syndicate

Individual Service Composition in the Web-age

Inauguraldissertation

zur Erlangung der Würde eines Doktors der Philosophie vorgelegt der Philosophisch-Naturwissenschaftlichen Fakultät der Universität Basel

von

Sven Rizzotti aus Basel

Basel, 2008

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät auf Antrag von

Herrn Prof. Dr. Helmar Burkhart, Universität Basel
und
Herrn Prof. Dr. Gustavo Alonso, ETH Zürich, Korreferent.

Basel, den 11.12.2007 Prof. Dr. Hans-Peter Hauri, Dekan

To my parents, Heide and Fritz Rizzotti and my brother Jörg.

On the Internet, nobody knows you're a dog.

Peter Steiner cartoon in The New Yorker
(5 July 1993) page 61

Abstract

Nowadays, for a web site to reach peak popularity it must present the latest information, combined from various sources, to give an interactive, customizable impression. Embedded content and functionality from a range of specialist fields has led to a significant improvement in web site quality. However, until now the capacity of a web site has been defined at the time of creation; extension of this capacity has only been possible with considerable additional effort. The aim of this thesis is to present a software architecture that allows users to personalize a web site themselves, with capabilities taken from the immense resources of the World Wide Web.

Recent web sites are analyzed and categorized according to their customization potential. The results of this analysis are then related to patterns in the field of software engineering and from these results, a general conclusion is drawn about the requirements of an application architecture to support these patterns. A theoretical concept of such an architecture is proposed and described in detail.

The empirical part of the study includes an implementation of the proposal and a demonstration of the assembly of capabilities found in the World Wide Web. This implementation is based on established technologies but applies them to a new, specially-designed structure. It allows users to add selected facilities to arbitrary web sites simply by calling a specific web address. This gives the user the potential to adapt the appearance and function of web sites to his or her personal needs.

An in-depth analysis of the challenges and restrictions of the software design completes the proposed architecture. Practical examples of behavior patterns show possible implementations in a range of fields.

Finally, a vision developed from the results presented in this thesis is outlined and subjects for future research are examined.

Keywords: collaborative software, mashup, rich internet applications, semantic web, service-oriented architecture, web services, web 2.0, web 3.0

Acknowledgements

I would like to express my gratitude to people involved in my research process. Firstly my supervisors deserve my acknowledgements. I would like to thank Professor Helmar Burkhart and Professor Gustavo Alonso for encouraging me to commence the dissertation research process and also for making it possible financially. Professor Helmar Burkhart has provided me with invaluable guidance during the research and I am very grateful for all his efforts and for the time he has given me.

I would like to thank all the co-workers at the Department of Computer Science and Pharmacy, with whom I have discussed the various aspects of conducting research for dissertation. The discussions with Christina Weber, Alexander Vögtli, Martin Guggisberg, and Tibor Gyalog have particularly influenced the progress of my research and I would like to thank them for patiently listening to my torrent of words from time to time. I would also like to express my warmest thanks to Robert Frank, Beat Ernst, Pascal Betz and Adrian Ensner, with whom I have had many fruitful discussions. I am also very grateful to Andrew Brown, Stefanie Mahrer and Eric Fischer for their valuable time reading and discussing my writings.

The support provided by my family and closest friends has been truly indispensable throughout the project. Urs and Vera Schlittler made the writing process a delightful experience by hosting me in the wonderful Swiss alps and especially Ruth Schlittler deserves my warmest thanks for her endless support and understanding.

Financial support for the printing of this thesis was provided by the dissertation fund of the University of Basel.

Basel, June 10, 2008

Table of Contents

List of Figures

List of Tables

Introduction

Purpose of the study

In just 17 years, the World Wide Web has grown from an experimental research project into a technological cornerstone of the modern world. The web, originally developed to easily publish and link information, today offers a platform which can compete with desktop applications and has become a preferred platform to access many kind of applications and services. An enormous variety of often freely available software provides support and benefits for a whole range of needs in a digital life.

The rapid development in the field of web sites can reveal several trends. As regards content, increasingly more up to date data is presented. While in the formation phase of the World Wide Web mainly static pages of facts were accessible, content is nowadays continuously renewed, which offers real time behavior. According to a press release presented in 2006, 85% of the top twenty most visited web sites focus on live data [1].

In the beginning, web site representations were only text-based but have, in the meantime, been extended to multimedia publications. At the same time, handling and interaction opportunities have improved in such a way that differences compared to conventional desktop applications are no longer evident [2].

Another trend is on the linking of various content providers in the World Wide Web. In addition to the integration of information from different data origins, the functionality of external sources is embedded [3].

These advances are a quality improvement in a number of aspects: The latest data from various sources are presented in a multimedia way to the user. Interactive interventions let the user set up an individual view. Cross-connection and the use of multiple data sources allow specialization of the integrated areas. In comparison to earlier sites, today's offer more up-to-date information, enriched with related information or functionalities on a more individual level.

As a vision, I see further increase of development in these directions which will additionally be affected by the use of new devices and infrastructure. With the existence and prevalence of new display devices, the

World Wide Web will get new opportunities to establish itself. If these predictions become true, investigations of general character along these trends offer interesting research fields.

The advances of personal customization and integration of content and functionalities from external sites have one major drawback: They are all dictated by the developers of the particular web site. Functionalities, personal setting possibilities or connections to other data sources that are not implemented at the time of creation can only be supplemented at extra cost, if at all. Future breakthroughs in software friendliness may well depend on the potential to compose existing resources into new constellations. Though the wish for customizable applications that can be enriched after they have been installed is not new and solutions exist, they all demand additional effort.

The challenge of finding a way to allow users to include content and functionalities in existing web sites led to the research objective as stated below.

> **How to design an application architecture to enable an individual composition of existing capabilities found in the World Wide Web?**

To verify the theoretical model, a practical implementation shall prove the correctness of the solution. The realized application addresses the critical challenges and allows a closer examination in reality. The experiences and outcome of the usage will be discussed and analyzed.

The solution to the research problem described is generated through finding answers to the following, more focused research questions:

Question 1: How can existing web sites be categorized, and what are the resulting requirements of a software architecture which allows individual composition of these categories?

Question 2: How does an application architecture have to be designed in order to accomplish the stated requirements?

Question 3: How can the architecture be realized in practice and what are the experiences when applied to different fields?

This thesis describes Syndicate, a software architecture which proposes to answer these questions.

Strategies and sources used to answer the research questions

The phenomenon under study in this research belongs to a category of applications which would profit from a combination with other applications. In part one the existing literature and products on the research topic are discussed, and a positioning of the study in relation to these resources is presented.

Existing knowledge and positioning of the study

Based on the objective of this research, a closer description of the software category addressed is necessary. This research focuses on applications in which the combination of capabilities from disparate systems offers new functionalities and benefits the individual. Real-time or reactive systems are not part of the study. However, whenever the livelihood of a business depends on the ability to adjust quickly to changes in the marketplace or respond immediately to competitive threats, it is included in this study.

To an application developer this sounds very familiar as Service-oriented architecture (SOA) has become a de facto standard for developing component-based applications which can be accessed over a network [4]. SOA is an organized approach for applying a combination of service-orientation and distributed object computing to application architecture. SOA will be discussed in section 2.3.

This research focuses on an architecture that supports the user in composing existing services but also shows the consequences on the development side. Whenever possible, existing products or frameworks will be used to simplify the development cycle. Chapter 5 contains a survey of related projects and a comparison to requirements for Syndicate.

Dissertation structure

This thesis is divided into five parts starting from the generic and moving to more specific information. The structure follows the objectives of this study and the chosen research strategy. The content and composition of the thesis is illustrated in figure 1.

The first part provides an introduction to the research. It begins with a discussion of the emerging software solutions and the challenges they pose for service-oriented applications. After the general topic, the

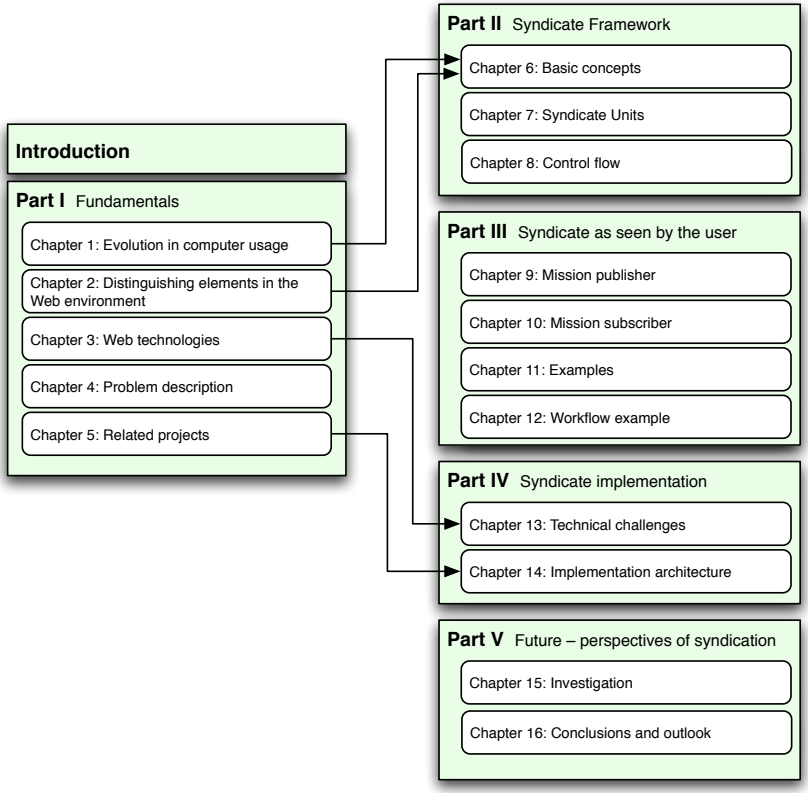

Figure 1: Dissertation Structure

study goes on to describe technologies and distinguishing elements that influence the research phenomenon. The subsequent chapter defines the research phenomenon in detail and states the research questions. A research of related projects and practical impacts on this work conclude the analysis.

