs run in a virtual machine, a developer does not need to take care of writing different Servlets for different platforms. Only the JVM is different for different platforms and operating systems. Additionally, Servlets are supported by almost all different types of Web servers available.

Enhancements to server-side programming technologies like Java Servlets have been incorporated by other frequently used server-side scripting languages like PHP, ASP or JSP. With the help of these languages, HTML-pages do not have to be generated by a server-side program. Instead server-side computations can be placed directly into prepared HTML-code.

4.3.5. Data Exchange

Communication between Web browser and Web server

The W3C specifies HTTP as a communication protocol to transport requests and responses between a Web browser and a Web server. A client requests and/or manipulates resources on the server by addressing them with the method name (i.e. “GET” or “POST”), indicating what has to be done with the resource, and a Uniform Resource Identifier (URI). To add data to a request the “GET” method gives the possibility of adding query parameters to the URI, while by using the “POST” method an additional data block is provided in the request, which contains data provided by the user by filling a form or similar. “POST” is therefore the preferred and more secure way (because sent data is not visible in the URI) of making annotations to resources and to post messages to bulletin boards, newsgroups etc. The server uses the HTTP protocol to send status codes to the client and to send back the requested resources like HTML pages, graphics or XML data. [Fie99]

Communication between Web Server and Database Server

Section 4.3.2 described the way server side scripting is used to access a local or remote database server from a Web server using a scripting engine. The scripts are responsible for establishing communication with the database management system (DBMS, the software that manages a database) using interfaces like JDBC (Java Database Connectivity, used in Servlets or JSP scripts) or ODBC (Open Database Connectivity).

The scripts use the interface to establish a connection to the database and send queries or update commands over this connection to the DBMS, which responds with the requested data or status codes on the database operation.

4.4. AJAX

Some years ago, comparing desktop with Web applications, it was clear that the richness and the fast responses a desktop application provides cannot be reached on the web. At that time, requests from a browser to a web server always resulted in the reloading of the whole content or the loading of a different web page in the browser, so that the user had to wait until the browser received the response from the server. For the initial use of the Web as a hypertext medium this was sufficient, but not for data manipulation and processing software applications. [Gar05]

However, [Hop07] states that by developing Internet Explorer 5 and Outlook Web Access (the Web frontend for Outlook), Microsoft added new functionality to their browser - XMLHTTP. This made it possible for the browser's Javascript engine to request data from a Web server without the necessity of totally reloading the content of the browser window.

XMLHTTP was shortly afterwards deployed to other browsers as well, but initially it was hardly used. Lately though XMLHTTP was noticed by the web developers community, when Google started to use it in their Web services like Google Mail[31] or Google Maps. E.g. users can zoom and scroll around in Google Maps and receive responses instantly, without page reloads and waiting times (see [Gar05]). People became aware that with XMLHTTP it is possible to drive the richness of Web applications towards the level of a desktop application and at the same time keep the strength of networked information sharing.

In an essay [Gar05], Jesse James Garrett gave this new approach of developing Web application the name AJAX, which is the abbreviation for "Asynchronous JavaScript and

[31] http://mail.google.com/

XML", and defined it, by declaring that it is not a new technology but rather the incorporation of the following listed technologies that play together [Gar05]:

- *Standards-based presentation using XHTML and CSS;*
- *Dynamic display and interaction using the Document Object Model (DOM);*
- *Data interchange and manipulation using XML und XSLT;*
- *Asynchronous data retrieval using XMLHttpRequest;*
- *JavaScript binding everything together.*

Using JavaScript as the binding of the different technologies, an additional layer between the client and the server can be introduced, responsible for rendering the user interface and communicating with the server. This layer is called the AJAX engine and allows the GUI to work independently – asynchronously – from the communication with the server. Thus a user can continue to use the application without having to wait for the server's response (see Figure 4.4). [Gar05]

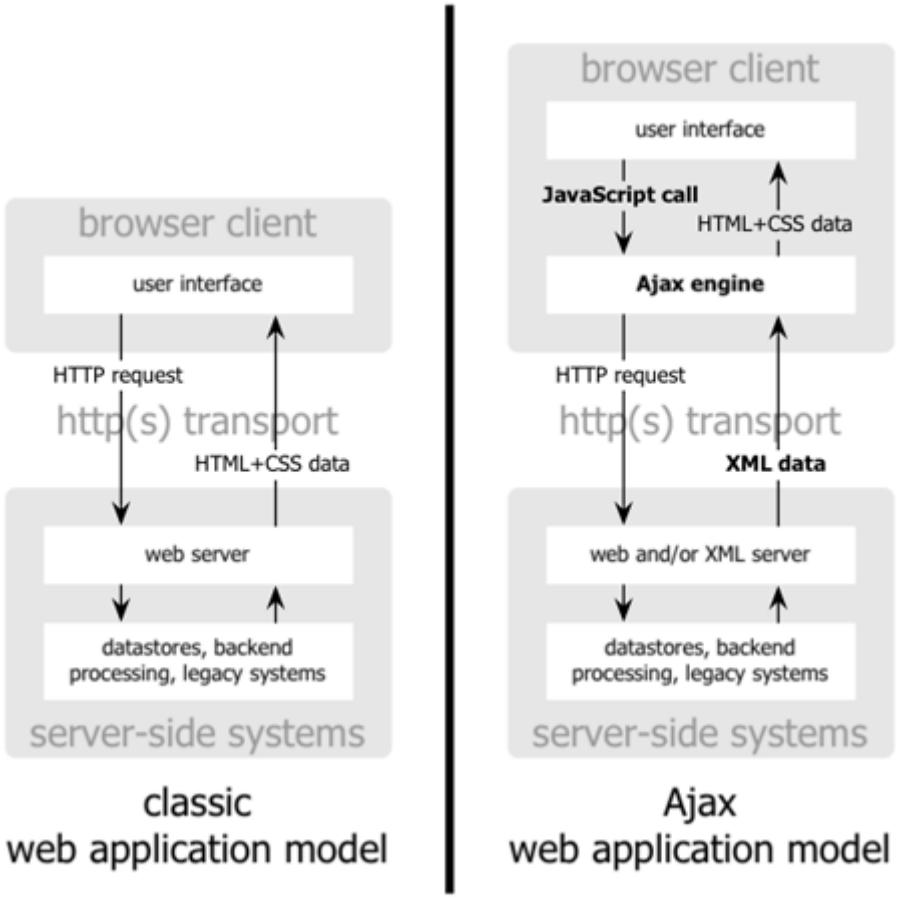


Figure 4.3: The traditional architecture for web applications (left) compared to the AJAX architecture (right). Source: [Gar05]

If a user wants to make a request to the server, now a JavaScript call is sent to the AJAX engine, while in classic Web applications a direct HTTP-Request is sent from the client to the server. The engine relays those requests asynchronously to the server, often using XML, without disturbing the user in his interaction with the application (see Figure 4.3). [Gar05]

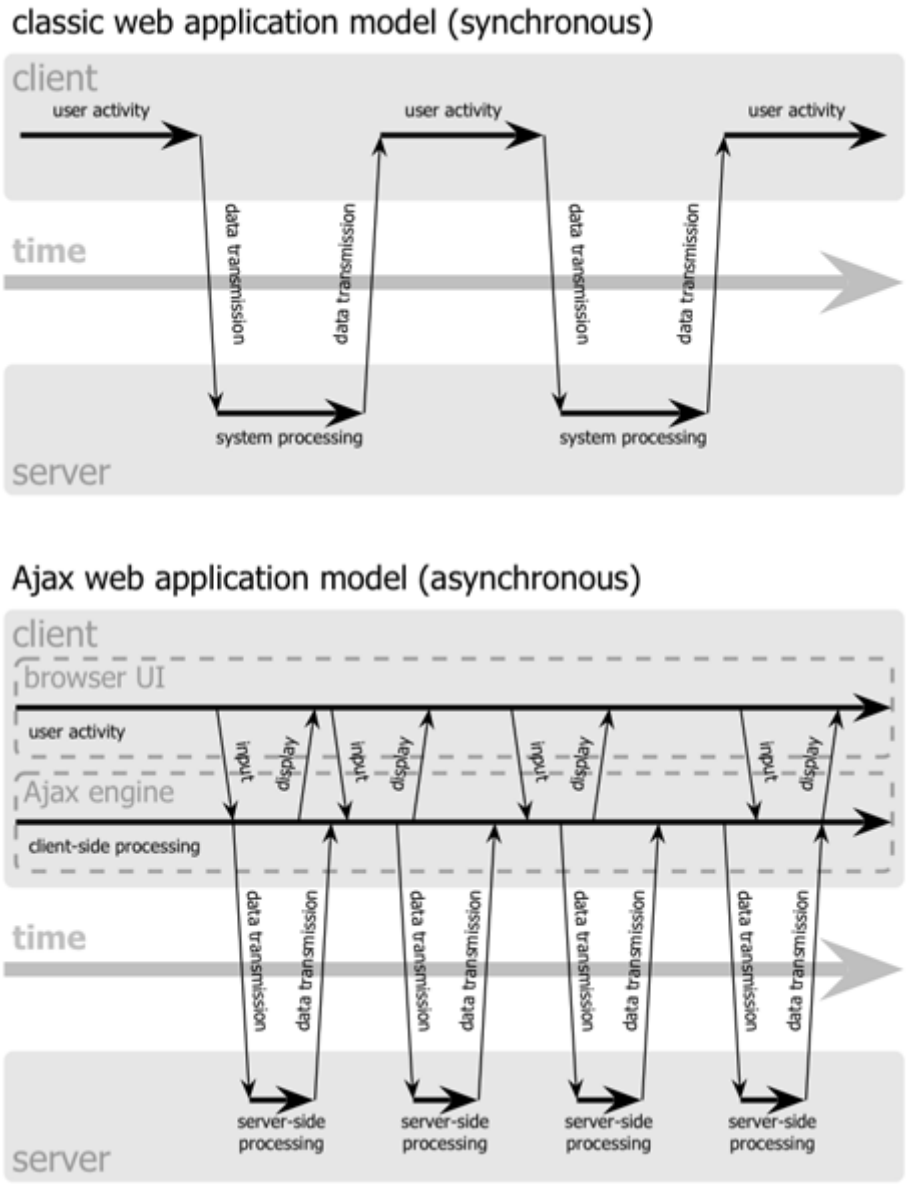


Figure 4.4: The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom). Source: [Gar05]

5. The Journal of Universal Computer Science (J.UCS)

5.1. What is J.UCS?

Following [Cal94], J.UCS was founded in 1994 as a high quality electronic publication covering, as its name already suggests, computer science in general. Electronic means that J.UCS is intended to be read in digital form on the World Wide Web, although it can also be retrieved in printed form (which cannot take advantage of a variety of features that the electronic version offers).

The foundation of the journal is attributed to the fact that its founders believed *"[…] that journals such as J.UCS will replace printed journals fairly rapidly"* [Cal94]. The reasons for their thesis are numerous:

- First there is the comfortable way of searching for a paper. A user can look for a paper by browsing through the different issues in different volumes (issues are published once a month and all issues of a year are merged to a volume). He is able to filter papers by categories, which are defined and used by the ACM, one of the world's most important providers of scientific resources on computer science, search by providing keywords or criteria like the author's name, which can be matched to papers.
- Due to the instant access to information over the internet an electronic journal provides on the one hand worldwide distribution, which gives the reader easy access and on the other hand a shortened span of time between the submission of a paper and the publication.
- The size of an issue is not limited, therefore all submitted papers that have been positively refereed at a particular point will be published in the next issue.

- Digital content goes beyond the simple representation of text and motionless graphics. Video, audio, and 3D models are examples of content that can be used except from the standard media types used by traditional printed publications.

[Cal94]

The right for calling it “high quality” is based on a large editorial board consisting of over 160 editors originating from all over the world, with a majority of computer scientists, and an intense refereeing process, at which at least 3 referees, called editors, judge the quality of a submitted paper.

It is notable that J.UCS joined the “Open Access Initiative for scientific literature” with January 2007, which means that the journal has been free of charge from that point in time on including already published issues.

5.2. J.UCS System Environment

Hyperwave IS/6 is a knowledge management system based on a powerful server that allows organizations to make information accessible to a large number of people and effectively manage this information. It stores documents in a database and maintains these documents. Maintaining documents includes updating the position of a document to its new place when it is moved, or removing a document from the database when the document is deleted. [Hyp061]

When speaking of documents, the IS/6 definition goes beyond simple textual content. In IS/6 a document can be any kind of digital content. The range starts from basic types like text or images and actually has no end, because new document types can be defined by application developers to support their needs.

The appearance of the user interface is determined by special files called *templates*. These files contain JavaScript, HTML, and an interface programming language called PLACE. IS/6 application developers can alter these templates to suit their individual needs. [Hyp06]

Because J.UCS is based on a Hyperwave IS/6 Information Server with an Oracle Database in the background, storing as well as modifying and representing documents is entirely done by IS/6. This shows that the basic functionality of an electronic journal can easily be incorporated.

5.3. Enhancing Digital Libraries

Digital libraries are electronic publishing systems, which contain multiple digital journals and assist users in retrieving information stored in one of their journals.

This section describes what kind of features a digital library can provide apart from basic functionality like providing a browsing structure, a search engine which returns a list of articles matching the query, a view mechanism for the articles, and an article categorization scheme. Because a digital journal can also be seen as a digital library containing only one journal all of the above functions can also be applied to a single journal.