In the second part the elaboration of the theoretical architecture is presented and completed with a discussion of various aspects challenging the suggested concept.

The third part focuses on the user's view. It shows strategies for combining existing facilities and provides information on how to create new implementations and access to other potentials on the web. Practical examples complete this section.

In the fourth part concrete challenges and an implementation of the generic considerations from the second part are outlined as a result of the research.

In the last part, perspectives of the future, developed on the results presented in this thesis, are drawn and possible areas for further research are presented.

Part I

Fundamentals

1 Evolution in Usage of the Web

This chapter gives a historical overview of the evolution of the World Wide Web usage. It is not intended to cover every variation of capabilities that has arisen in the past, but rather to outline fundamental changes that led to this thesis.

The most radical shifts in computer usage appeared before the internet started to exist. Quite a few concepts that have successfully evolved over time are now being transferred into the area of world wide connected computers without substantially changing the original purpose [5].

1.1 Before the World Wide Web

With the step from central, batch-oriented computing with simple input/output based terminals to personal computers, decentralized processing became a reality. New applications appeared along with the introduction of personal computers and brought more productivity to a wider public. But important characteristics found in centralized computing systems such as reliability, low maintenance, security, and the possibility for everyone to share information and applications easily, were still missing.

As a consequence, the concept was created to combine the strength of both system types. The central idea was cooperative processing where information is viewed and manipulated on personal computers and stored and processed with other data on large mainframe systems.

Inventions during that period were essentially driven by hardware advancements. Computer systems themselves have always been the center of interest but networking is a core part of today's computer system. In 1984, John Gage, Vice president of Sun Microsystem, coined the phrase "the network is the computer" to describe the emerging world of distributed computing [6]. But it took more than two decades until this vision finally happened.

This vision, coming to life in the area of the World Wide Web, brought another evolution of software applications with it. The following sections

outline web usage starting from its beginning in 1990 from the user's and developer's point of view. The main advancements closely mirror the innovations in computer history.

1.2 From information silos to rich interfaces

The first web sites that appeared on the internet were based on static information. Without interaction possibilities other than hyperlinks, data was made available as a series of interlinked pages. Web sites, acting as information silos, are still in place today and offer their content in the sense of reference books. Web sites following this principle of information silos are basically an electronic version of books where references within the book or across to other books can be resolved by selecting the reference text.

From a user's point of view the next step in interactivity came with the appearance of web-based formulae. Text fields or clickable option buttons offered the user simple ways to enter data and allowed the server to select specific data for a requesting user. In the early days the submitted information was not yet used to start processes other than selecting web pages.

Convenient interaction possibilities familiar from desktop applications were for a long time out of reach for internet sites. Both users and developers learned to get around limitations found with formula-based access. But it actually meant that users accustomed to using the mouse have been thrown back in to the area of central computing where simple input/output from a terminal was state of the art.

Different approaches tried to impose more convenient ways of interaction, mostly by distributing behavioral information that was played back by a program stored on the user's computer. In that sense, web sites were not perceived as applications but still behaved as data providers using the internet as a distribution channel. Programs on the client side needed to be installed and maintained in addition to the universal program of the web browser.

Another approach came with the recognition and acceptance of pre-existing technologies by major web browsers. With further developments in standards supporting interaction possibilities and concepts of designing web applications, a new generation of rich interfaces became reality. This circumstance made rich web applications possible.

1.3 Rich web applications

The same principles used to display server-based applications on client desktops allows applications to be displayed across the internet. Security barriers have to be overcome but as long as a client can connect to a server, applications can be displayed. The type of connection is not important and LAN, WAN, Internet, and dial-up connections are possible.

With the growth of internet usage and broadband connections, the web browser became more and more important. With browser-based execution it is now possible to start an application by clicking on a hyperlink on a web page. Software distribution and maintenance costs could be cut dramatically because the situation is likewise to central computing.

The combination of user-friendly interfaces and the rediscovery of possibilities of browser-based executions changed the way web applications were built. Figure 1.1 compares a shopping store built as a traditional web application with a event-driven way of building web applications in figure 1.2. A single page was the unit of work in traditional web applications while nowadays an application splits into components. Current applications are based on a true client/server model while early web application reduced the client part to validation issues of entered data.

1.4 Web applications vs. desktop applications

Web technologies of today can offer a rich experience to the user and are an alternative to desktop applications. Nowadays there is a strong trend towards web applications for several reasons: Supporting multiple operating systems on the desktop implies added costs to developers and administrators. End-users that lack technical skills to deal with upgrades and operating system complications require continued maintenance. Additionally the growing demand of mobile workers who want to have access to their applications has to be satisfied. Web applications address these issues; the advantages are summarized in the following list [5] [2] [7].

Installation-free: Web applications do not require installation, they are accessed by the web browser as a generic program.

Development: Reduced development costs due to operating system independent frameworks.

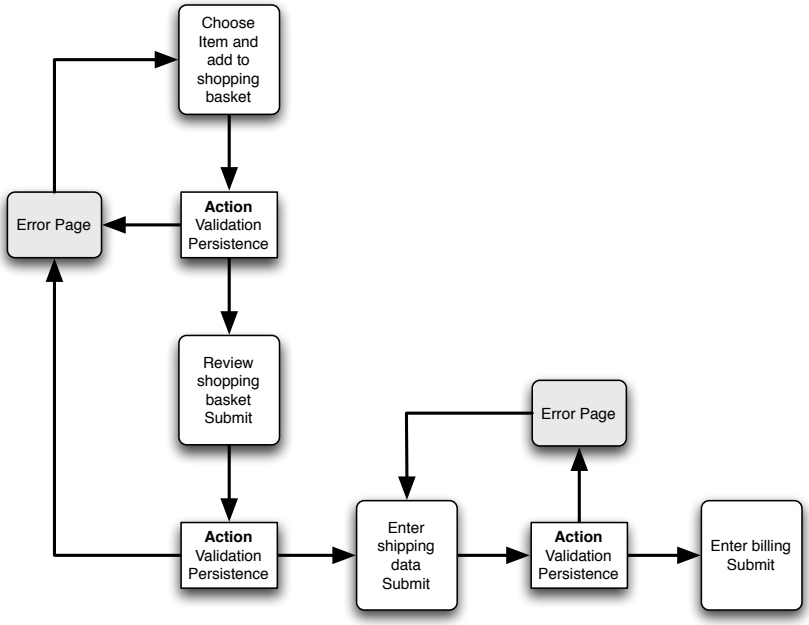

Figure 1.1: Traditional web application

Maintenance: Lower maintenance costs because of central administration. Upgrades without interrupting business.

Reliability: User problems do not require a technician to visit the desktop. Simple procedure to replace the accessing device in case of a failure.

Universal access: Accessibility from any device that connects to the internet. Collaborations with other users are possible.

Bandwidth savings: Caching strategies on the user-side and on distributed servers allow bandwidth savings.

Scalability: Easier scalable because distribution of applications have been eliminated.

Security: Possibilities to increase security because web applications only save content in a user's cache and leave the computer untouched.

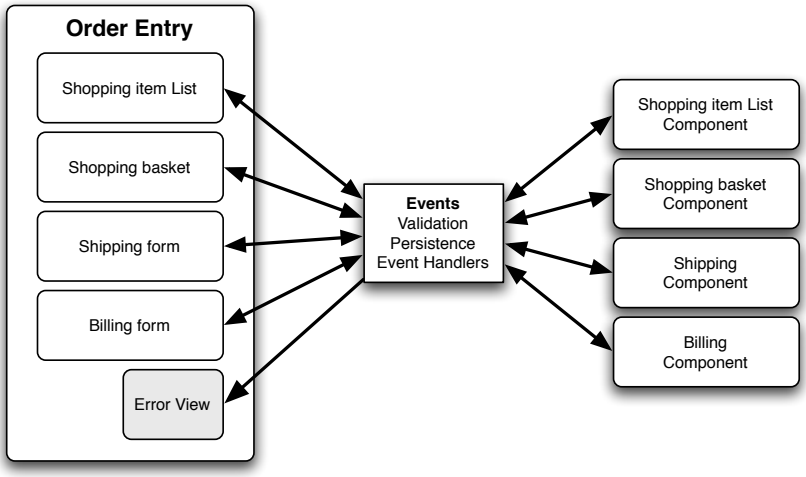

Figure 1.2: Event-driven web application

> Applications and data are maintained centrally. All user activities can be monitored remotely.

The debate of web applications that will eventually replace desktop originals did not just start with the appearance of web applications. The concept of personal computers that make use of central servers for processing activities, and mainly focus on interaction with the user and the remote server, has been around for quite a while. Unix, with its X-Windowing system, brought up similar discussions in 1970 and since then it has always been a controversial subject.

1.5 Summary

In just 17 years, the World Wide Web has grown from an experimental research project into a technological cornerstone of the modern world. The web, originally developed to easily publish and link information, today offers a platform which can compete with desktop applications.

Important characteristics already found in centralized computing systems such as reliability, low maintenance costs, security, and the ability for everyone to share information and applications paired with a rich user

experience, make the World Wide Web the ideal platform for modern, information centric applications.

However, it is remarkable that the key technologies of the web such as the communication protocol have not substantially changed since its introduction.

2 Distinguishing Elements in the Web Environment

2.1 Overview

This chapter covers concepts, architectures, and technologies that are essential in the area of web applications and are frequently referred to during the introduction of the proposed solution to the research questions. Section 2.2 starts with a review on monolithic software architectures and continues with the foundation of service-oriented architectures, a design philosophy used in order to achieve independence of involved software components. The following section deals with web services and different business models that were made possible by using the new technologies. The section continues with a closer analysis of the terms web 2.0 and Mashups since they provide central concepts. Browser technologies are finally outlined which built a starting point for later discussions.

2.2 Monolithic software architecture

One aspect of the majority of existing applications are their monolithic structure. According to a prediction of Gartner in 2003, by the end of 2008 the 40-year domination of monolithic software architectures will be relieved [8].

The design principle of monolithic architectures has different consequences and has thoroughly been studied in the context of computer kernel development [9] [10]. Monolithic systems do have their operation field, for example when processing speed and seamless integration is a core issue.

On the other hand monolithic systems try to pack as many tasks as possible into its environment. This leads inevitably to a strong binding to the chosen platform and makes it impossible to select other alternatives. With the choice of advantages from one specific system, all

the disadvantages or unwanted features in other areas are automatically selected as well.

Already learned from analyzing monolithic kernel structures, several drawbacks occur. They are very likely to recur in software applications, designed to the same architecture model [11] [12]. It has been shown that extensibility, portability, maintainability and reusability are critical properties in monolithic systems.

2.3 Service-oriented architectures

Service-oriented architectures (SOA) is a natural evolution from other software modularization programming techniques and has been around in the industry for 30 years. This standard-based technology differs from other architectures in the sense that the hardware and network have matured enough to support it. In the meantime SOA has reached a level where platform vendors, analysts, consultants and developers all say that the entire industry is adopting SOA. IBM Global Service Vice President Michael Liebow answered a question after his keynote presentation with: "SOA is the only architectural style we need to know about" [13].

Service-oriented architecture is a design philosophy with the idea of achieving Loose coupling among interacting software agents [8]. Obtaining business to business interaction among different partners with heterogeneous infrastructures is not a simple task. In the past, this has been accomplished either by time-consuming, delicate manual operations, or through hard-coded actions that are difficult to maintain. service-orientation is an approach to modularizing and organizing distributed resources with loose coupling. Applications based on this architecture principle enable great flexibility in orchestration of business processes among different companies and the automation of manual tasks.

Service-orientation uses standardized protocols and interfaces to access the underlying business logic. The offering of services through contracts allows the individual service providers to change their underlying implementation with no impact on the service consumer. Service-oriented architecture simplifies the development process, makes it easier to maintain, and manages complex distributed resources. This results in faster response to business needs and therefore saves costs.

Figure 2.1 shows a diagram of a service-oriented architecture. The characteristics which typify service-oriented architecture are summarized

in the following list [14]:

Loose coupling: Loose coupling is an architectural principle to build applications that promote loose coupling among components. Loose coupling can mean independence in terms of time or format.

Access through interfaces: Software components called services publish their interfaces in a platform-, language-, and operating system-independent way.

Service registry: Consumers must be able to find and discover services dynamically.

Communication through messages: Services are interoperable and communicate through messages.

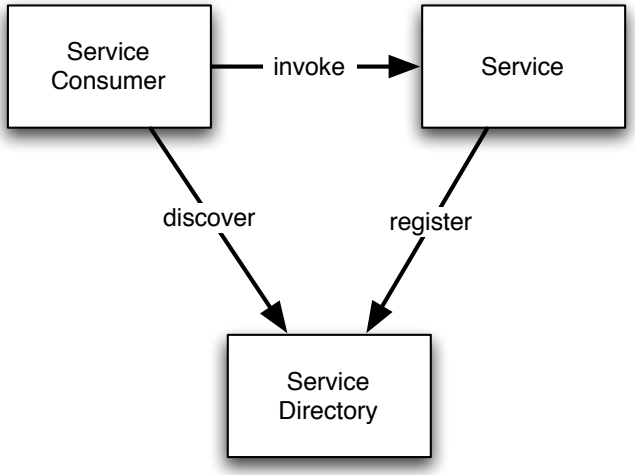

Figure 2.1: Service-oriented architecture

2.3.1 Approach to service-oriented architecture

Service-orientation is about an integration of different resources. That includes business data, legacy systems, applications, and connections to other partners. These resources are generally autonomous and can

be stored on different operating systems and are accessed via different technologies and communication protocols.

Once the decision to implement an SOA architecture has been made, a careful investigation of the business needs is the first step. SOA is more than a development paradigm. It proclaims an optimization of the alignment of business needs and information technology. Nevertheless the business goals have to be the main focus, not SOA.

The next step is partitioning of the functionality into small pieces. This guarantees an efficient response to changing business needs. Time-to-value is a critical criterion and to continually demonstrate alignment with use-cases from business needs, small steps are preferable.

SOA is gradually replacing monolithic architecture as the premier design principle for new business applications. This process is driven partly by the inherent benefits of SOA for new application projects. Reuse of application business logic is a requirement of an increasing number of projects. Personalized client/server, web-based and portal-style user interfaces demand such reuse as a central part of economically developed applications.

Different users need access in different situations using different devices to the same set of application functionalities. Such user categories can include administrators, operators, customers, managers, and employees. Devices might include laptop computers, personal digital assistants or phones; and situations cover offices, transport possibilities, hotels or public places.

In all possible combinations, unrestricted access to the back-end business functions is essential. Loosely coupled SOA, provides a natural way to achieve that business-driven demand.

2.3.2 Approach to service-oriented architecture

As shown in the next section, SOA is not the same as web-based services. SOA is a design philosophy and web-based services are an implementation methodology which uses specific standards and language protocols to expose system functionality. But web services are definitely a good way to execute an SOA solution.

2.4 Web-based services

In recent years business models have changed dramatically in electronic commerce. New technologies and greater bandwidth have made new marketplaces possible. In this section an overview of driving technologies and resulting business models is given. Examples of revolutionary applications endorse the acceptance of the different categories.

The upcoming new business models have not only influenced and changed the way producers and consumers interact with each other, but have also made a cultural shift in business behavior possible. Along with new areas of consumer involvement, additional third party services have arrived and tried to profit from successful concepts.

All business models described here are based on electronic commerce, a marketplace on the internet. Electronic commerce is a term to describe distributing, selling, buying and marketing of goods or services over electronic media such as the Internet or similar networks. The first implementation of electronic commerce started in 1993 with the first banner advertisement, placed on a web site of Global Network Navigator (GNN) by O'Reilly & Associates, Inc. [15].

Electronic commerce merges the standards, simplicity, and connectivity of the internet with the core processes that are the foundation of business. The new dominant applications are interactive, transaction-intensive, and let people do business in more meaningful ways [16].

2.4.1 Business models

Although any electronic commerce solution is unique, it is generally possible to categorize them. Depending on the participating partners the solutions can be grouped into business to consumer (B2C), business to business (B2B), consumer to consumer (C2C) or consumer to business (C2B) oriented implementations.

Business to consumer (B2C)

Business to consumer electronic commerce (B2C) is a form of communication or trade relation in electronic commerce between a producer and a consumer. In this traditional model a firm or company acts as a producer and sells goods or services to a consumer.

An online bookstore is an example where goods typically can be ordered through formula based web applications. Access has limited

interactivity and some information such as a shopping basket is stored on the company side. Frequently other services are involved as well. Credit card or shipping information can often be accessed through the same portal.

Companies providing mapped driving directions or information about public transport timetables are examples of service providers. Mapping data are nowadays often presented in an interactive manner with instantaneous adaption as the user modifies its visible area.

B2C electronic commerce has several advantages to retail stores:

Up to date Information: Information is accurate and can change instantaneously.