[Kro02] proposes that a system should give the user the ability to bookmark articles. These bookmarks could then be shared with others and reading lists to certain topics can be created. Apart from digital journals, bookmarking services working like that can already be found on World Wide Web (e.g. del.icio.us[32]). Surprisingly, these services are still not provided by many digital journals or libraries (ACM Digital Library[33] is one of the few examples where bookmarking can be done). Other functionality suggested by [Kro02], which should be highlighted are that single words or technical terms should be interlinked to a dictionary or thesaurus, so a user can get the meaning of it.

Functions for a digital library, suggested by [Dre04], should also be obtained by a single digital journal:

- Journals, as well as libraries, should adapt the interface to the user and "*[…] implement personalization, i.e. adaption of the content, personalized listings of content, recommendations to content of interest to the user, etc*".

[32] http://del.icio.us/
[33] http:// http://portal.acm.org/dl.cfm

- Alerting services could notify the user of newly published articles or the user could only be notified about newly arrived articles from a certain author or of a certain field of research.

A concept called "Active Annotations", suggested by [Hei00], enrich digital journals by giving users the possibility to add comments or additions, ask questions or take part in a discussion about an article in a comfortable and powerful way where even multimedia content can be used. *"The publishing process therefore does not end after a successful submission of an article. It continues as long as questions are asked or comments are made."* [Dre04]

The "Active Annotations" concept is part of the "Active Documents" idea by [Hei00], which gives a discussion about an article also the mechanism of answering questions of users automatically by measuring similarities between the question and older and already answered questions. Questions with a high similarity can then be presented to the user who then decides whether there is already an answer to his question.

A great feature digital journals could possibly offer, which is already incorporated to a certain level in J.UCS, is the concept of "Links into the Future". By quoting [Dre04] the idea can easily be explained:

"*An article A cites an article B. B was published somewhere in the past using an arbitrary system. Currently, most systems create links from article A to article B. Since links in ordinary web-technology are unidirectional, no reader of article B will ever notice that the author of article A cited this article."*

With powerful Information Servers or Document Management Servers like Hyperwave references from article B to article A, in the example above, could easily be determined at run-time due to the storing of links in such servers. By populating so called *White Lists* of journals, which are compatible to the one viewed, and by using XML-based remote service invocation, articles in alien journals could be referenced too.

Interoperability of journals is even more supported by providing articles in various formats (as is done in J.UCS), like PostScript (PS), Portable Document Format (PDF) or even HTML.

A big issue in making digital journals more comfortable is the problem of increasing the quality of a search. Most systems provide search mechanisms, which result in a linear list of articles ordered by a scheme like significance of the article in relation to the query parameters or by user ratings. [Dre04] suggest a more interactive mechanism between the searcher and what is being searched and therefore come up with the idea of representing search results in a more structured way and of keeping close to the proposal of "concept maps" by [Don01] where search results are organized by personal factors of the user instead of applying the usual methods. [Foo01] discusses the visualization of different usage processes in a digital library like visualizing a search or navigation session or the structure of a search. Various visualizations should be provided whereby a user can profit differently from each of them

Following [Kro02] digital libraries should have mechanisms that give the reader the answer to the questions "*Which other resources should be read to understand the article?*" and "*Who are the real experts on this topic?*".

5.4. Strengths and Weaknesses of J.UCS

By having a look at the table in [Foo01], listing which features the various digital journals offer, it gets clear that J.UCS was indeed already among the most mentioned and thus most innovative journals already back in the year 2001. With the introduction of functionality like "Links into the future" it asserts itself. Other mechanisms worth mentioning because they are not self-evident are:

- Offering of papers in different representational styles (PDF, PS, HTML)
- Extended search mechanisms (browsing by author, topic, publication)
- Search for similar documents
- Providing of BibTex citations
- An annotation feature for discussion and reviewing

Nevertheless J.UCS is also lacking features other journals already offer (see [Foo01]):

- Incorporating other publishers in the search
- Advanced display feature, e.g. "*to produce a view of the relationships between articles (e.g. 3D)*" [Foo01]
- Embedding of multimedia content (audio, video, 3D models)
- Support of supplemental data or program code
- Personalization of reading habits and search preferences

5.5. Motives for Creating Mashups for J.UCS

"As J.UCS develops it is clear that additional demands will be placed on it. … including interfaces with other applications, dynamically updated glossaries or bibliographies and the like." [Cal94]

This statement contains the description of one of the key motivations of J.UCS: the aim of always staying on top of the art, which in that case is the information representation and visualization. The journal not only tries to be equal to the competition but tries to exceed the achievements of the others.

In section 5.3 a lot of suggestions to enhance J.UCS can be found and it is often a matter of foreign data sources to incorporate them. The possibility of fetching these various J.UCS independent data sources and databases by the introduced Mashup technology supports J.UCS to build these new features and give the journal the required boost to realize the above-mentioned aim.

"A Mashup for a digital journal can serve both as a mean of harnessing the social computing on the WWW and as an administrative support tool (to support rapid expansion). Mashups can be applied to provide rich collaborative media and content management facilities to support journal administration, structured academic publication, and to manage digital libraries of scholarly content." [Kul07]

It is assumed that the attributes mentioned by [Kul07] are interrelated in a close way and the interaction of a user with the Mashup yields information, which has imminent

influence on the administration of the journal, e.g. when the user edits contact information about a person using the Mashup, the J.UCS administration can be able to contact an author who moved somewhere else without doing research for his actual location.

Section 5.3 proposes the visualization of search results to enhance the quality of a search. By using Mashups it is possible to visualize data of J.UCS in an easy way by using the interface to public services, which can display information representative and nice, and as a consequence to that visualization new conclusions regarding the existing data can be drawn. For example, the concentration of research on a certain subject in a particular geographical area can lead to the decision that a planned conference or congress on this topic shall be organized somewhere in this area. The Mashup with the geographical visualization will give the necessary hint in that decision process.

As a consequence there will be a tool that matches a scientific topic (i.e. published papers in an ACM category) to geographical regions where most of the research on it is done. Of course this would be a quantitative tool, whereas no conclusions on the quality of research in a region can be drawn using this mechanism. Further on, a Mashup could assist in generating statistics on the J.UCS data.

It can also be thought of using Mashups (and therefore external data sources) to build “concept maps” or any other information structuring devices to organize search results in a digital journal, as [Don01] think of.

The variety of services publically available are not only for visualizing journal data, but also enhance or round off data. To give an example, it is possible to link the J.UCS database to information provided by other electronic journals or scientific encyclopedias available. The retrieval process of literature in scientific work would be made much easier this way.

Giving the user the possibility to interfere with the information provided by J.UCS raises the question of the necessity of creating Mashups at all. Undoubtedly, some features which could be thought of (and with the daily growing number of available data sources the number is uncountable) could cause more problems than they solve, but due to the mentioned aims of J.UCS, each idea enriching the human-computer-interface on

scientific publishing and research with electronic journals should be worth trying and so it is done as described in the following sections.

6. J.UCS Mashup Prototype

The previous chapters described the background for building a Mashup for J.UCS. The aim of building this Mashup prototype is to show how a digital journal can profit from harnessing external data sources and social computing in a Web 2.0 manner and as already declared to help enhancing the functional range of J.UCS.

6.1. Situation Analysis

J.UCS is not just a simple digital journal. It wants to be innovative and tries to avoid stagnation in its presentational and administrational functions (see section 5). An inspection of the main feature of J.UCS – offering readers high quality scientific publications – shows, that the journal contains much more than just a collection of papers to view. In fact each paper is associated with a lot of additional information (metadata) stored in XML files accompanying the papers. This metadata includes attributes about the position in the J.UCS hierarchy, topic classification, and author details making it possible for users to specify search parameters for a paper lookup (see Figure 6.1).

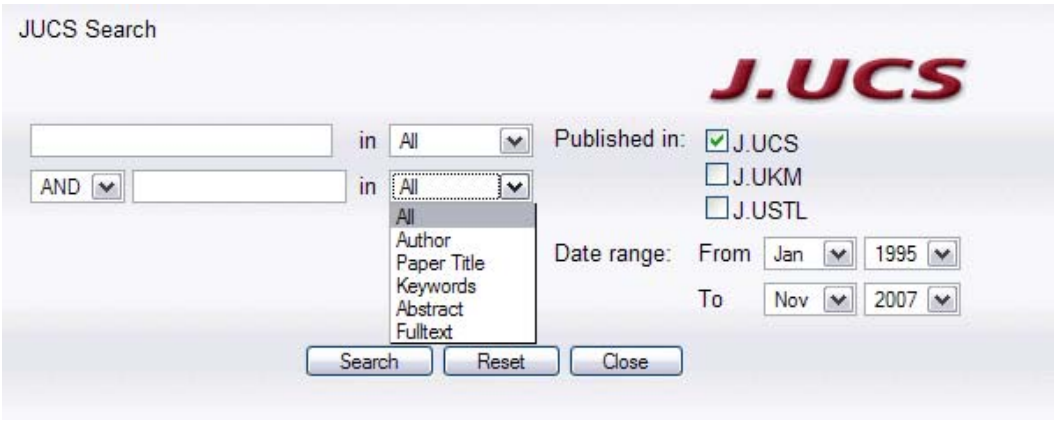


Figure 6.1: The J.UCS search form

A sample Metadata file about a paper:

```
<?xml version="1.0" encoding="UTF-8"?>
<article xmlns="http://www.ujseries.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.ujseries.org
     http://www.ujseries.org/article-20050623.xsd"
  id="jucs_9_3/compiler_optimization_meets_compiler">

  <title lang="en" runningHead="J.UCS Special Issue on ...">
     J.UCS Special Issue on Compiler Optimization meets
     Compiler   Verification (COCV 2002)
  </title>
  <abstract lang="en"></abstract>
  <author id="1" type="corresponding">
    <firstname>Jens</firstname>
    <middlename></middlename>
    <lastname>Knoop</lastname>
    <email>knoop@complang.tuwien.ac.at</email>
    <phone></phone>
    <city>Vienna</city>
    <zip></zip>
    <country>Austria</country>
    <institution>
      <name>Vienna University of Technology</name>
      <url></url>
    </institution>
  </author>
  <author id="2">
    <firstname>Wolf</firstname>
    <middlename></middlename>
```

```
    <lastname>Zimmermann</lastname>
    <email>zimmer@informatik.uni-halle.de</email>
    <phone></phone>
    <city></city>
    <zip></zip>
    <country>Germany</country>
    <institution>
      <name>Martin-Luther University Halle-Wittenberg</name>
      <url></url>
    </institution>
  </author>
  <submissionDate>2003-04-23</submissionDate>
  <acceptanceDate>2003-04-29</acceptanceDate>
  <publicationInfo journal="jucs"
                   issue="3"
                   issueType="special"
                   issueAccess="restricted"
                   volume="9"
                   date="2003-03-28"
                   managingEditorColumn="yes"/>
  <pageInfo from="189" to="190" number="2"/>
</article>
```

Comparing the information J.UCS presents the users with the types of metadata that are available through the XML-files, it occurs that some fields, like the address fields, do not have a significant computing functionality (they can't be selected as search parameters, etc.). They just provide some extra information for the user. However, by thinking about what other benefits could be gained from fields about i.e., city and country, and by being aware of publically available web service API's (e.g. Google Maps) as well as knowledge of emerging new paradigms like Mashups, it seems possible that such a Mashup, which displays authors and editors of J.UCS in a map, could gain additional administrative power and also enhance the user experience of scientifically interested readers. Such administration power could be the derived from coupling the visualization of J.UCS data in a map with statistical information won by the data currently displayed, like the distribution of papers over the years.

Possible benefits from such an application for administrative purposes involve getting a geographic overview of which regions in the world have a lack of editors on which topics. Getting this knowledge could activate the journal's responsible staff to recruit extra editors for this regions if necessary for whatever purpose possible. Other information resulting from the Mashup would be which topics gained importance recently

and which are disappearing from current research. Further on it helps profiling editors and authors.

6.1.1. Basic Requirements

User requirements

A system should be built that is able to display authors of papers and editors of J.UCS in a geographic map with the following features:

- Users should have the choice, if they either want to search for authors or for editors.
    - Authors to display are selected by the ACM categories of the papers written by them and/or through the selection of whole volumes, issues or single papers.
    - Editors are selected by the ACM categories they are editing.
- The displayed persons are represented by placing an icon into the map.
- Authors or editors located in the same city should share the same icon. The size of the icon indicates the number of persons at the same spot though.
- By selecting an icon displayed at a certain location on the map, information about the authors or editors at this location should be provided
    - Authors: Displayed information should include the number of papers of an author matching the selected search criteria, a list of those publications along with information on the position in the J.UCS hierarchy (volume, issue, pages, ACM categories) and personal information on the author like name, professional address, email, and website.
    - Editors: Information about editors should be name, professional address, email, website, and the ACM categories he is editing for J.UCS.
- Results should not only be displayed on the map, also statistical information should be derived from the search results:

  - Information on the distribution of the authors and editors over the countries should be given.
  - Statistic information on the time of publication of the requested papers should be created. It should count the number of papers published in the same year and the number of papers for each issue in that year.
  - A quantitative comparison of the main ACM categories should be made on the requested papers. Thus for each main ACM category the according papers from the search result should be counted.
- The system should give the user the possibility to filter displayed locations by their country. This means that authors who are not from the selected country will be hidden in the map.

System Requirements

- The performance of the system should be as fast as possible without restricting the full implementation of other requirements.
- Data should be stored in a relational way so references to one and the same record, instead of having duplicate records, are preferred where they are possible. This way information stays consistent if edited. At the same time, data storage and retrieval should be efficient in this relational data storage facility.
- Although an application accessed over a web browser is operating system-independently, it should be compatible with the currently most popular web browsers like Internet Explorer version 6 and higher or Mozilla Firefox version 1.5 and higher.
- The system should be built in modules, which are reusable in other applications.
- A programming language independent way of communication between modules has to be realized and module APIs should be as simple as possible.

6.1.2. Problems Recognized during the Analysis

A deeper analysis shows that a high percentage of authors and editors do not provide detailed information about their address (see Table 6.1). They often do not offer information about the city or country they work in, which blocks the insertion of their location into a map. As a consequence, the Mashup could end up in meaningless results as a lot of authors or editors are spared out and so conclusions cannot be drawn.

Total Number of Author addresses:	**2014**	
Number of addresses where City Information is given:	846	42%
Number of addresses where City Information is missing:	1168	58%

Table 6.1: Comparison of authors who provided a city name and those who did not

To deal with this problem the Mashup has to be further developed to support a mechanism, which tries to automatically determine a person's missing location data from the data he has provided. However, detecting missing data automatically could lead to errors or could also end up without a result if the quality and quantity of the provided information was too low. Therefore the system that has to be constructed should also have the power to embrace the user community's knowledge to add and edit the information about persons.

6.1.3. Additional User Requirements

Due to the faced problems the following additional user requirements have to be realized:

- The system should try to add missing data of a location as good as possible.
- Locations of authors and editors on the map should be separated into confirmed and unconfirmed locations and represented by different icons for different types.
- Initially all locations are unconfirmed. By social computing or, in other words, by using the knowledge of the user community it should be possible to confirm locations of authors or editors.

- If users find authors whose addresses have been geocoded to the wrong location and these locations are unconfirmed, there shall be a way to move this author to the right position in the map.
- It should also be possible for users to edit and complete and therefore correct information about authors or editors.
- Additional statistical information on a search:
    - The numbers of confirmed, unconfirmed, and not geocodeable locations should be displayed.

6.2. J.UCS Data Migration and Cleaning

J.UCS as an information system is realized and made public using the powers of a Hyperwave Information Server (IS/6). This server stores papers in a publishing hierarchy ordered by volume and issue numbers (though links from i.e. ACM categories stored in IS/6 to papers are present). As mentioned above, each paper has an XML-file containing metadata about it. However, the system must be capable of determining relationships of one paper to others with similar attributes, for which the browsing of XML files is insufficient, and to achieve the system requirement of keeping data relational, the metadata from theses files needs to be stored differently. Also the system and the users would benefit if prior to the use of data in the Mashup a cleaning process took place in which spelling mistakes would be rectified from the present data and unambiguous names or abbreviations would be used for cities and countries (e.g. instead of having occurrence of U.S., USA, and United States as country name in the database only one of these three should exist or as another example either Munich or München should represent the Bavarian capital). As a consequence, the metadata has to be migrated from IS/6 to a relational database to store the data the way it is required in the Mashup. This is done in two steps[34]:

[34] Depending on the used DMBS a third step has to be taken in between: If the DBMS stores data in another encoding than UTF-8, the result of step 1 has to be converted to ensure the right encoding.

1. Exportation of the metadata to a single XML file using the Hyperwave API.
2. After the exportation a Java program parses the metadata and inserts it into the relational database. It also makes sure that none of the records which have already been inserted are imported twice.

It is then the task of a legitimated person to run a batch which executes above routine to keep the relational database up to date when new papers are published in J.UCS. In future the migration batch process should be triggered automatically when new papers are published in J.UCS

While inserting in the database cleaning of data is done by utilizing the GeoWorldMap database to verify the city and country names. City and country names which do not occur in the GeoWorldMap repository are identified in this process and are later on processed manually to associate them with the intended cities and countries. [Kha08]

6.3. Architectural Design

6.3.1. A Middleweight-Client-Server Multi-Tiered Web Application

To create a Mashup with the above-mentioned requirements the chosen architectural style is a multi-tiered web application adapted for Mashups. It consists of the following parts:

- Presentation tier: A Javascript capable Web browser takes charge of the presentation of the Mashup and the interaction with users.
- Application tier: This layer keeps hold of all program logic, which does data processing, calculations, and data structure management that is necessary for the Mashup. It also directs and processes data between the presentation and the data tier or the third party tier.
- Data tier: A relational database forms the data tier, responsible for data storing. The application tier requests and manipulates the stored data.
- Third Party tier: Newly introduced to the multi-tiered web application architecture, this layer is a compound of all the third party APIs, which are

addressed by the Mashup to incorporate their functions. Communication between external sources and the application runs over the client's Javascript capabilities.

Apart from introducing the additional third party tier to the multi-tiered architecture another modification caused by this tier becomes evident. Usually the client handles as few calculations and data processing steps as possible and the majority should be done on the server-side, also known as "Thin-Client"-systems. Because third party services are accessed using Javascript on the client, it is more efficient to process the data requested from these services on the client than sending it to the server. So a lot of the business logic returns from the server to the client. Instead of calling it a "Thin-Client-Fat-Server"-system, the term "Middleweight-Client-Server"-system would be more appropriate. Thus a separation of the application tier into a client application layer and a server application tier has to be done. Figure 6.2 shows a schematic of the proposed Mashup architecture.

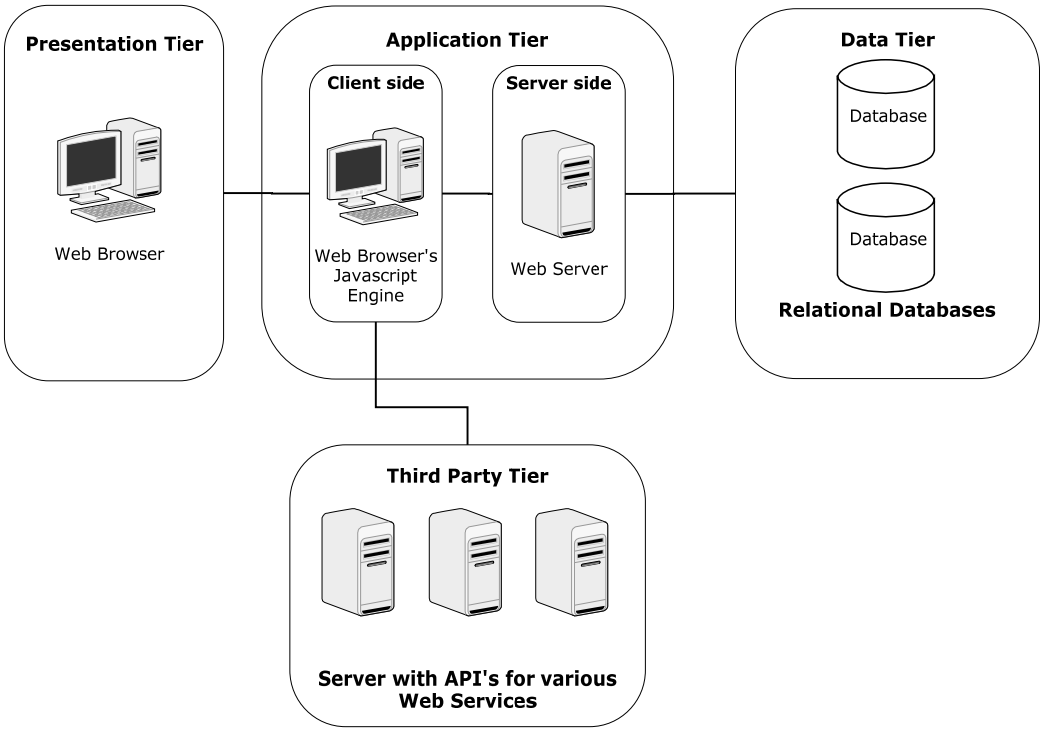


Figure 6.2: Mashup architecture

6.3.2. System Component Architecture

The following software components provide the necessary environment to implement the requested Mashup.

- Javascript capable and Google Maps supporting Web browser: This is the only installation prerequisite a user must fulfill to access the Mashup and is therefore used as the client. Recommended Browsers are:
    - Internet Explorer 6.0 or higher
    - Firefox 1.0 or higher
- Apache Tomcat Web Server 6.0 or higher: A web server hosting the Mashup's web site and client side scripts as well as Java Servlets for server side processing and other responsibilities of the server side application tier. The server uses a JDBC driver to connect and communicate with the database.
- Microsoft SQL Server 2000: A DBMS for the relational database of the Mashup. SQL Server implements the SQL standard and is therefore easily accessed from Java Servlets. Because it is a Microsoft product the system has to use a Windows version of NT 4.0 or higher as an operating system.
- Google Maps API Server: Is a server operated by Google and therefore part of the third party tier. The API is integrated into the Mashup using a Javascript file located on a Google hosted server. Once this Javascript file is loaded by the Web Browser it can be used to communicate with the Google Maps API Server and to integrate this server's data into the Mashup.

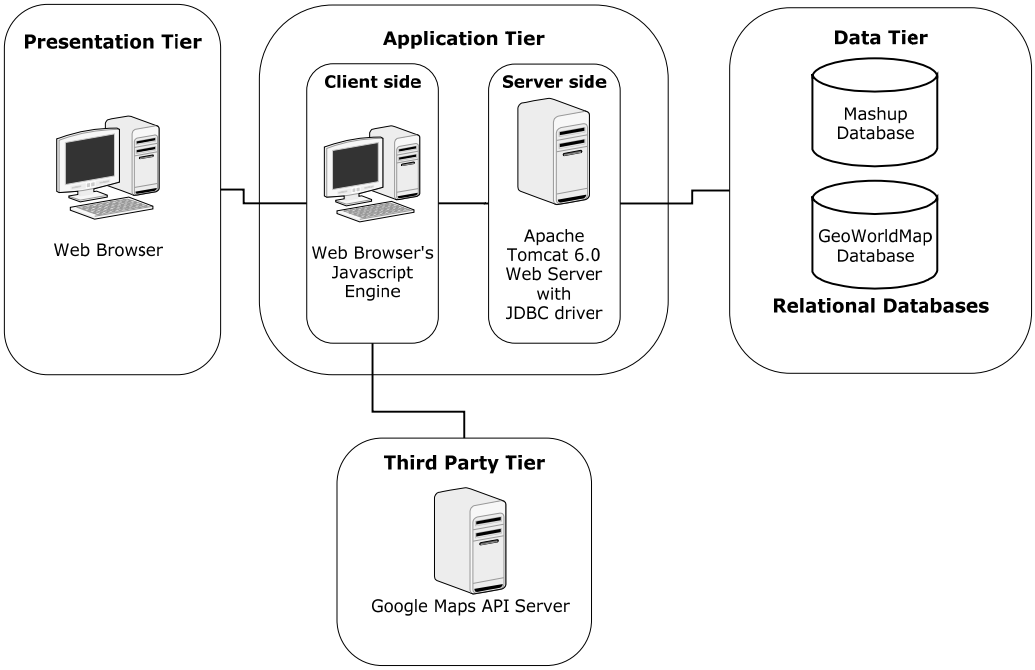


Figure 6.3: System component architecture for the J.UCS Mashup

6.3.3. Functional architecture

This section discusses the functional architecture of the Mashup. The architecture is divided into modules where each is capable of handling a specific task in the usage process of the Mashup and is composed of the necessary data structures, functions, and interfaces to fulfill this task successfully. A module uses its interfaces to communicate with other modules and retrieve data from them. Considering the requirements and the system components identified in the previous sections, the modules and their interactions shown in Figure 6.4 have been derived and are explained below.

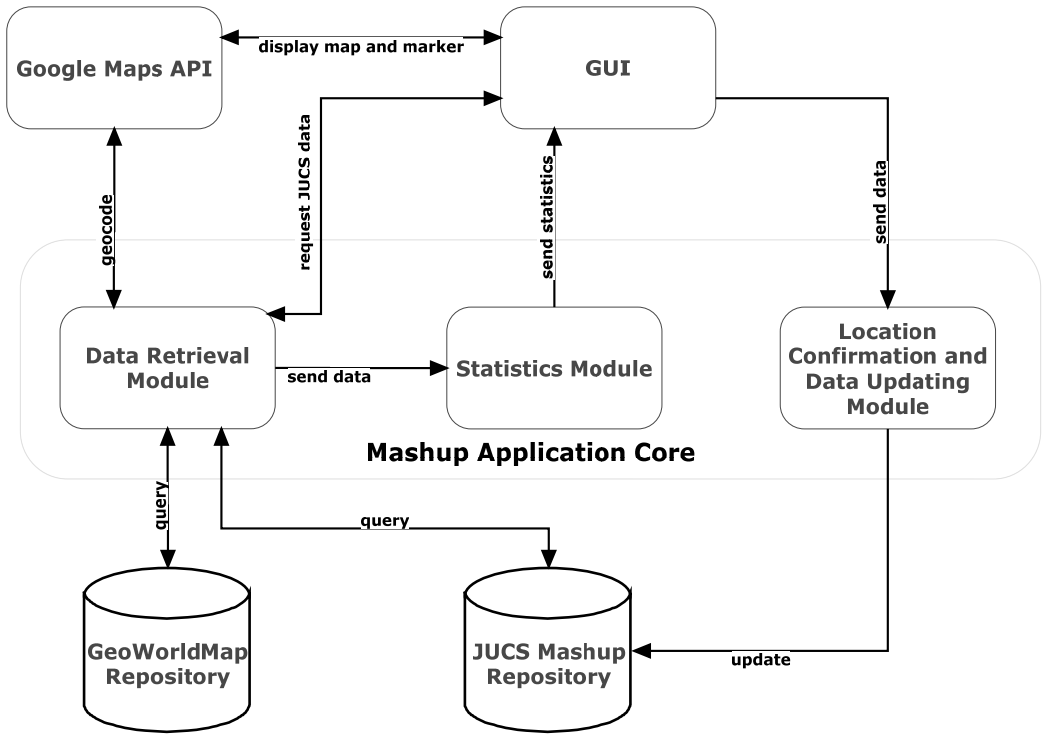


Figure 6.4: Functional architecture of the Mashup

Google Maps API

Because this module is part of the third party tier and like the name API already indicates, it provides an interface to the external Google Maps web service (Figure 6.5 shows the web service directly addressed via a web browser showing a map on the right side and search and navigation facilities in the top left part of the service). The interface provides means to include a map into a custom built web site and add value to it by placing markers on the map using Latitude and Longitude coordinates and attach information windows to markers or by drawing polygons into a map. The map works in the same way as in the normal Google Maps web service, thus loading of new map parts and zooming in and out is done over an AJAX engine, whereby the Mashup developer does not need to take care of this as this is automatically enabled once a map is loaded into the web site using the API. With AJAX the navigation process in the map is quite fluent due to the fact that the map is split into small images attached to each other, so while the map is

displayed the surrounding pieces of the current view are loaded in the background and are already available for display when navigation processes take place. In contrast to the standard Google Maps web service the API provides means to customize the navigation up to a certain level implemented in the API (e.g. zooming can be done by using a mouse wheel, double-clicking the map or by buttons in the map). Additionally, it is possible to geocode street addresses to coordinates using the API, which is an important feature used in the Mashup.