Instantaneous Communication: Shopping can be faster and more convenient.

Global access: Products and services can be addressed globally.

Greater availability: Limitations like opening hours vanish. Products and services are available 24 hours a day, 365 days a year.

Customization: Product selection can be adapted to customers.

Reaching a larger audience: With different advertising possibilities, a larger audience can be reached.

Direct access: Without intermediaries, faster and cheaper trades are possible.

The missing possibility of goods inspection is the main disadvantage of B2C electronic commerce. In electronic commerce, it is not possible to physically inspect the goods. The buyers have to rely on the information provided.

Business to business (B2B)

Business to business categorizes a commerce model where transactions occur between different companies or businesses. The idea is to fully automate the necessary interaction between companies without manual interaction.

Major advantages of business to business solutions include:

Cost saving: Electronic exchanges are more cost effective than paper-based ones, because many of the overheads of manual interactions can be avoided.

Speed: Processing can start immediately after receiving instructions.

Robustness: Electronic exchange can be better controlled than manual operations and is less likely to fail.

Monitoring: Electronic interactions can better be logged and used for analysis purposes than paper-based ones.

Several standards exist for Business to Business interactions, often competing each with other. Examples of already established standards are the Electronic Data Interchange (EDI), used in manufacturing, SWIFT, used in the financial world, or RosettaNet supporting trading relationships between IT supply chain partners.

Consumer to consumer (C2C)

A relationship where consumers interact directly with each other. This constellation can be initiated or supported by a platform where consumers meet. Examples of such platforms are online auctions or data interchange platforms. These community oriented platforms bring customers together and act as intermediate gateways among participating users. Depending on the role a user is playing, he can appear as a supplier or as a customer.

Consumer to consumer solutions have the following advantages:

Potentiality: Without the provided platform it is very unlikely that users would find each other to start trading.

Variety: The more users participate, the greater the variety which exists.

Consumer to business (C2B)

Consumer to Business is a variant of the Consumer to Consumer situation where the focus lies on providing products rather than trading or exchanging goods. Examples are photographic repositories, where users provide private photographs to sell via Internet channels. Other examples are providers that collect reviews where users contribute their ratings for the community.

This business model has the same advantages as in the C2C model, especially because it allows everyone to start a business, leaving out high administrative overheads.

2.4.2 Summary: The community effect

To summarize, what all business models have in common is that they get more effective the more users benefit from them and the more users contribute to the system. This community influence is one of the key factors of successful businesses and applies not only on the web. The community effect is not only relevant to consumers or producers but also to developers contributing to open interfaces of web-based services [17] [18].

2.4.3 Web 2.0

The term Web 2.0 describes technologies for a next-generation internet and can be seen as enabling technologies for the business models described in section 2.4.1. It represents an important shift in the way digital information is created, shared, stored, distributed, and manipulated. There is no precise definition but Tim O'Reilly provided a compact description of web 2.0 in a blog in 2006:

> Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more people use them." [19]

Web 2.0 is more a philosophy than a list of features and the term has evolved over time. Tim O'Reilly formulated a set of web 2.0 design patterns that describe the core of the web 2.0 [20] [21]:

The Long Tail: The majority of the content on the internet is produced by small sites.

Data is the Next Intel Inside: Applications are increasingly data-driven.

Users Add Value: In order to compete with internet applications, user must add their own data.

Network Effects by Default: The use of an application should have the aggregation of data as a side-effect.

Some Rights Reserved: Applications should focus on licenses with as few restrictions as possible.

The Perpetual Beta: Users are engaged as real-time testers with an ongoing update.

Cooperate, Don't Control: Cooperation is a central idea in web 2.0 applications and re-use of data and services are a general requirement.

Software Above the Level of a Single Device: Internet applications have to be designed to be used by multiple devices.

Tim O'Reilly further compares a platform built upon the stated design patterns with conventional applications and concludes: "A Platform beats an application every time" [21].

2.4.4 Mashups

The term Mashup [22] [23] [24] [25] originates from the music business and means the mixing of different songs together. Now almost all creative combinations of media such as photographs, videos or audio data are considered to fall into the category of mashups.

In the area of the Web 2.0, mashups signify the product of combinations of applications or content found on the web. The fundamental idea is based on the fact that content is of greater importance than applications and providers of web sites making public access to functionality and content possible. The following definition of mashup can be found in Wikipedia [26]:

> "A mashup is a web site or application that combines content from more than one source into an integrated experience. Content used in mashups is typically sourced from a third party via a public interface or application programming interface (API)."

In combining existing products into new ones lies the real power of mashups. While the participating products themselves may not be of great interest, the combination of them might offer real value to other users.

With the appearance of web interfaces, a new remix culture became possible. These circumstances form a do-it-yourself community where users play a central role as described in section 2.4.3. When everything can be mixed together, this offers a playground for creativity and led Vint Cerf, Vice President of Google to comment:

> "We know we don't have a corner on creativity. There are creative people all around the world, hundreds of millions of them, and they are going to think of things to do with our basic platform that we didn't think of. So the Mashup stuff is a wonderful way of allowing people to find new ways of applying the basic infrastructures we're propagating." [27]

In order to make the publication of mashups possible, a few components have to be guaranteed: content, data-streams, and application functionality must be accessible. Ideally providers would offer an API and allow the public to make use of the application functionality.

Beside these technical requirements, legal restrictions have to be considered as well, allthough nowadays a growing number of web users offer their content freely or based on a purchase contract to interested users. ProgrammableWeb [28] and MashupFeeds [29] provide an overview of available programming interfaces and mashups. Huge communities grew around interesting web services and that is where providers gain advantage with cost-free marketing.

Mashup types

Mashups can be built client- and server-side. Client-side mashups can allow interactive user experiences and integration with the client. Scripting languages that are executed by the users browser are a possibility to build client-side mashups.

Server-side mashups are composed of several already exposed web services or data sources merged into a single service. Server-side mashups may also only transfer an existing service into one with other functionality or they might simply implement the client-side mashup logic.

Besides more control, flexibility, and a richer environment on the server-side there are also fewer security issues to build mashups. Security restrictions on the client-side can usually be overcome on the server-side. For example integrating more than one external source at the same time is generally not possible on the client-side.

Another reason for server-side placement of a mashup is the independence of a scripting language the user's browser must understand. A user's web browser might have disabled script execution or other client devices such as mobile phones may not understand the scripting language at all.

Examples

The first groundbreaking application of Web 2.0 Mashups was Google Maps [30]. In 2005 Google allowed access to and manipulation of its licensed satellite maps with a simple programming interface. Housingmaps [31] combined the advertisements of available apartments and houses with city maps of Google maps. Other examples are local.alkemis.com [32] which provides a map of Webcams and traffic control cameras from New York city and Chicagocrime [33] with a map of crime locations found in the database of the Chicago police. These two examples were two of the first accessable mashups and meanwhile, more than 40% of mashups are based on Google maps. Figure 2.2 shows another example of a web site where restaurant locations are highlighted.

Figure 2.2: A Site that Highlights Restaurant Locations

Mashups are not limited to the use of mapping. A growing number of mashups involve multimedia content from sites providing photographs,

video streams, or include services from commercial shopping sites.

Responses of mashup servers can come in a variety of different formats, but no common accepted standard has evolved so far.

2.5 Browser technologies

All modern browsers allow the installation of additional functionality and come in "skimmed" versions to minimize software bugs and allow the user to enhance the browser as they wish. Depending on the web browser, the names for such extensions change but the general meanings are equivalent. All variants have in common that they need installation by the user in advance and all extensions are browser specific.

2.5.1 Browser extensions

Browser extensions are a way to enhance the existing functionality of a web browser with additional features. Firefox uses the term "add-ons" which is subdivided into plugins, extensions, themes, and search engines. Examples of extensions can be grouped into tools that support other protocols, additions that allow configuration changes, and other functionality. Protocol extensions may include RSS readers, or access to FTP servers. Configuration tools can be proxy server switching extensions, bookmark organizers or toolbars that add specific features. E-mail clients and debugging tools may be examples of the third group.

Browser extensions are the preferred way to bring new functionality to the client-side. Since extensions can take full advantage of the client-side environment without access restrictions, they allow every possible improvement. At the same time installing an extension means opening up all personal data stored within a web browser's reach.

2.5.2 Plugins

One typical application of plugins is the extension of the web browser to recognize and display other media formats of images, video streams or runtime environments.

While plugins may offer great advantages, they can at the same time open up security and allow others to run malicious code within the user's execution environment. The plugin API allows plugin content to interact with scripts on a page (or create their own), without restriction.

The installation of a plugin does not guarantee trouble-free integration. For example Java Applet execution is not possible without the installation of the Java runtime environment. While a runtime environment does exist for the major web browsers, it does have implementation differences among them, even if Java has been designed with the idea to avoid this problem.

The browser itself does not handle the execution of Java Applets, it only recognizes applets as an object within an HTML page and passes the responsibility over to the Java virtual machine. Only the detection of Java Applets and the configuration of runtime specific environment settings such as caching strategies are usually done with the help of a plugin.

2.5.3 Themes

Theme add-ons are a possibility to change the way a web browser presents itself to the user by modifying the graphical user interface. Themes are a packed version of color settings, button and window styles and background images. A change in themes results in a modification of the browser appearance but does not change the content of a web page nor does it increase the possibilities to support other data formats or protocols.

3 Web Technologies

3.1 Overview

Web technologies can be separated into server- and client-side components and communication elements connecting both sides. While XML has generally been accepted as a standard base for transport protocols to exchange data, different technologies at the server- and client-side are in place. This section gives a brief overview of major languages and technologies to build web applications and serves as a basis for later descriptions.

3.2 Server-side

Server-side programming describes any software that is executed by a web server. These applications reside on the server and can therefore interface with other server-side resources such as databases and e-mail. Server-side programming is based on two types: Server-side scripts and Server pages. The generated content is commonly HTML, but may be other data such as XML.

3.2.1 Server-side scripts

Server-side scripts are programs that build dynamic web pages. Such scripts interact with the web environment and execute when a hyperlink points to the code file. Typical uses of server-side programming include personalized page content or interfaces to a database with the web browser.

CGI (Common Gateway Interface) is a simple scripting mechanism supported by most web servers. In the early days those scripts were written in C, Perl or shell scripts and were executed by the operating system. Since those scripts were running in separate processes, performance was very poor. As of today these and server pages scripting languages such as ASP and PHP are executed directly by the web server itself or by

extension modules to the web server. This allows scripting execution within the same process context and is a lot faster.

Java Servlets are another technology used to build dynamic content on a web server using the Java platform. Servlets can maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting. Java Servlets have an enormous performance advantage over CGI.

3.2.2 Server pages

Server pages describe technology for embedding scripts into a web page that can interact with the web environment. Server pages are mostly standard web pages with added instructions and are therefore easier to maintain by a web designer. As with server-side scripts the output is mostly HTML but other formats are also possible. The following list summarizes common scripting languages.

ASP (Active Server Page): Solution from Microsoft which allows the usage of various languages but generally VBscript is used. [34]

ColdFusion: An application server and software development framework originally developed by Allaire Corporation in 1995, and which now belongs to Adobe Systems. ColdFusion includes services such as GUI widgets, platform-independent database querying, cache management and XML parsing, querying and validation. [35]

JSP (Java Server Page): JSP is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a web client request. JSP enables rapid development of web based applications that are server- and platform-independent. The technology allows Java code and certain pre-defined actions to be embedded into static content. JSP code is compiled into Java Servlets by a JSP compiler and runs very fast as the final byte code is similar to hand-written Java code. The usage of the Model-View-Controller pattern is greatly supported by JSP. [36]

Lasso: Lasso is an interpreted middleware programming language and server for developing internet applications developed by LassoSoft, LLC [37]. Lasso's language, Lasso Dynamic Markup Language

(LDML), can be written with procedural or object oriented techniques and structures. Lasso is an extendable language and currently includes image manipulation tools and broad support for industry standards such as XML, SOAP, WSDL, JSON, Java EE, and Java beans.

PHP (Personal Home Page Tools): With its latest release 5, PHP [38] has basic object-oriented programming functionality, database access, integrated SOAP support and exception handling. PHP has good performance because most web servers can run PHP scripts directly in the server process. Over 20 million internet domains are currently hosted on servers with PHP installed. [39]

Server-side Javascript: JavaScript was originally limited to the client-side but today different products allow programming the same language at the server-side. SpiderMonkey [40], Caucho Resin [41], Rhino [42], and JScript [43] are JavaScript engines running on different server platforms.

SMX (Server Macro Expansion): SMX is a Lisp-like opensource language designed to be embedded into an HTML page. Since SMX is a block-oriented language it is uniquely well-suited for doing block-rendering and template generation. [44]

SSI (Server-side includes): SSI is an easy server-side scripting language primarily used to include the contents of one or more files into another. Today, SSI has largely been replaced by more complex programming languages such as PHP, PERL, ASP and JSP. [45]

Ruby on Rails: Ruby is an open source project written in the Ruby programming language. It aims to increase the speed and ease with which database-driven web sites can be created. Applications using the Rail framework are developed using the Model-View-Controller design pattern. The two fundamental principles of Ruby on Rails include “Don’t repeat yourself” and “Convention over Configuration” where a developer only needs to specify unconventional aspects of their application. [46]

3.3 Client-side

Computer programming at the client-side refers to the class on the web known as client-side scripting that are executed by the user's web browser, instead of server-side on the web server. Client-side scripting is the fundamental part of Dynamic HTML (DHTML) that allows changing content depending on user input, environmental conditions or other variables.

3.3.1 JavaScript

JavaScript [47] [48] is a scripting language most often used for client-side web development. JavaScript is the implementation of the ECMAScript standard from Netscape Communications Corporation, now Mozilla Foundation and is a registered trademark of Sun Microsystems.

JavaScript instructions are often embedded within an HTML document, but they may also be requested from the web server at the time of referencing it by the document. JavaScript programs are interpreted by the web browser and therefore need a browser that supports the JavaScript language. Fortunately all major web browsers today have a JavaScript interpreter built in. While the language itself has been standardized and with the current version 1.7 a commonly accepted environment exists, the interface for document modifications is browser dependent. Cross browser techniques are used to overcome the situation of the platform variety but the behavior of scripts need to be reviewed on desired web browsers and operating systems.

Because JavaScript runs on the client-side and has greater access to information and functions available on the user's computer, the user has the possibility to protect the web browser from the execution of such scripts. But if the execution has been enabled, JavaScript offers a potential security problem with its access to information stored while the user browses through different web pages. This information includes cookies, stored passwords, history of visited web pages and other authorization data stored together with visited URL's.

JavaScript can make calls to web servers after a web page has loaded and can obtain new information which can merge with the existing loaded document. This is the basis of Ajax programming as described in section 3.4.1.

Nowadays JavaScript programming is very popular because the interpreter is included in the web browser and enabled by default. Most

interactive web sites make use of some sort of JavaScript and encourage the user to have JavaScript enabled. The lack of installation together with fancy possibilities in web site programming are the main reasons for increasing interest in JavaScript programming [48].

JavaScript has a long and inglorious history of atrocious security holes. Beside implementation errors there are numerous ways where the execution of JavaScript can raise security problems without violating any security policy [49].

Security compared to Java

Despite its name JavaScript is not related to the Java programming language. Especially when it comes to security issues, protection must be handled separately in JavaScript. Java is able to protect classes and methods by signing them and has different levels of access mechanism. With the protected, private or final attribute of a class an extension of a method is not possible. JavaScript on the other hand has no concept of method protection and all methods can be changed and properties of objects can be modified at runtime. Therefore a protection at runtime is necessary, which happens with the signing script security model as explained in section 13.1.6.

3.3.2 Flash

Flash from Macromedia, now running under Adobe Systems, was originally designed to bring animations to the client as Flash movies. ActionScript is the scripting language of Flash and a way to communicate with a program. ActionScripts allow interactive movies by controlling the flow of a movie and handling user events such as mouse gestures or key strokes. ActionScript follows the ECMA-262 standard which is the same standard as used in JavaScript.

Flash movies are not simply limited to animations. With the possibility to retrieve and send data from or to a remote server-side script, complex applications can be implemented as Flash movies. In addition, server-side scripts can request information from a database and relay it to a Flash movie. Such scripts can be written in many different languages as mentioned in section 3.2.1.

To make use of the possibilities of Flash, a Flash player has to be installed for each web browser on the client-side. Because Flash is a proprietary format and the Flash players are written by the same company

that writes the development tools, an implementation problem does not exist. Flash drawings are based on vector graphics and the graphical user interface follows the vector graphic standard. Flash applications usually load and run very fast because of the lightweight program description.

Since the Flash player is installed as a plugin on the client side, it can at the same time open up security and allow others to run malicious code (see 2.5.2).

3.3.3 Laszlo, OpenLaszlo, DHTML

OpenLaszlo [50] [51] is an alternative to the commercial Flash environment. OpenLaszlo is an open source platform for the development and delivery of rich Internet applications. Programs are written in a combination of XML and JavaScript and transparently compiled to Flash or DHTML. OpenLaszlo claims the "write once, run everywhere" paradigm similar to Java Applets. The OpenLaszlo architecture takes program code written in their LZX format and translates it to an intermediate language based on ECMAScript Release 4. Multiple back-ends continue the processing and produce an appropriate format for the destination runtime such as SWF byte code for Flash or compressed JavaScript for DHTML.

3.3.4 Java Applets

Java Applets, invented by Sun Microsystems in the late 90s are an attempt to overcome the security issues found with other client-side programming. Applets are programmed within the Java language and are compiled and stored as Java byte codes on the server-side. Whenever a user visits a web page with an embedded Java Applet, the byte codes are transfered to the client and are interpreted by the Java run-time environment. The interpretation guarantees a secure execution of the code because all access to information found on the user's side is prevented.