Figure 6.5: Google Maps web service

Data Retrieval Module

Searches triggered by a user from the GUI are sent to the data retrieval module. The module's task then consists of querying the J.UCS Mashup database to get all known data matching the search parameters. The database response contains all data on authors or editors that is needed to fulfill the specified requirements, and also additional information like the coordinates of the person's location if they are known yet. In case that the coordinates of a location are undetermined the module's geocoding algorithm (see section 7.2.1) tries to calculate the coordinates of the person's location (locations are only geocoded to their city though, street address level is omitted because it is clearer if people

from the same city are joined at the same map spot instead of having 2 different coordinate pairs hardly distinguishable on a high scaled map due to their closeness).

The retrieved and determined data is stored in data structures for authors, editors, and papers on the client which are used by the GUI to insert them as markers in the map.

Statistics Module

Information gathered by the data retrieval module is stored in data structures for authors, editors, and papers. This information also contains relevant pieces to build the requested statistics. Thus these pieces are sent to the statistics module via its interface. Moreover, the interface is also used by the GUI to get the statistical data out of the module and present it to the user once a search triggered by him is fully processed. Inside, the module takes charge of calculating significant values and, as a consequence, building the data structures for the statistics.

Location Confirmation and Updating Module

Giving the user the possibility to engage himself in confirming locations as accurate and editing details of authors or editors, a logic for taking charge of the necessary data updates is obligatory. For this reason, the module which handles the updates has to make sure that while updating location and institution data of authors or editors no information on other persons' data is changed. For example, if multiple authors are related to the same location but the address of one of them is changed due to a new engagement at another institution, only this author is assigned a new location, while the other authors are still related to the old location. In other words, the module has to determine when to update a record in the J.UCS Mashup database and when to insert a new one and relate other records to it.

Graphical User Interface (GUI)

The GUI is the layer responsible for the interaction with the user as well as for presenting content and interaction aids to the user in a clear and appealing way. In the J.UCS Mashup the GUI consists of the following three main parts:

1. Search interface: This interface lets the user decide, whether he wants to retrieve information on authors or editors, which type of locations should be displayed (known, unknown or both), and to which topics and/or publications the results belong. The last mentioned search parameter has to be distinguished between search for editors and search for authors:
    a. Editors: In the case that information on editors is requested, the user can constrain the result to specific topics classified by ACM categories, whereas constraining to certain issues is disabled, as editors do not belong to single issues, but rather edit for a long time.
    b. Authors: If information on authors is required, topic and publication constraints can be used alone or in combination to narrowly determine data. As papers written by authors are filed to ACM categories, papers matching the specified topic search are retrieved. The time range in which a search should be performed can be given using the publication constraining where whole volumes, multiple issues from a volume or even single papers can be included in the search.

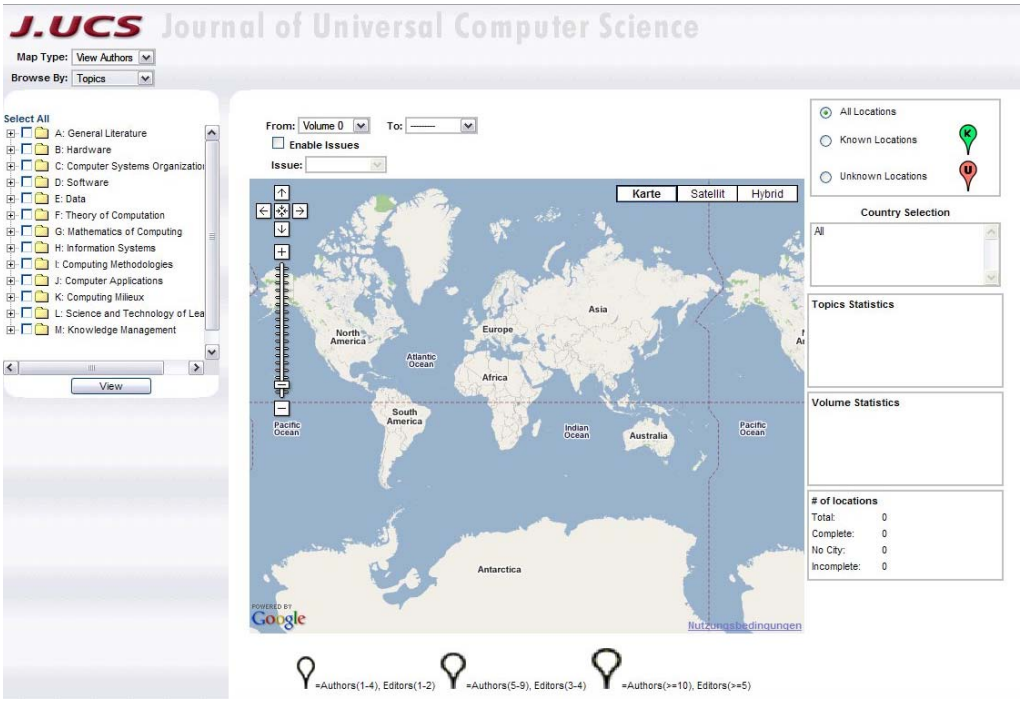


Figure 6.6: Graphical user interface of the J.UCS Mashup

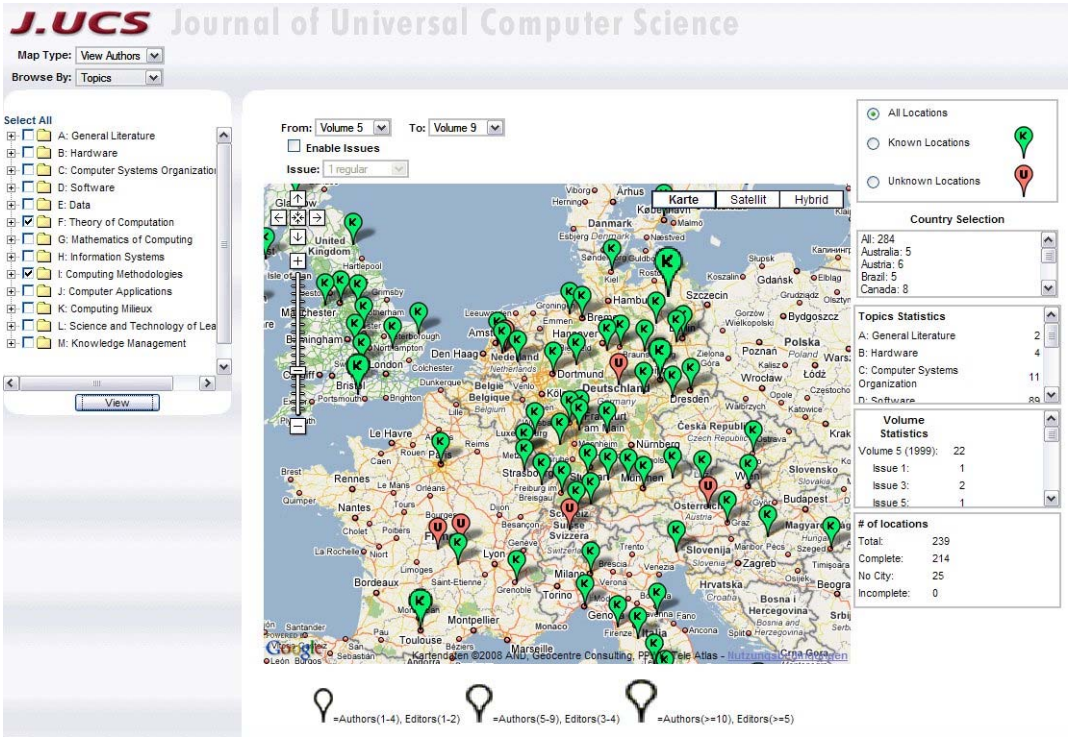


Figure 6.7: Finished search with mapped locations and statistical information

2. Map: A map retrieved using the Google Maps API. Search results are displayed in it as markers at an author's or an editor's location. By clicking on such a marker an information window opens, providing the user with information on the author and the papers he has written according to the search parameters or information on the editor, along with the ACM categories he is editing. Additionally, such an information window gives users the possibility to mark an unknown location as known or to access an editing interface where he can correct or complete a person's data. Locations displayed as unknown can also been dragged around the map to the correct location as another way of enhancing incomplete data.

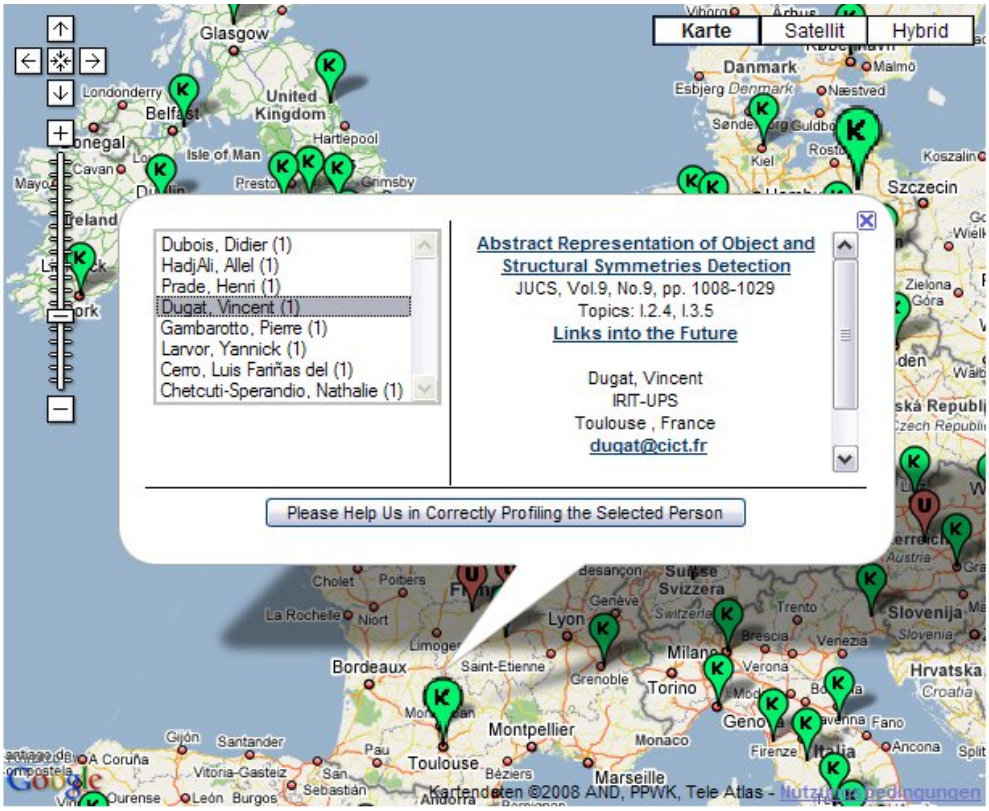


Figure 6.8: Information window linked to a displayed location

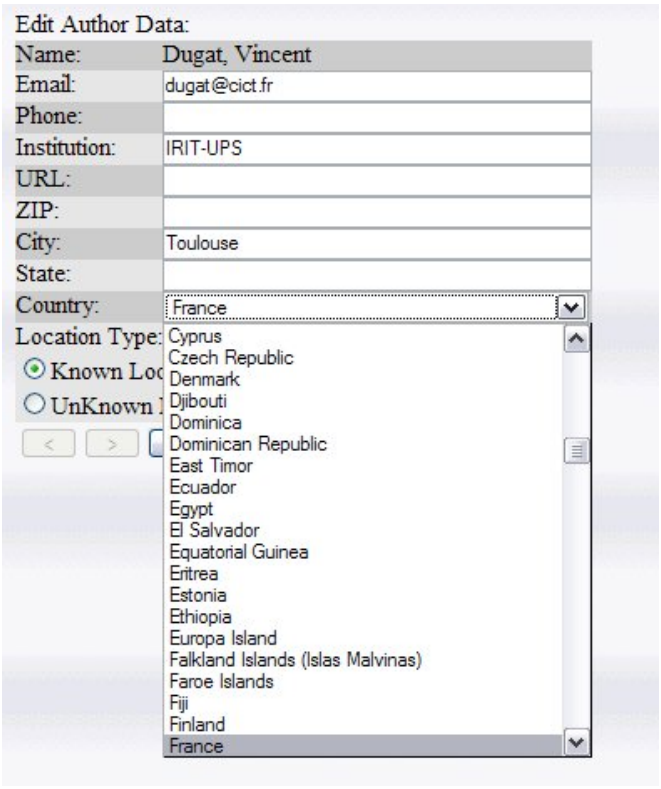


Figure 6.9: Author data editing mask of the J.UCS Mashup

3. Statistics: Along with the search result displayed in the map, statistical information is derived from this result and displayed in the Mashup as specified in the requirements. Furthermore, the user has the possibility to filter search results in the map by using the statistics on the distribution of persons by country. A simple selecting of the desired country does the trick. This means that people located in other countries than the one selected will be hidden from the map.