Nevertheless, the concept of Java Applets includes a possibility to integrate the access to user data but involves the configuration of a trusted communication agreement in advance. This situation almost compares to an installation of a program on the client-side and decreases the flexibility of the original click and run idea.

In order to make use of Java Applets, the Java run-time environment needs to be installed on the client's computer. Together with the Java interpreter, a set of libraries for communication, graphical user interface

programming and other commonly used functions are installed and made available. Additional functionality may be loaded from the server at run-time but often requires the download of multiple libraries each time an applet has been started. With the latest revision of the Java programming language, version SE 6 (JDK 1.6), it is possible to read, interpret and execute JavaScript at run-time. This allows the manipulations of running applications together with document modifications based on user interactions.

The initial enthusiasm for Java Applet programming has decreased dramatically. Missing support in web browsers, namely the Internet Explorer from Microsoft and differences in implementations made application programming difficult and unattractive.

Adobe Flash, Laszlo or DHTML programming offer alternative ways of client-side code execution without the burden found with Java Applets. Especially DHTML development has lately obtained significant importance with upcoming frameworks and support for state of the art AJAX support.

3.4 Communication

With modern application architectures, heavily based on the client/server paradigm, communication is becoming more and more important. The emerging Ajax design pattern that combines asynchronous JavaScript and XML to develop highly interactive web applications has been growing in popularity. SOAP, REST and other alternatives are protocols used in the wold of rich application models running on the web. SOAP and REST deserve a more detailed description since they are found most often in the world of web applications.

Data formats that are being transfered include XML, JavaScript Object Notation (JSON), HTML, plain text, and RSS/ATOM. The structure depends on the usage, but no common accepted standard has evolved so far.

3.4.1 AJAX

The communication layer with the server is the foundation that makes AJAX [52] possible. The most complete option for performing this communication is the JavaScript *XMLHttpRequest* object. There are other

alternatives such as hidden IFrames and cookies, but the XMLHttpRequest object is used most often. The reason for the popularity of the XMLHttpRequest objects comes from two unique features. First it provides the ability to load new content without that content being changed in any way. This makes it extremely easy to fit AJAX into the normal development patterns. Second, it allows JavaScript to make synchronous calls. This is usually not a frequently used option, but it can be very useful in cases when a request must be completed before further actions are taken [53].

Originally, Microsoft designed XMLHttpRequest to allow Internet Explorer (IE) to load XML documents from JavaScript. Even though it has XML in its name, XMLHttpRequest is really a generic HTTP client for JavaScript. The XMLHttpRequest object offers only a few methods and properties, but the implementation is different between browsers. Since these differences affect only object instantiation and event handling, they aren't hard to work around. Section 13.1.1 covers the topic of cross browser scripting in more detail.

While Ajax brings a lot of advantages in terms of a richer user experience in application usage, it also challenges developers with its security restrictions. Ajax transactions allow asynchronous data communication between client and server. The user's web browser security sandbox is responsible for keeping personal information secure. To protect against possible maliciousness, most browsers allow communication only with the computer system that hosts the web site, from which the page was loaded. If an application requires a service on a site that differs from the origin of the web site, it is disallowed by default. This restriction adds security but makes the creation of applications more difficult.

3.4.2 SOAP and related standards

The step from Enterprise Application Integration (EAI) to the usage of web services automatically requires negotiating with standardization challenges. The syntax to be used, interaction mechanism, service descriptions and naming and service lookup must be solved somehow to deal with EAI problems that occur when different operating systems are involved. These systems generally have different interfaces, use different data formats, have different security requirements and use different protocols. But these problems have already been solved with message brokers such as EAI middleware when the interacting systems are connected

through a local area network (LAN). The transition from systems on the LAN to the internet required a new protocol and other infrastructure because of the existence of firewalls that act as barriers against unwanted network traffic. Firewalls block many communication channels and make the application of conventional protocols such as RPC, RMI, or GIOP/IIOP impossible. One approach to the basic web services needed is a combination of XML, SOAP, Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) as described in this section.

SOAP [54] originally came from Simple Object Access Protocol but is not used anymore as an acronym because it not only handles access to objects and because it is also not simple. SOAP's original intent was a definition of how to send transient XML documents to trigger operations or responses on a remote host.

It is a protocol on top of HTTP or SMTP and its data is encoded in XML for one-way communication. Since the SOAP communication protocol is also stateless, it is created by design to support Loose coupling applications that interact by exchanging one-way asynchronous messages with each other [55].

Beside its main advantage of being a standard to handle complex data transactions, SOAP also has some criticism. The RPC-style mechanism of sending messages is not loosely coupled because they are not asynchronous.When performance is an issue, SOAP is definitely a bad choice as its XML messages are about 25 times larger when sending and about 100 times larger when receiving a message. This is not only a problem with bandwidth of the network, but also the generation and analysis of messages can use too much processing power compared to other RPC protocols.

SOAP implementations

SOAP implementations exists for a variety of different languages. Table 3.1 shows freely available SOAP packages:

WSDL

The WSDL was originally submitted as a W3C note by Ariba, IBM and Microsoft in March, 2001 merging three previous proposals. WSDL is an XML document used to describe web services and specifies the location of the service and the operations or methods the service exposes. The

Language	Implementation
Java/C	Apache AXIS (1 & 2) [56]
	JAX-RPC [57]
JavaScript	JavaScript SOAP Client [58]
	Mozilla SOAP API [59] [60]
	IBM ws.js [61]
PHP	PHP-Soap [62]
COM/C++	PocketSOAP [63]
Perl	SOAP::Lite [64]

Table 3.1: SOAP implementations

role and purpose is similar to that of IDLs in conventional middleware platforms [55]. The difference lies in the possibility to use different protocols for each web service while the access mechanisms were identical among all middleware platforms.

UDDI

The UDDI protocol is a specification that dates back to September 2000. As of today, more than 300 companies combine their efforts to improve the specification. The control of the UDDI project is now in the hands of the OASIS organization [65]. The UDDI V3.0.2 specification states the purpose of UDDI as:

> "A UDDI registry, either for use in the public domain or behind the firewall, offers a standard mechanism to classify, catalog and manage web services, so that they can be discovered and consumed."

The basic goal of UDDI is a framework for describing and discovering services and service providers, which is essentially a sophisticated naming and directory service. It supports the lookup of services based on a general keyword. The registry itself can be accessed as a web service.

3.4.3 REST

REST [66] is an acronym standing for Representational State Transfer and describes an architectural style of networked systems. The term resource is a central element in its definition and application states and functionalities are divided into resources. Resources are uniquely addressable using a universal syntax for use in hypermedia links and share a uniform interface for the transfer of states between client and resource. A client/server protocol, that is stateless, cacheable and layered is also part of its principles. Improved response times and lower costs due to the fact that less software needs to be written are some of the key benefits of REST.

REST architecture

REST is an architectural style, not a standard. However, the REST definition describes how to apply standards and is based on the idea of the web itself, seen as the world's largest implementation of a distributed application. The architecture is based on constraints that guide the behavior of components in order to obtain a uniform interface.

REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and,hypermedia as the engine of application state.

Web services that implement REST concepts make use of four basic functions, namely create, read, update and delete (CRUD operations) and all other methods are implemented with these operations. Generally, the realization of a RESTful web service is based on three steps:

1. Group application into resources and groups of resources.

2. Identify an address scheme.

3. Map existing functions onto the generic CRUD operations.

This simplicity is based on a principle, known as the principle of parsimony [67] and is a key argument for choosing REST.

SOAP vs. REST

SOAP and REST are both trying to solve interoperability with a communication protocol for web services. The approach of SOAP is based

on an attempt to bring in a new standardized specification while REST tries to use existing structures.

The introduction of SOAP and its cooperating protocols WSDL and UDDI led to a complex specification based on XML. On the other hand RESTful web services are based on a few simple constraints and this simplicity is what many SOAP adherents criticize. Systems are complex and therefore would require complex solutions. But it has already been shown that simple components can be sufficient to build complex systems [68] and simple standards tend to be accepted much more easily than complex ones. Code examples as well as an analysis of contract based protocols provide more reasons to establish communication through the REST architectural style [69].

Almost all currently developed service interfaces are based on REST. Simplicity and performance are the major reasons why REST has been chosen as a commonly accepted interface to web services [70].

3.4.4 Alternatives to SOAP and REST

SOAP and REST are not the only way to implement remote procedure calls. Several other protocols offer similar functionality and try to address the drawbacks of SOAP. Other protocol implementations include BEEP, CRISPY, GXA, Hessian, JSON-RPC, XINS, or XML-RPC.

4 Problem Description

This chapter narrows the subject under investigation down to a particular set of applications and analyzes possible areas of intervention possibilities. The aim of this chapter is to describe web individualization categories which allow a processing of smaller entities. Further, a list of requirements of a software architecture supporting these categories shall be formalized.

The majority of web sites offer a predefined, fixed view onto selected content, similar to paper based productions. Within web pages, simple interaction possibilities enable the user to exchange the content as the user desires. Hyperlinks, selection buttons or text-based search fields are mechanisms to communicate amendments. These possibilities have existed since the first introduction of the World Wide Web in the 1990.