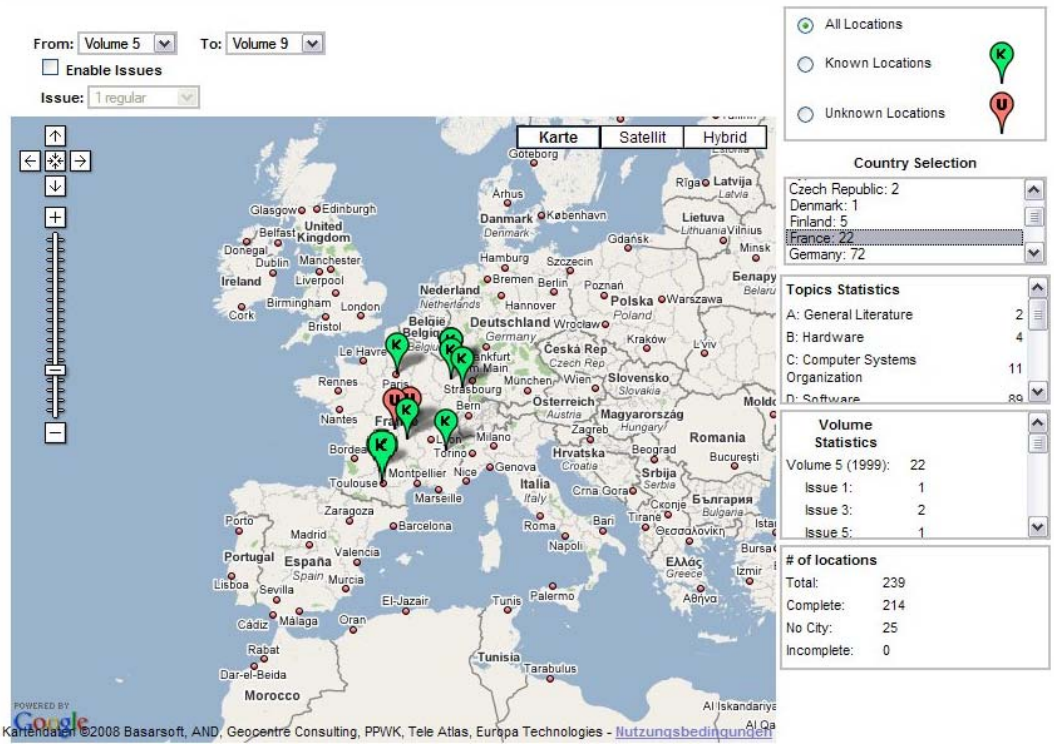


Figure 6.10: Search result filtered by country (in this case France)

Further the GUI also provides storage for temporary data like search results because this data is integrated in the map and in the statistical information blocks, although not all of this data is visible at all times (i.e. the content of a marker's info window will only be visible when the info window is opened).

7. Evaluation of the J.UCS Mashup Prototype

7.1. Interfacing Map and Journal Data

The developed J.UCS Mashup presents a way to interface a digital journal's data with a geographical map, which constitutes a novel approach of extracting information from a journal. The key in realizing the Mashup was the integration of different databases - the J.UCS database on the one hand and geographical databases (GeoWorldMap and Google Maps) on the other - into an application. Integrating a map into the application and accessing the Google Maps location database, which is used to translate addresses to coordinates is easily achieved by using the Google Maps API. For the access to the J.UCS database and the GeoWorldMap[35] database Java Servlets were developed during the deployment of the prototype. These Servlets are part of the Mashups core application, but they could also be seen as a basic J.UCS and GeoWorldMap data API. What all of these data interfacing means have in common is the fact that they use XML for communication with the Mashup application, thus making the remixing process of the databases straightforward and simple.

[35] http://www.geobytes.com/FreeServices.htm

7.2. Faced Problems

Three main challenges arose in the early stages of the Mashup development:

1. Data storing and cleaning
2. Automatic location determination
3. Social computing

The prototype has to deal with these problems at different stages of its work. However, challenge 1 depends on the proceeding of challenge 2, which again depends on the proceeding of challenge 3. The following subsections explain why these issues turned out to be problems and how they were dealt with.

7.2.1. Data Storing

As the J.UCS data relevant for the Mashup is initially stored on and managed by a Hyperwave IS/6 machine, preliminary experiments tried to incorporate this information directly into the Mashup, when the user sent a request from the GUI. However, it became clear due to the storing of metadata on papers and persons in an XML file that editing of data would lead to file writing operations and each request of a user would lead to a lot of file reading operations. Another argument against the direct approach to access J.UCS data is that for some cases an urge to store this data relationally is given. For example, the determination of other institutions which an author has worked for is easier when he is directly linked to an institution relation instead of looking through each XML-file of papers he has written.

The above-mentioned problem identification lead to the J.UCS data migration module as described in Section 6.2. The storing of information in a relational database fulfilled the needs of the Mashup prototype. Unfortunately, storing J.UCS data in a relational database is not free of disadvantages. First of all the database has to be in sync with the IS/6, which means that if new issues in J.UCS are published and therefore put on IS/6, information on the new publications shall also be migrated to the relational database for the Mashup. Vice versa, if data is edited by users or missing or not given information is determined by

the Mashup automatically and stored in the relational database, the updates have to be transported to IS/6 if the changes are taken seriously.

As the data migration IS/6 to the Mashup's database is handled by triggering batch processing of 2 programs, either the responsible person publishing on IS/6 has to do it manually after the publishing process or it has to be done automatically and simultaneously while publishing. During the process of writing this thesis it was done manually due to the reason that incorporating an automatic update would have gone beyond the extent of this thesis and is thus left as an improvement suggestion for the future. Also the update of data changed via the Mashup in IS/6 is subject to be discussed by the administration of J.UCS and still constitutes an unsolved problem.

Cleaning data prior to its use in the Mashup is a non-negligible factor in the relevance of Mashup results. For example, the information on the distribution over countries is insignificant if the same country is split into several units since each is spelled differently. So although it is a cumbersome approach to clean the data during the migration of the data from IS/6 to the relational Mashup database and afterwards assign misspelled names to their real cities and countries manually, drawing conclusion from the data is much easier once ambiguities are eliminated. In future, this cleaning mechanism may even be used to verify the data of an author at its entry moment. Thus the author can be requested immediately to check his information when the system cannot verify his address.

7.2.2. Automatic Location Determination

Experiments showed that in many cases giving the address of an author simply to the Google Maps geocoding facility was insufficient and didn't result in successfully displaying a person's location on the map. The reason for that was found in the fact that a lot of persons do not provide their full addresses, or the required information is contained in the wrong meta-data fields. For example, 1168 (see Table 6.1) paper authors did not provide a city or appended the city name to the institution name (i.e. "The Computing Laboratory, The University, Canterbury"), or thought that they do not have to provide the city name because it is already contained in the institution name (i.e. Stanford

University). Because the Mashup assumes that city and country information is stored in the right place it is clear why geocoding fails in many cases.

This problem resulted in the task of creating an algorithm, which determines the right coordinates of a person even when information is hidden or missing at all. This algorithm is described below and works with the information a person provided on the name of the institution he works for, the ZIP code of the city, the city name, and the country name of the institution.

Geocoding algorithm

1. Check if city or ZIP code is given in the information the author provided. If yes, jump to step 3, otherwise proceed with step 2.
2. If the person has provided an institution name, look up address information of persons with the same institution name in the J.UCS Mashup Database. If an address containing information about a city or a ZIP code is found, it is adopted by the Mashup to use in the geocoding of the person's location and the algorithm moves on to step 3. Otherwise the algorithm goes to step 4.
3. Send a request to a Google Maps geocoder. The responses of this geocoder are coordinates of the most probable result (most probable because there often exist cities with the same name, e.g., Springfield, USA, which exists in the state of Illinois as well as in Missouri and a lot of other states). Additionally, the ZIP Code, if given, narrows the result (as e.g. in the given example about Springfield). If Google Maps cannot find a location although the address contains the ZIP code and/or city and country, go to step 4.
4. Analysis of the institution name: The institution name is separated into single words and each word is checked for an occurrence in the city table of the GeoWorldMap[36] database. Such a match yields that the institution name contains the name of a city and the GeoWorldMap database also contains the latitude and longitude coordinates to pinpoint this city on the map. However, because

[36] GeoWorldMap is a database which contains coordinates and other useful information like Country and Top-Level-Domain of the country and can be retrieved from the internet and inserted to many highly used database systems (like SQLServer, MySQL, Oracle).

sometimes city names consist of more than one word, e.g. New York, caution is demanded and a second run of looking up GeoWorldMap DB is done by checking for a city with two words that stand next to each other in the institution name. Found results of the second run would override the first one (e.g. in case the institution name is “Columbia University, New York”, New York is the requested city and not York). With the coordinates directly taken from GeoWorldMap DB geocoding is completed and the algorithm ends. Otherwise the city could not be determined and the algorithm continues in step 5.

5. The algorithm was not able to determine the city name but if at least the country as the only location information is given, a geocoding request to Google Maps will be made, which returns coordinates to a location somewhere in the heart of the requested country. If no location information is available at all geocoding will fail.

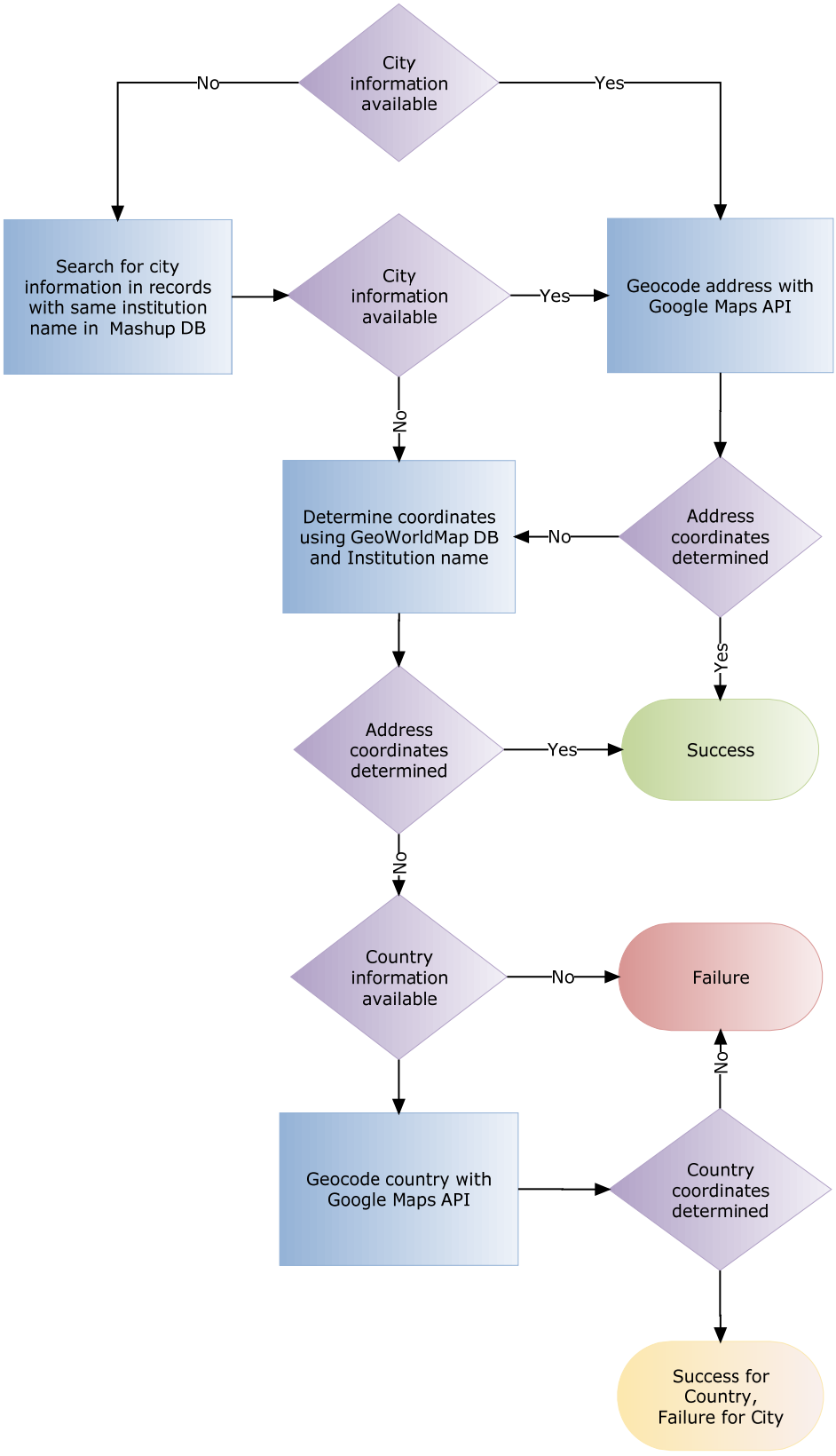


Figure 7.1: Flow chart of the geocoding algorithm

To sum up, geocoding is generally done by the geocoding facility offered by Google Maps. If the city information is missing though, it is tried to determine the city name by checking if another person has provided it and is working at the same institution. Otherwise the institution name is analyzed to check if it contains a city name aided by the GeoWorldMap DB. If the latter fails as well, just the country name is converted to coordinates somewhere in this country.

If the geocoding algorithm determines coordinates for a location successfully, these coordinates are inserted as information to the location in the J.UCS Mashup database. Thus on the next Mashup request to this data no more geocoding has to be done.

By applying this algorithm, the amount of authors whose city information is known has increased significantly (compare Table 6.1 with Table 7.1).

Total Number of Author addresses:	**2014**	
Number of addresses where City is known:	1743	87%
Number of addresses where City Information is missing:	271	13%

Table 7.1: Known vs. unknown city names of addresses after the geocoding algorithm run

Disadvantages of the automatic location determination

Although the attempt to find location information with the described algorithm, the correctness is never ensured due to various reasons:

1. Getting address information by relying on the assumption that another person associated with the same institution also has the same institution address can be wrong. For example, has the “University of California” has campuses in different cities (Berkeley, Los Angeles, etc.). The algorithm could then assign a person the address of the wrong campus.
2. The Google Maps geocoder as well as the institution name analysis can lead to coordinates of the wrong city although the city name is correct (like in the Springfield dilemma mentioned above). The result determined by Google Maps is

based on the size of the city, so i.e. smaller Springfields are neglected. In the institution name analysis the first record in the result set of the database query is taken for the coordinates.

7.2.3. Social Computing

With the lack of accurate address information of persons involved in J.UCS and the fact that the location determination algorithm cannot resolve this issue completely, the idea of using the knowledge of J.UCS users to complete the missing address details was born. Oriented on other Web 2.0 platforms like Wikipedia, the vision for the J.UCS Mashup was to harness collective intelligence and knowledge of the user community to gather the unknown pieces of information more effectively than by employing a person to seek contact to the hundreds of authors who did not provide their working location in the needed way. Following this path led to problems well known to other web services trying to take profit from collective knowledge too. There is no guarantee that a user who modified a person's data provided correct information. It is not hidden from public that a lot of people try to do harm on the internet; often for their own profit, sometimes to satisfy desires for revenge or sometimes they do it because they think it is funny or are simply bored. The question is how to decide what is correct and what has to be reverted.

The initial way of giving users total control of editing data and confirming location would surely end up in chaos as people could change data all the time, giving open access to destructive forces. However, locking out the community of the editing process would again completely prevent J.UCS from profiting from a community's intelligence and would limit the gained knowledge and experience dramatically. Therefore a balance between the anarchic way of letting the users do what they want and the total restriction had to be found. This balance is found in reviewing the modifications done by users through a qualified person on a regular base. This means that an interface to confirm or deny edited data is provided for a J.UCS administrator, which has the responsibility to cross-check this data and is able to filter out false modifications.

Another concern facing the social computing component of the J.UCS Mashup is the fact that the community using the Mashup is too small to gather the required data. Hence the

author of this thesis wants to make it compulsory for the persons in charge of J.UCS to encourage readers of J.UCS to take a look at the Mashup, because they could find colleagues of them on the map and could provide J.UCS with the missing information on them.

7.3. A J.UCS Mashup Usecase

In this section an example is given how readers, authors or the administration of J.UCS can use the Mashup for making decisions. In this example a user (whether it is a reader, author, editor or administration person) is interested in the distribution of authors who published a paper in J.UCS which is assigned to the topic category “Hardware” (ACM category prefix “B”). So the user configures the Mashup by using the browsing type “Topics”, selecting the requested category “B: Hardware” and defining the range of volumes that should be included in the search from “Volume 0” to “Volume 13” (the latest volume at the time of writing of this thesis). Finally, by triggering the “View” button, the search is started. The result of this search is illustrated in figure 7.2.

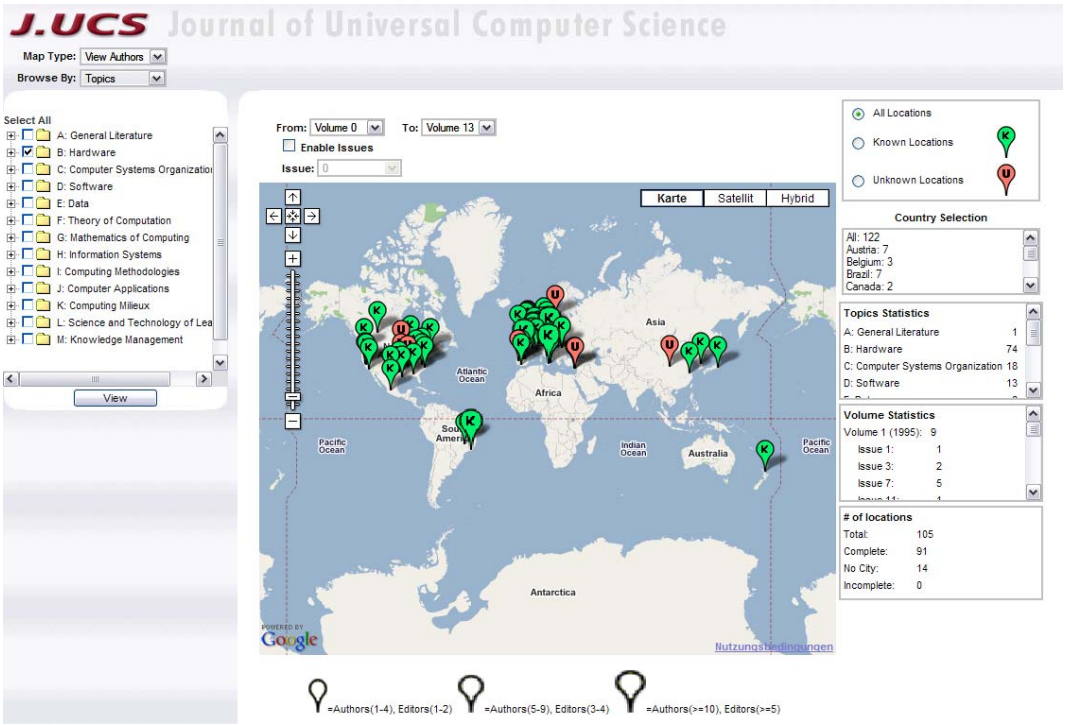


Figure 7.2: Distribution of authors who published on subject "Hardware"

From this result immediate conclusions can be drawn. Figure 7.2 clearly shows that there are no authors from Africa and Australia, and there are hardly any authors situated in Asia or South America, except Brazil. The vast majority of experts in hardware is found in Europe and North America. For readers and authors interested in working on this subject and considering to move to a place where they are surrounded by experts and where conferences on their subject of interest most probably take place, Europe or North America recommend themselves. The administration of J.UCS might wonder why no papers from Africa or Australia are submitted and might trigger another search on the hardware topic, only this time by searching for editors. The distribution of the editors on topic “B” is shown in Figure 7.3.

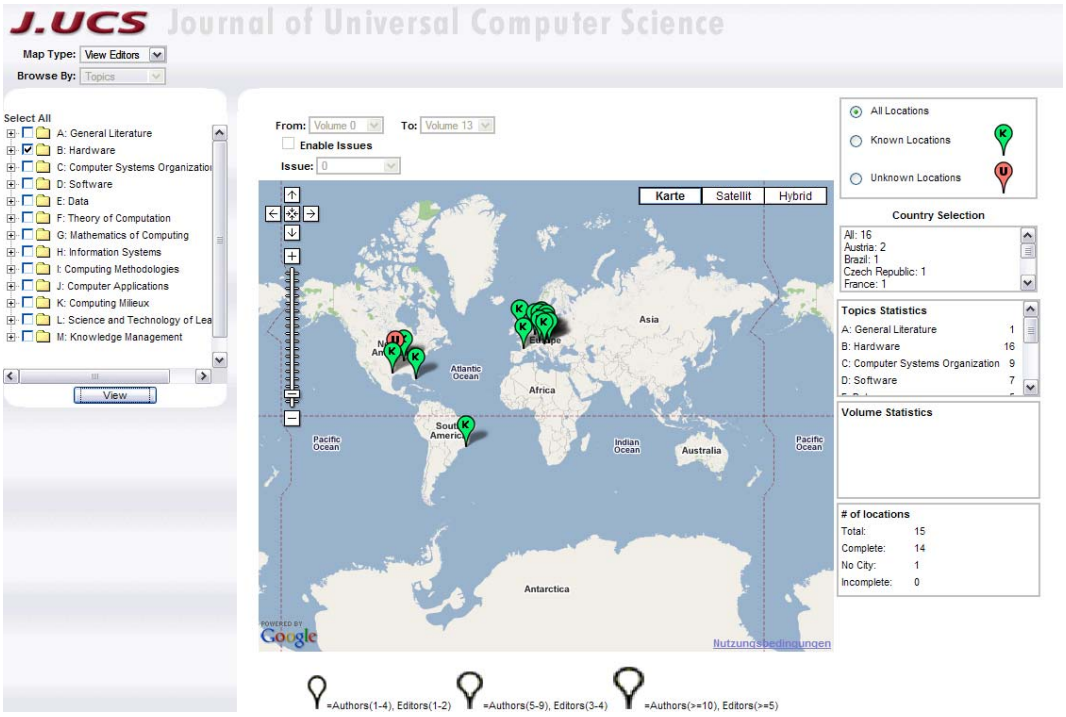


Figure 7.3: Distribution of editors on subject "Hardware"

The result in figure 7.3 obviously illustrates that editors on "Hardware" are only located in Europe, North America, and Brazil. Comparing this result with figure 7.2 gives a hint why no papers subjecting hardware are found in certain regions of the world. If this absence of editors in Australia or Asia is considered as a serious lack of influence of J.UCS in these regions by the administration, the finding via the Mashup can lead to efforts in hiring new editors, which hopefully leads to a gain in global importance of J.UCS.

The example discussed in this section presented the power of the J.UCS Mashup as a decision making tool. Considering the fact that the detection of the absence of editors in certain parts of the world and the logically resulting absence of published paper from these areas was surprisingly easy, this Mashup suggests itself to the J.UCS administration and the readers strongly.

8. Summary and Outlook

In this diploma thesis the introduction of an emerging new web architecture paradigm called Mashup was applied to the Journal of Universal Computer Science (J.UCS). For this work the evolution of the World Wide Web to the current state has been presented, research on technologies which are involved and embedded in the paradigm has been done, and technologies to be studied by the readers of this thesis have been processed. Furthermore, has the thesis discussed the appearance of digital journals, the benefits for the scientific community gained by them, and the role J.UCS plays, and how the journal is silhouetted against others. Based on these insights an idea for a J.UCS Mashup prototype has been born, designed, and implemented in the practical part of this thesis, which operates as an J.UCS metadata visualization and administration tool by displaying this data with the help of Google Maps, and embraces social elaboration to enhance wrong or missing information pieces.

Web 2.0 is the buzz phrase for the current state of evolution of the WWW. The thesis analyzed columns, like remixable data sources and harnessing collective intelligence, that carry the Web 2.0 concept and the technologies that emerged from it, like social networks or wikis. This analysis showed the advantages that could be gained by using the one or the other component of the concept. Further it showed that the analysis forms an important base to understand Mashups and encircles the possibilities for enhancing a digital journal with functionality and knowledge from the Internet.

Consequently the advantages of outsourcing functionality respectively including external data to a web application has been themed in the topic about Mashups. Through the discussed sprouting of available data, accessing APIs and data remixing tools, means became available to link various databases almost automatically and this fact will surely stimulate efforts of joining databases from all over the internet into all kinds of applications. The importance for this thesis lies in the subject that Mashup host and data providers build a symbiosis where both profit from each other and therefore the usage of an external source for J.UCS in a Mashup enriches the journal evidently. It is the true

belief of the author that the future will even bring more functionality to the journal as innovative tools appear for the remixing process and therefore shorten Mashup development time.

The introduction of Mashups and the philosophical concept of Web 2.0 are completed in the development basics of Mashups by the topic of Web 2.0 technology basics, which deals with aspects important for the thesis. For the desired prototype, information on web architecture styles is given and enhanced by proficiency in XML as a data communication and metadata description standard. As the mentioned data communication in Mashups is a very important factor in this thesis, an explanation of AJAX as a way of drawing web application towards the performance of desktop programs is given. Additionally, as web applications, especially the J.UCS Mashup prototype, are mostly data driven, the data layer in the form of a relational database is discussed.

With this Web 2.0 background a rapid application development approach becomes available, which offers the possibility of including external sources in a convenient way which not only deliver structured data but also parts that users can instantly interact with. Moreover, these sources, assembled together with own data, can result in a data mix application called Mashup.

To create a Mashup for J.UCS it was necessary to analyze how digital journals and especially J.UCS work and what experts think whcih functionalities they should offer in the future because it gets clearer which data sources could possibly be used for the implementation. It has been listed what J.UCS already supports and which defects have been found. Based on this analysis, visualization of J.UCS data was considered as a realizable feature through Mashups.

The decision was made to create a data visualization and administration Mashup and this thesis shows the efforts and the outcome of a prototype of this Mashup. Along with the designing of the application a new architectural concept was introduced which is derived from a multi-tiered client/server architecture and is called “Middleweight-client-server-system” by the author of this thesis. The importance of this architecture lies in the realization that the data of multiple, mostly external sources is mixed on the client side unlike in closed client/server architectures.