In order to facilitate a more individual access to web content, web sites can allow personalization. To cope with this demand a registration process is necessary prior to visiting sites which support personalization. This enrollment can happen transparently by filling out a formula or can be hidden from the user by storing information to the user's web browser. Either way, the main purpose is to identify users and possibly prepare customized web pages for them. As a consequence, individual settings can change the web sites appearance. Such settings include presentation, functionality, color definitions or page style changes. Including additional information blocks or language settings are examples of functionality customisation.

The stated interaction possibilities for content selection or site personalization will further be addressed by the term *personal services*. Briefly, these include interaction mechanisms to individually select the content to be displayed, to change the way the content is presented and to modify the functionality of the site. These personalized services naturally depend on the content provider and usually do not conform to each other. The ways to select content, how it is presented and which sources contribute to the content usually change from web site to web site.

Nowadays, more and more web site developers realize what advantages can be delivered to the user by integrating different content providers. Therefore the number of useful services which are of general interest

is growing rapidly and developers have a hard time keeping up with facilities available in the World Wide Web. For example, it has become very popular to add interactive maps to address locations which occur on a page and developers are now adapting their web sites to answer these expectations. This is only one example of many other conveniences which make web users more productive. Other facilities which seem to be becoming a general standard are translations into different languages, possibilities to download a spoken version of a site's content or making annotations to a web site which are visible to other users.

If a required feature is missing, it is possible to add it manually. As an example, address information can be copied into another web site where the equivalent map is looked up. This additional effort also has its advantage, in the sense that the user can select their favorite content provider which issues a desired functionality. This justifies itself for two reasons: First, the user comes across familiar interfaces and does not have to learn another way of handling interactive components. Second, the user can select content providers based on the quality of the data provided. Quality criteria can be how up to date the data is, amount of data, response time or any other benchmark which brings the most useful information to the user. As such, the quality criteria is an individual rating.

While the transfer of information from one site to another to achieve a specific task seems reasonable if it is nonrecurring, it can require a disproportionate effort if the strain repeats itself. It can become almost impossible if an assignment requires the combination of several web sites: only the most accommodating users are likely to accept the undertaking of several intermediate steps to complete more complex tasks.

4.1 Categories of individualization

As outlined in the last section, personalizable web sites offer different areas of individualization to the user. These areas have been summarized by the term *personal services* and include content selection mechanisms, appearance modifications and customizable functionalities. These categories form different dimensions of individualization and are independent of each other. Transforming a web sites appearance does not affect a reorganization of the site's content or a mutation of its functional behavior.

4.2 Software architecture to support category-based personalization

In order to allow web sites to be personalized according to the stated categories, a suitable software architecture is necessary. The aim of this section is the analysis of software architectures which support a component-based design. A division into components is a consequence of the demand of the stated independent categories.

Software architectures of information portals have passed several stages in recent years and have become increasingly large. From batch-oriented processing to client/server computing and multi-tier applications, more and more layers were added to the application architecture. But the general structures are still very monolithic. Once an application has been developed and installed it is generally limited to individual needs which were not taken into account at development time.

Extensions of an application or replacing functionality with more sophisticated ones has always been a challenging task. Technical solutions exist but they still have drawbacks. Especially when it comes to the integration of functionality from independent contributers, a satisfactory generic concept without having access to the source code and the need for programming is hard to achieve.

Service-oriented web applications are replacing desktop applications as outlined in section 1.4. With the shift from application development for the desktop, to applications running on the World Wide Web, new capabilities became possible.

In this chapter universal requirements are analyzed which allow software applications to be extended by individual components and concrete demands are stated. The main goal is to facilitate dynamically composed, interoperable heterogeneous applications which integrate capabilities from different providers.

Service-oriented architectures are the preferred way of integrating different resources into a composite application (see section 2.3). Extensibility, portability, maintainability, reusability, and substitutability are the key elements when transferring a monolithic system into a component based architecture (see 2.2). The resulting demands are essential in any modular system and are summarized in the following list of operators [71]:

Augmenting: New components can be added to create new solutions.

Porting: Components can be applied to different contexts.

Splitting: Components can be made independent.

Substituting: Components can be substituted and interchanged.

Inverting: The hierarchical dependencies between components can be rearranged.

Excluding: Existing components can be removed to build a usable solution.

These operators are of a general nature and inherent in any component-based design. The operators are actions which transform existing structure into new structures in well-defined ways.

The entire internet works at this level. New functionalities and content appear every day. The only problem is that they generally do not work together. Some information found at one place usually cannot be used directly in another place. Adaptation is made by the user who copy/pastes information from one site into another. The user may also have to change the data format at the same time. In the example of a simple web formula appearing on two different web pages, a user typically copy/pastes information form one formula into another. At the same time it might also be necessary to adapt the data format to suit a particular structure required by the target web page.

4.3 Challenge of this work

A service-oriented architecture is seen as a general concept which addresses the stated requirements of a component-based design. Component independence especially is achieved with the Loose coupling concept found in service-oriented architectures. Although the entire industry is adapting to this architectural design, not all problems have been solved yet. As of today, a universal solution where distributed, heterogenous components can dynamically be found, integrated and used without programming or installing has not been achieved. This research focuses on concepts and a framework to accomplish an individual composition of components into a unified application. A concrete implementation in the area of the World Wide Web should allow a detailed analysis and serve as a proof of concept with practical examples. The stated requirements of a

component-based system serve as a leveling rule and are being used to evaluate related projects.

5 Related Projects and Assessment

This chapter provides an overview about products which address the issue of web automation and personalization. The solutions found are tested for compliance with the stated demands for a component-based system and the value of usage are estimated. The key requirements are an independence of the components and a possibility to integrate new components without installing or the need of programming.

Several systems have addressed specific tasks in web automation and modifications such as changing the presentation or the content of web pages. Essentially all analyzed products fall into two categories, the first one groups systems which work at the client side and are installed as extensions for a specific browser. The second category containing systems which work as proxy servers acting as mediator between the browser and contacted web sites.

A summary at the end of the chapter highlights functionalities of the analyzed products and possible impacts.

5.1 Projects based on Firefox Extensions

All projects within this group share the way they are installed. With one exception all products depend on the FireFox web browser and need to be installed as a FireFox extension (see section 2.5.1). As products which have to be installed locally they do not comply with the demand of installation-free usage. Nevertheless the analyzed projects offer interesting functionalities which have to be taken into account in further solution strategies.

Chickenfoot

Chickenfoot [72] offers a programming environment in the Firefox web browser for writing scripts to manipulate web pages and automate web browsing. Chickenscratch is the programming language used and is a superset of JavaScript including special functions specific to web tasks.

The primary goal of Chickenfoot is a platform where the user can define automating interactions with the web. Secondly it allows the end user to automate and customize web sites based on the web site's user interface. The example in figure 5.1 shows a formula where the user is attempted to enter login data and the equivalent code in Chickenscratch to automate the form entry.

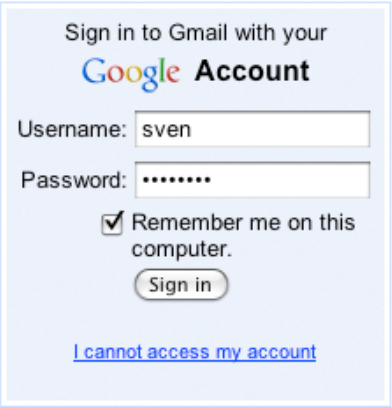

Figure 5.1: Filling out a form with Chickenscratch

Since Chickenfoot is a development environment for an end user it does not fulfill the requirement of a programming free environment. Nevertheless due to its simplicity which allows programming statements based on a users web interface, it offers great functionality which could be implemented internally.

Greasemonkey, Creammonkey

Greasemonkey [73] [74] allows users to load and install scripts which make on-the-fly changes to specific web pages. Greasemonkey scripts are persistent and changes made to the web pages are executed every time the page is opened, making them effectively permanent for the user running the script.

Installed scripts allow adding new functionality in a way as requested but require the local installation of such scripts. Configuration of script specific parameters happens through the locally installed Firefox exten-

sion. Pages where the scripts can run on are defined at development time of the script. This violates the requested principle of programming-free usage because changes would involve analyzing the code of the script.

Writing Greasemonkey scripts requires deep technical understanding of JavaScript and the source code of the page being scripted. This project offers the closest of the requested functionalities, but the usage philosophy differs. Greasemonkey puts a web page in the center of interest and allows permanent modifications of that page. With the proposed approach in this thesis, functionality builds the core of attention and a user should be allowed to call certain selected functions on every loaded page.

Several web sites [75] [76] maintain a collection of user written scripts. As of the time being it is time consuming to find scripts for a particular task. Scripts are stored in the order of the time when added to the collection and can be retrieved by simple text search. Most of the scripts treat with very specific assignments.

Nevertheless by the time of the creation of this document, Greasemonkey offered the largest collection of scripts on the net. Greasemonkey is further used as a reference while describing other products and provided functionality need to come up with the ones from Greasemonkey.