The development process of the J.UCS Mashup brought to light that displaying J.UCS data in a map was not as easy as it seemed to be and administrating values were hard to derive from a Mashup. Thus the following conclusions have been drawn:

- First of all the combining metadata from XML files on a Hyperwave IS/6 server with external data stated a task which could not be fulfilled due to the fact that the task of incorporating the Google Map API into IS/6 seemed impossible and that data was not stored in a relational way. Therefore the Mashup had to run on a separate Tomcat server and the data had to be migrated to a relational database.
- During the data analysis it occurred that a lot of information was omitted by authors or editors when they were asked to provide additional details to their papers which were stored as metadata in the XML files. To solve this issue the thesis discussed an algorithm to determine missing attributes. However, the algorithm could fail when too little information is available and also guarantee of correctness cannot be given.
- Another way of correcting and completing data of J.UCS was achieved by letting users incorporate their knowledge through data editing masks and dragging of markers. But mistaking is part of the human nature and also acts of vandalism can't be excluded.

However, the prototype still extends J.UCS with novel functionality. The displaying of authors on a geographical map was successful and statistical as well as administrative information can be accessed in an innovative way now. Also by finding a balance between open access to the data editing process and total control by the administrating executives, J.UCS data will get more accurate in shorter spans of time.

Mashup usage in J.UCS is still in its infancy, so it is expected to incorporate more remixable data sources in the future and it is thinkable to offer additional Web 2.0 functionality in J.UCS. Perhaps someday papers can be directly linked to blogs of authors or to the podcast of the lecture an author holds at university to the subject he published in J.UCS. It is even possible to build social networks of authors, where the system recommends other authors or editors to them, who are interested or have already published on the same topic they research on. This could even be supported by wikis where they collaborate on their scientific interests.

The above-mentioned ideas could be incorporated into a whole J.UCS research gateway. It is also possible to think in a totally different way by not offering all this in a centralized environment but instead making J.UCS itself remixable. The modular design of the J.UCS Mashup and the relational database model in J.UCS could set the starting point for creating a J.UCS API, an approach also other digital journals and libraries could follow as well, which would make research through libraries easier and more complete once tools evolve that combine all of these libraries.

Additionally, development efforts are taken by the IICM to administrate changes in the author and editor data made by users of the Mashup. For the future additional features are already under development, including a mechanism to determine paper acceptance patterns to compare the amount and distribution of submitted papers with the ones that were actually accepted and published.

To sum up, a Mashup is finally available that successfully integrates J.UCS data into a map. It shows that J.UCS can be mixed easily with other sources of data using structured data. In other words, further applications can be created without difficulty using the J.UCS database besides others. The achieved state of enhancement of J.UCS and the benefits of remixing data using the Web 2.0 paradigm should be taken as good reasons for further explorations on the subject of this rapid web application development through Mashups by the academic community.

9. Personal Lessons Learned

The author of this thesis originates from a mathematical approach to computer science. Nonetheless was the work on the emerging concept of Mashups which actually uses already existing technologies and merges them into something new, an exciting experience and extended his competency in more philosophical and architectural aspects of computer science. Moreover was the work and the research on J.UCS as well as other journals an opportunity for the author to gain more knowledge on working with journals for scientific research than the average writer of a diploma thesis.

Although previous knowledge of the technologies involved was brought along, taking these technologies and assembling them to a Mashup, which could be a nearly unlimited data resource, was an eye-opening adventure.

Last but not least, the team surrounding the author in his work played an important part in finishing this thesis, whereby he wasn't the central point, but hopefully a valuable team player and an important cog in the J.UCS machinery. Therefore a general thanks goes to all the people of the IICM who were involved in supporting and motivating the author and giving him the possibility to realize this thesis.

A. J.UCS Database Creation

A.1. Normalization

Following [Cod90], normalization is a process of splitting up relations into smaller relations to prevent anomalies in the behavior and meaning of data, which are caused by insertions, updates, and deletions in the database. Normalization is done in up to 9 steps, of which each ends up in another normal form. This section only discusses the normalization up to the third normal form, because it already provides data consistency in almost every case and is also often used in commercial databases. Performance reasons can also be the deciding factor to not normalize to a higher normal form than the third, because in higher levels more relations are required and more movements between the relations have to be done, which is slowing down database transactions. For higher normalization see [Fle89].

The normalization is explained by using the example of a shop selling office products. Figure A.1 shows an example of an entity which holds records of orders that have to be shipped, identifying the customer and the products in the ordered quantity. However, the database has not been normalized, so modifying of data could lead to inconsistency. For example, if the price of a product has to be corrected and is only corrected in order with the ID 1, orders 2 and 3, which contain the same product, will not be affected.

OrderID	CustomerID	CAddress	ProductID	PName	PPrice	PQuantity
1	1	Joerg Dissauer, 8010 Graz	2	HQ Pen	10,00	4
2	2	Steve Smith, 1120 Vienna	1	Copy Paper	5,00	2
2	2	Steve Smith, 1120 Vienna	2	HQ Pen	10,00	1
3	3	Michael Miller, 8020 Graz	2	HQ Pen	10,00	10
4	1	Joerg Dissauer, 8010 Graz	3	Leather Notebook	20,00	1

Figure A.1: Example of a not normalized database of a shop

A.1.1. First Normal Form

"Reduce entities to first normal form (1NF) by removing repeating or multivalued attributes to another, child entity." [Fle89]

In above example above, this would lead to a splitting of the CAddress field into a name, a zip, and a city field, storing the information in different atomic attributes (see figure A.2). This will make updates for addresses in the database easier, as customers usually change their home more often than their name.

OrderID	CustomerID	CName	CZIP	CCity	ProductID	PName	PPrice	PQuantity
1	1	Joerg Dissauer	8010	Graz	2	HQ Pen	10,00	4
2	2	Steve Smith	1120	Vienna	1	Copy Paper	5,00	2
2	2	Steve Smith	1120	Vienna	2	HQ Pen	10,00	1
3	3	Michael Miller	8020	Graz	2	HQ Pen	10,00	10
4	1	Joerg Dissauer	8010	Graz	3	Leather Notebook	20,00	1

Figure A.2: Shop database in 1NF

A.1.2. Second Normal Form

"Reduce first normal form entities to second normal form (2NF) by removing attributes that are not dependent on the whole primary key." [Fle89]

As the details about a customer only depend on the OrderID key part and the product name, and the product price only depends on the ProductID, these attributes are relocated to another table as shown in figure A.3. The initial table references these tables with the

OrderID as a foreign key to the table with the customer details and the ProductID attribute as a foreign key of the table storing the product information. Together this two foreign keys build the primary key of the table, keeping the amount of products ordered by the customer in this order.

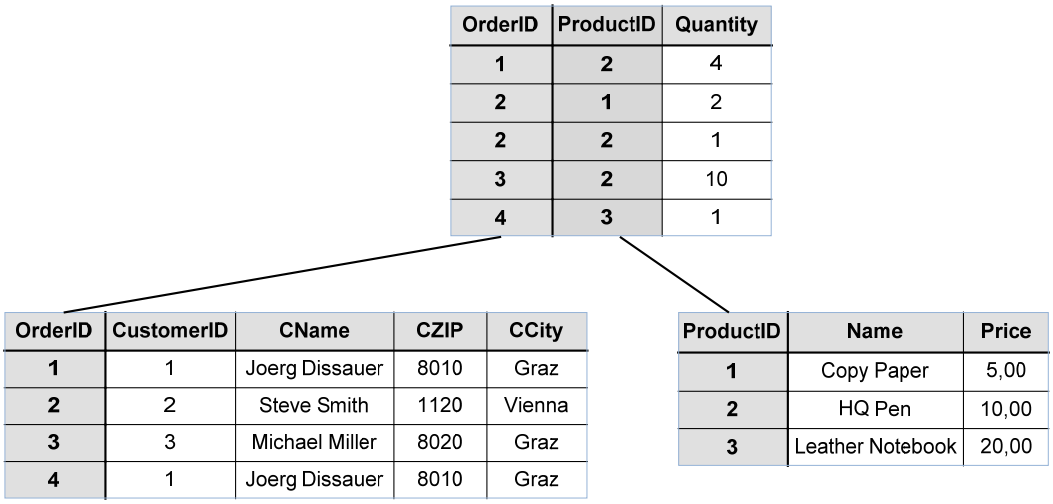
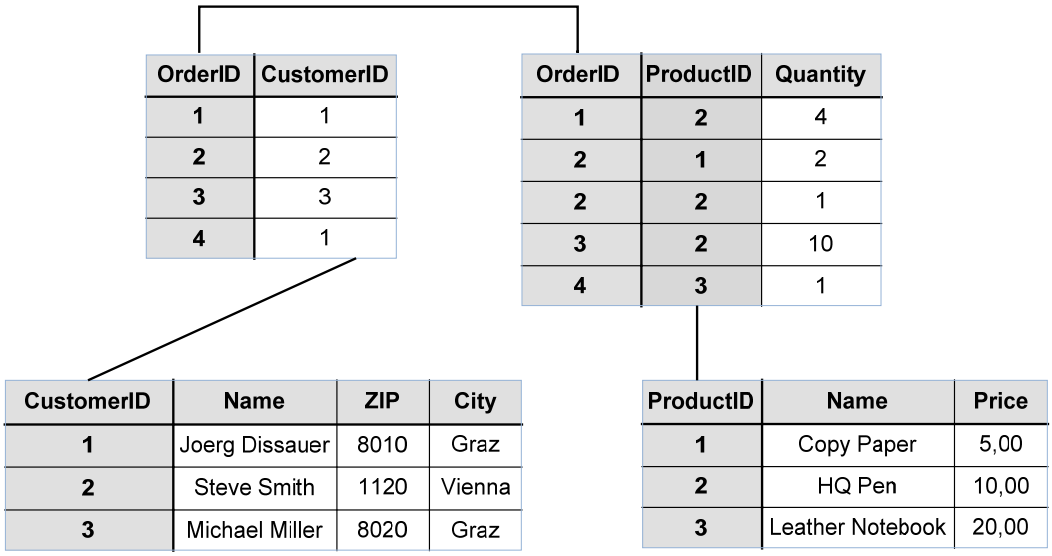


Figure A.3: Shop database in 2NF

A.1.3. Third Normal Form

"Reduce second normal form entities to third normal form (3NF) by removing attributes that depend on other, nonkey attributes (other than alternative keys)." [Fle89]

Bringing the order database to 3NF is done by identifying the fields CName, CZip, and CCity as only dependent on the CustomerID but not on the OrderID, so they have to be outsourced to another table, where the CustomerID acts as the primary key in this new table and as a foreign key in the old table which is just linking an order to a customer. Figure A.3 shows the database in the sufficient 3NF.



OrderID	CustomerID
1	1
2	2
3	3
4	1

OrderID	ProductID	Quantity
1	2	4
2	1	2
2	2	1
3	2	10
4	3	1

CustomerID	Name	ZIP	City
1	Joerg Dissauer	8010	Graz
2	Steve Smith	1120	Vienna
3	Michael Miller	8020	Graz

ProductID	Name	Price
1	Copy Paper	5,00
2	HQ Pen	10,00
3	Leather Notebook	20,00

Figure A.4: Shop database in 3NF

A.2. Database Operations

To incorporate the data of a database into an application it is necessary to have means of querying and modifying this data. Therefore most Database Management Systems support the Structured Query Language (SQL), a database manipulation language with simple syntax and easily understandable semantics due to commands, which build almost grammatically correct English sentences. SQL basically uses four commands to interact in the requested way with an application: "SELECT", "INSERT", "UPDATE", and "DELETE". SQL supports the restriction of these commands to certain records or tables by adding filtering clauses where the user can set the filtering parameters.

A.2.1. Querying

SELECT is the querying command of SQL. Queries are not restricted to a single table, but instead cross-table queries are possible. By adding ORDER BY and GROUP clauses to a query the results can further be returned in a sorted and more structured way. For

example, a query to retrieve all product names which the customer named “Joerg Dissauer” has ordered is done by the following command:

```
SELECT Products.Name
FROM Products
INNER JOIN Products.ProductID ON Orders.ProductID
INNER JOIN Orders.OrderID ON OrderCustomer.OrderID
INNER JOIN OrderCustomer.CustomerID ON Customers.CustomerID
WHERE Customers.Name = ‘Joerg Dissauer’ ORDER BY
Products.Name
```

A.2.2. Modifying

INSERT is the command for inserting new records to the database. I.e., the command for inserting a new product would be:

```
INSERT INTO Products VALUES (4, ‘Document Folder’, 3.50)
```

If the primary key value for the ID is declared as an automatically incremented value, it has to be omitted in the INSERT command.

The UPDATE command modifies records. If for instance the address of a customer has to be changed, the command will be:

```
UPDATE Customers SET ZIP = 6010, City = ‘Innsbruck’
WHERE CustomerID = 3
```

Finally, DELETE removes records from the database. If an order has been shipped, it is removed like this:

```
DELETE FROM Orders WHERE OrderID = 1;
DELETE FROM OrderCustomer WHERE OrderID = 1;
```

A.3. J.UCS Mashup Database

The described normalization process was used in the creation of the relation J.UCS database used in the Mashup. Data on papers, categories, authors, and editors, which was stored in XML files previously is imported into the relations illustrated by figures A.5 and A.6.

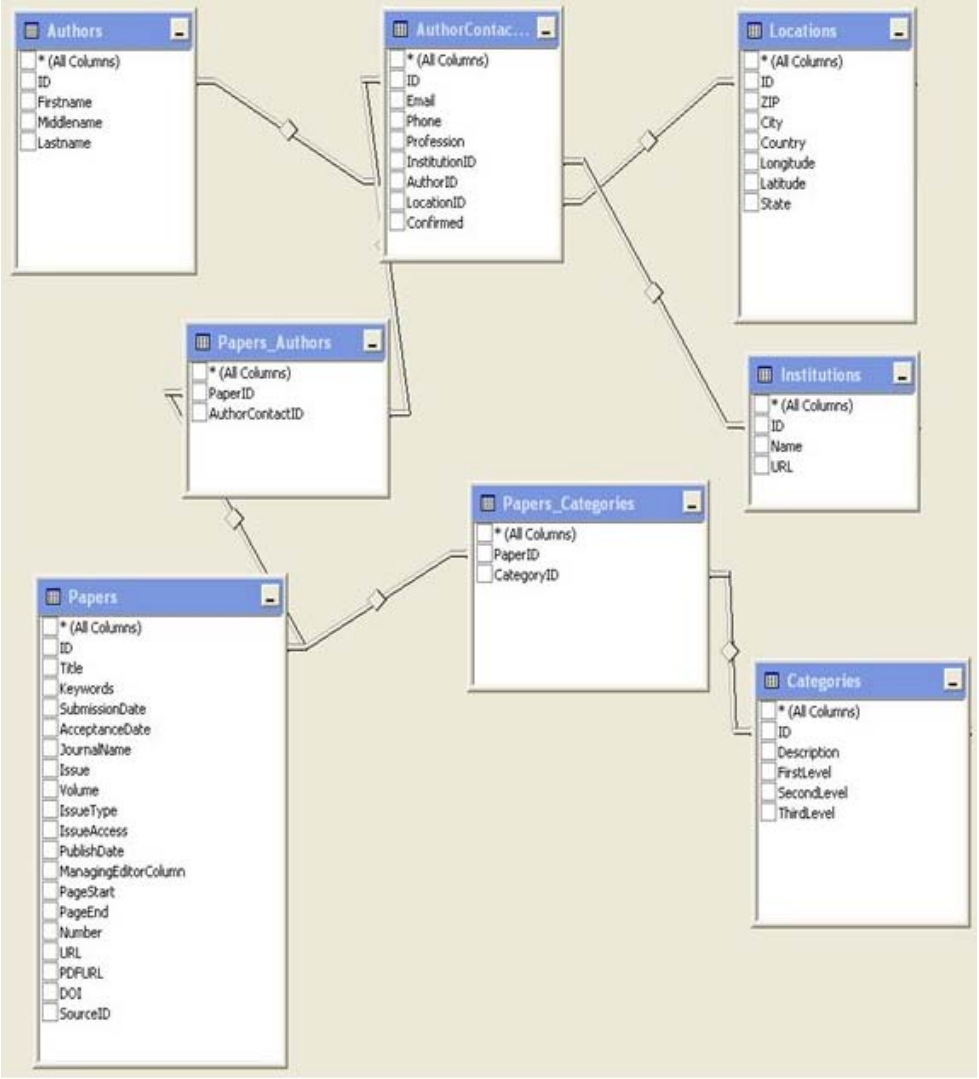


Figure A.5: Scheme of the J.UCS Mashup DB (Part 1)

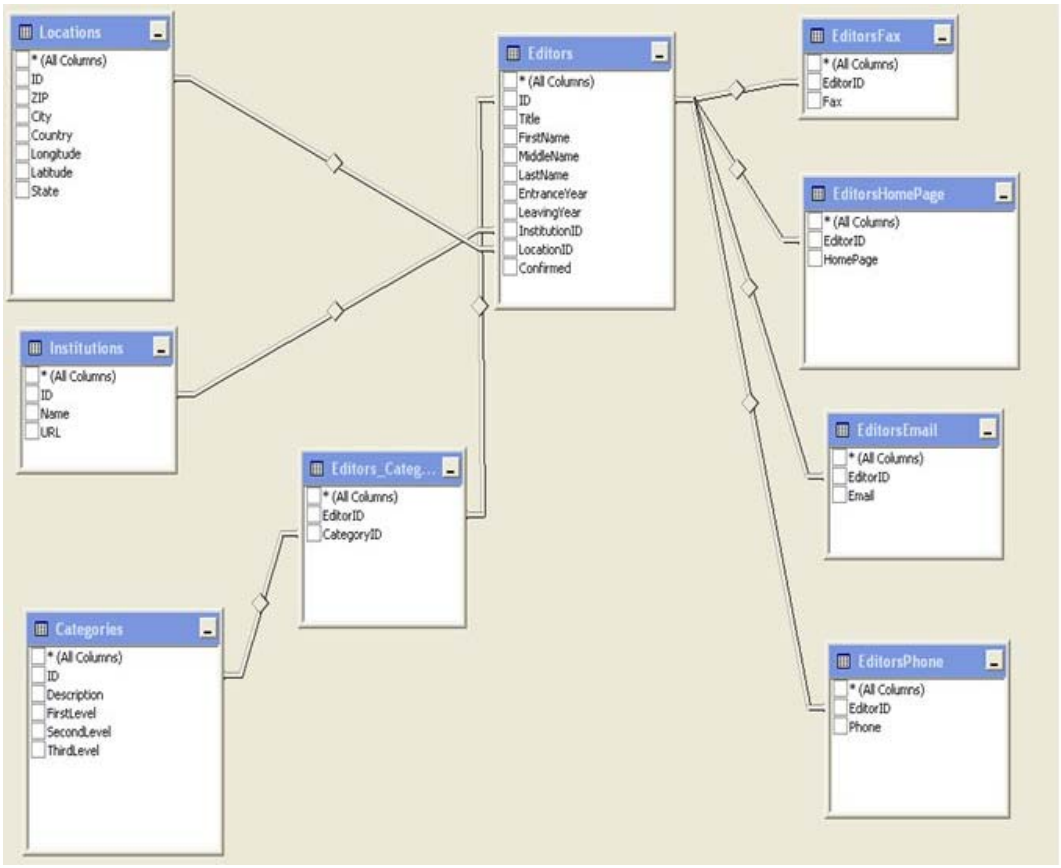


Figure A.6: Scheme of the J.UCS Mashup DB (Part 2)

This scheme shows how attributes of objects, which are handled by J.UCS, are organized in the database. For example, information on an author of a paper is, unlike in the XML file (see also section 6.1), stored separately in an extra author table, which is only loosely bound to the rest of the paper's data using keys and IDs of the tables. The database design shown in figures A.5 and A.6 thus satisfies the demand to manage the information of J.UCS using the Mashup in a relational way to keep data consistent.

List of Figures

List of Tables

References

[Cal94] Calude, C. H., et al. (1994). *Journal of Universal Computer Science*. Journal of Universal Computer Science, vol. 0, no. 0, pp. 109-115.

[Cod70] Codd, E. (1970). *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, vol. 13, no. 6, pp. 377-387.

[Cod90] Codd, E. (1990). *The relational model for database management : version 2*. Reading, Massachusetts: Addison-Wesley Publishing Company.

[Cra07] *Craigslist*. (2007). `http://www.craigslist.org/`. [Online; last visit: 26-February-2008].

[Don01] Dong, A., and Agogino, A.M. (2001). *Design Principles for the Information Architecture of a SMET Education Digital Library*. ACM+IEEE Joint Conference on Digital Libraries 2001. Roanoke, Virginia, USA. ACM 1-58113-345-/01/0006.

[Dre04] Dreher, H. K., et al. (2004). *What we Expect from Digital Libraries*. Journal of Universal Computer Science, vol. 10, no. 9, pp. 1110-1122.

[Fei00] Feinstein, W. (2000). *Database: A study of technologies for Client/Server applications.* Proceedings of the 38th annual on Southeast regional conference ACM-SE 38 (pp. 184-193). ACM Press.

[Fie00] Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* `http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf`. [Online; last visit: 27-January-2008].

[Fie99] Fielding, R., et al. (1999). *RFC2616: Hypertext Transfer Protocol - HTTP/1.1. W3C.* http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Online; last visit: 7-November-2007].

[Fle89] Fleming, C., and Von Halle, B. (1989). *Handbook of Relational Database Design.* Reading, Massachusetts: Addison-Wesley Publishing.

[Foo01] Foo, S., and Liew, C.L. (2001). *Electronic documents: What lies ahead?* Proc 4th International Conference on Asian Digital Libraries (ICADL 2001), pp. 88-105. Bangalore, India.

[Gar05] Garrett, J.J. (2005). *Ajax: A New Approach to Web Applications. Adaptive Path.* http://www.adaptivepath.com/ideas/essays/archives/000385.php. [Online; last visit: 12-November-2007].

[Gil05] Giles, J. (2005). *Internet encyclopedias go head to head.* nature. International weeky journal of science: http://www.nature.com/nature/journal/v438/n7070/full/438900a.html. [Online: last-visit: 10-December-2007].

[Hal00] Hall, M. (2000). *Core Servlets and JavaServer Pages.* Upper Saddle River: Prentice Hall, Inc.

[Hei00] Heinrich, E., and Maurer, H. (2000). *Active Documents: Concept, Implementation and Application.* Journal of Universal Computer Science, vol. 6, no. 12, pp. 1197-1202.

[Hop07] Hopmann, A. (2007). *The story of XMLHTTP*. http://www.alexhopmann.com/story-of-xmlhttp/. [Online; last visit: 12-November-2007].

[Hyp06] Hyperwave. (2006). *IS/6 Administrator's Guide.*

[Hyp061] Hyperwave. (2006). *IS/6 User's Guide.*

[Isk07] Iskold, A. (2007). *Web 3.0: When Web Sites Become Web Services.* http://www.readwriteweb.com/archives/web_30_when_

web_sites_become_web_services.php. [Online; last visit: 18-December-2007].

[Isk071] Iskold, A. (2007). *Yahoo! Pipes and The Web As Database.* http://www.readwriteweb.com/archives/yahoo-pipes-web-database.php. [Online; last visit: 18-December-2007].

[Isk072] Iskold, A. (2007). *Dapper: The Quest To Unlock Web Data (Legally).* http://72.47.210.69/archives/dapper-quest-to-unlock-web-data.php. [Online; last visit: 18-December-2007].

[Kha08] Khan, M. K., et al. (2008). *Applications of Mashups for a Digital Journal.* Accepted for publication in Journal of Universal Computer Science .

[Kol06] Kolbitsch, J., and Maurer, H. (2006). *The Transformation of the Web: How Emerging Communities Shape the Information we Consume*. Journal of Universal Computer Science, vol. 12, no. 2, pp. 187-213.

[Kro02] Krottmaier, H. (2002). *Current and Future Features of Digital Journals.* EurAsia-ICT '02: Proceedings of the First EurAsian Conference on Information and Communication Technology, pp. 495-502. London, UK: Springer-Verlag.

[Kuc98] Kuchling, A. (1998). *XML, the eXtensible Markup Language.* Linux Journal, vol. 1998, no. 55, article no. 8.

[Kul07] Kulathuramaiyer, N. (2007). *Mashups: Emerging Application Development Paradigm for a Digital Journal.* Journal of Universal Computer Science, vol. 13, no. 4, pp. 531-542.

[Lin03] Lindahl, C., and Blount, E. (2003). *Weblogs: simplifying web publishing.* Computer, vol. 36, no. 11, pp. 114-116.

[NIS04] NISO. (2004). *Understanding Metadata.* http://www.niso.org/standards/resources/UnderstandingMetadata.pdf. [Online; last visit: 27-Februrary-2008].

[Ope07] OpenKapow. (2007). *Learn more about openkapow robots.* http://openkapow.com/blogs/getting_started/archive/2007/02/15/Learn-more-about-openkapow-robots.aspx. [Online; last visit: 19-December-2007].

[Ope071] OpenKapow. (2007). *Creating Robots.* http://openkapow.com/blogs/getting_started/archive/2007/02/19/Create-Robots.aspx. [Online; last visit: 19-December-2007].

[ORe07] O'Reilly, T. (2007). *What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software.* http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. [Online; last visit: 26-Februrary-2008].

[ORe072] O'Reilly, T. (2007). *Pipes and Filters for the Internet.* http://radar.oreilly.com/archives/2007/02/pipes_and_filte.html. [Online; last visit: 18-December-2007].

[Pro07] ProgrammableWeb. (2007). *ProgrammableWeb*. http://www.programmableweb.com. [Online; last visit: 26-Februrary-2008].

[Pro071] ProgrammableWeb. (2007). *Mashup Matrix. ProgrammableWeb*. http://www.programmableweb.com/matrix. [Online; last visit: 27-December-2007].

[skr05] skrenta. (2005, 2 12). *topix weblog.* http://blog.topix.com/archives/000066.html. [Online; last visit: 27-Februrary-2008].

[Spe07] Sperberg-McQueen, C., and Burnard, L. (2007). *A Gentle Introduction to SGML.* http://www.tei-c.org/Papers/gentleguide.pdf. [Online; last visit: 22-October-2007].

[Tec07] *Technorati*. (2007). http://technorati.com/about/. [Online; last visit: 27-October-2007].

[The07] *The University of York: Client-side programming*. (2007). http://www.york.ac.uk/weboffice/docs/whb/index/07_programming/00_client_side.html. [Online; last visit: 27-Februrary-2008].

[W3C06] W3C. (2006). *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C: World Wide Web consortium. http://www.w3.org/TR/xml/. [Online; last visit: 22-October-2007].

[WCL07] WCL. (2007). *Washington College of Law Podcast*. http://www.wcl.american.edu/podcast/podcast.cfm. [Online; last visit: 10-November-2007].

[Wei05] Weiss, A. (2005). *The power of collective intelligence*. netWorker, vol.9, no. 3, pp. 16-23.

[Wes07] Westphal, F. (2007). *Mashups: Remix me!* http://www.frankwestphal.de/Mashups-Remixme.html. [Online; last visit: 26-Februrary-2008].