The functionality of Greasemonkey is limited to FireFox, but Creammonkey [77] and PithHelmet [78] are similar tools for the Safari browser. Creammonkey [77] claims to use compatible scripts with Greasemonkey but small differences do exist.

PithHelmet [78] is a plugin for the Safari web browser to give the user more control how web sites are shown. The basic purpose of the plugin is to block advertisements, movies or audio data from web pages. Blocking rules can be defined based on filters build from regular expressions. These rules can be exported into a file and shared with others.

Beside removing unwanted content, an advanced mode allows the execution of JavaScript code, called Machete Scripts.

iMacros

iMacros allows the user to record and replay so called "Internet Macros" for web automation, data extraction or web testing. Web automation includes filling out forms even if stretched over several pages or downloading of files and page content.

The opposite of form filling is data extraction which is also implemented in iMacros. With data extraction parts of a web page can be read and

stored to a text file. Web testing is a possibility to measure response times of web applications.

The facility to measure the performance of web applications is an important part of a component-based system with components coming from different web sites. Elements of iMacros could be helpful to support the demand of selecting a suitable partner if alternative choices are available.

Koala

Koala [79] is a system which enables users to capture, share, automate, and personalize business processes on the web. It offers a collaborative programming based on demonstration. User interactions can be recorded, edited and played back and are stored in a user- and machine readable format. A typical example is the recording of step by step instructions to fulfill a certain task such as filling out different formulas on the web.

With its user interface Koala offers a way to define and control interactions with web applications such as formula based processing. Although Koala offers a comfortable way of programming it still requires programming skills which is not intended by the demand of programming-free usage.

Marmite

Marmite [80] is an end-user programming tool which can be used to create Mashups. Marmite supports a data flow architecture, where data is processed by a series of operators in a manner similar to Unix pipes. Marmite works by displaying a linked data flow / spreadsheet view, letting people see the program as well as the data simultaneously.

The Marmite tool offers a comfortable way for building new mashups but its target audience are users willing to create new constellations rather than users who focus on using existing functionalities.

Stylish

Stylish [81] allows client-side manipulation of web page content through Cascading Style Sheets. Stylish focus on the appearance of web pages which is an implementation of the presentation pattern.

5.2 Proxy-based projects

Monkeygrease

Monkeygrease [82] is a Java Servlet which can be used to alter the output of a closed-source Java web application before its output is sent to the client. The Servlet acts as a filter and allows the injection of JavaScript, CSS and other elements to the page. The general behavior of Monkeygrease fulfills the demand on programming-free usage but it is limited to an access through this proxy server.

MouseHole

MouseHole [83] is a client side proxy server which allows manipulation of web page content using the Ruby language. The advantage over the other projects within this category lies in the simplicity of Ruby.

Privoxy

Privoxy [84] is an open source, client side proxy server which allows manipulation of page content. The capabilities include cookie management, access control and different filters to remove pop-ups, banners or advertisements.

Privoxy provides most desired functionalities but needs a proxy server and does not allow arbitrary interconnections among different services.

Proximodo

Proximodo is an open-source customizable web filtering proxy server inspired by- and interoperable with Proxomitron. It allows manipulation of page content acting as a mediator between the browser and the web site. The project comes with a set of predefined filters to block banners, pop-ups and allow the configuration of the browser's cache. It is also possible to write own filters but the scripting language is quite complex. The project is still in early development stage and the web page has not been updated regulary.

Proxomitron

Proxomitron allows filtering possibilities at the server-side and is generally used to block pop-ups and banners and remove embedded sound or

animations from web pages. A graphical user interface allows editing and modification of filter configurations. The configuration uses a language similar to the standard regular expressions used in text editing.

The first public release of Proxomitron appeared in 1999 but development has been stopped in 2003 and the project is now closed. Proximodo, as described above, carried on and cloned the software of Proxomitron continuing the work.

5.3 Other related projects

Yahoo! Pipes

Yahoo! Pipes is a web application from Yahoo! which provides a graphical user interface for building applications which aggregate web feeds and other services. These applications can combine various sources with a piping mechanism and are being published after additional filter rules have been defined.

An example is the remix of a news feed with a web-based image database. In this example tagged photos from a photo library on the web are appended to matching keywords found in the feed. Published Pipes can be accessed by other users through a shared URL.

Yahoo! Pipes follows the requirements of a component-based system with its offered pipe concept to concatenate data streams. The Yahoo! application concentrates on user friendliness through its drag and drop interface but is limited to the usage of feeds.

Since Yahoo! Pipes offers a possibility to access it via URL, the output could be included in other architectures. This could allow to make use of the comfort of the user interface without limiting the usage to Yahoo!.

Bookmarklets

Bookmarklets are another technology which can be used to execute arbitrary JavaScript code on any page, but they require a user to click them, rather than running automatically. Bookmarklets are not available as a specific product but allow similar functionality as other projects described in this chapter. The Bookmarklet technology fulfills most of the stated demands on a component-based system but the technology has certain limits (see: 13.1.5).

5.4 Summary of evaluated projects

Several of the presented projects provide functionalities of a component-based system as requested. But this thesis has its focus more on the application user rather than on the developer-side which is common in all presented projects. Nevertheless, existing solutions could contribute to an upcoming approach. Table 5.1 lists facilities of related projects that are of general interest in finding an approach to the desired component-based system.

Project	Competences
Bookmarklets	The Bookmarklet technology offers flexibility in adding individual functionality to a loaded web page.
Chickenfoot	A programming language to assist a component-based development of web facilities which includes functions specific to web tasks.
Koala	A visual editor which enables programming by demonstration.
Marmite	A spreadsheet-based programming tool for creating web-based mashups.
Yahoo! Pipes	A composition tool to aggregate, manipulate and combine RSS feeds which can be accessed by a web address when being published.

Table 5.1: Functionalities of related projects.

Part II

Syndicate Framework

6 Concepts and Terminology

Loose coupling has been identified as a key challenge to support individual service composition in the web. Service-oriented architecture (SOA) has become a de facto standard for developing component-based applications which can be accessed over a network using standard interfaces.

This chapter introduces additional requirements of a service-oriented architecture which enables pattern-based composition of existing facilities in the area of the World Wide Web. Based on these requirements a new terminology and approach for a software architecture is proclaimed. The focus lies on a component-based description and the interactions among them. An analysis and a concrete implementation accompanied by practical examples follow.

6.1 Pattern-based individualization

It has been outlined in chapter 4 that personalizable web sites offer different areas of individualization to the user. These areas have been summarized by the term *personal services* and include content selection mechanisms, appearance modifications and customizable functionalities. In this section the presented categories will be related to concepts originating from software engineering.

This separation of content and functionality together with a detached mutation of the outlook can be seen as an instance of the model-view-controller paradigm (MVC) [85] [86] [87]. I suggest a segmentation of the individualization areas into components, proposed by the model-view-controller paradigm.

With this division, the selection of content plays the role of the model, the altering of a web site's appearance corresponds to the view part and the integration of several information providers relates to the controller part. These individualization categories are described in more detail in the following sections and are illustrated with practical examples. I propose three distinguishing patterns for these individualization areas: the content pattern, the presentation pattern and the integration pattern, associated with the model, view and controller of the MVC paradigm

[88] [89] [90]. The proposed patterns will enlarge the components of the MVC paradigm with related functionality.

6.1.1 Content pattern

Individualization areas belonging to the content pattern are related to the model of the MVC paradigm. The MVC paradigm defines the model as a representation of application data and the business rules that govern access to and updates of this data. In the field of the World Wide Web, data can come from local databases or can be retrieved from external sites. The model is an abstraction of these data resources and encapsulates data access.

Advertisers which contribute small information blocks to be included in web pages are an example of the content pattern. Such a web page consists of independent data portions which appear on the same page but are not connected to each other otherwise. I suggest that the content pattern additionally includes strategies to decide on and discover the origin of the content.

6.1.2 Presentation pattern

In the case of the MVC paradigm the view renders the contents of a model. It accesses application data through the model and specifies how that data should be presented. The view is responsible for maintaining consistency in its presentation when the model changes. If a system follows this paradigm a strict separation of the content and the presentation holds and multiple simultaneous views of the same model are possible.

If a web site allows customization of its appearance, it already follows the principle of exchangeable view components. While the content of the page remains the same, the outlook can entirely change. An example of such changes in the visual representation can include modifications of font or color settings or the shielding of entire portions of a web page.

On top of changes of the visual presentation I also propose additional characteristics being added to the presentation pattern. I suggest that functionalities which change the behavior of visual components are also counted in this group. Such functionalities can include direct modification of the appearance or the enabling of additional possibilities of such modifications to the user. In any situation, customization within the presentation pattern is restricted to visual representations without modifying the content or the origin of the content in any way.

6.1.3 Integration pattern

The integration pattern relates to the controller of the MVC paradigm. A controller as defined in the MVC paradigm is an intermediate component negotiating between the model and the view. Its main purpose is to solve the problem of decoupling of the data access and business logic from the data presentation.

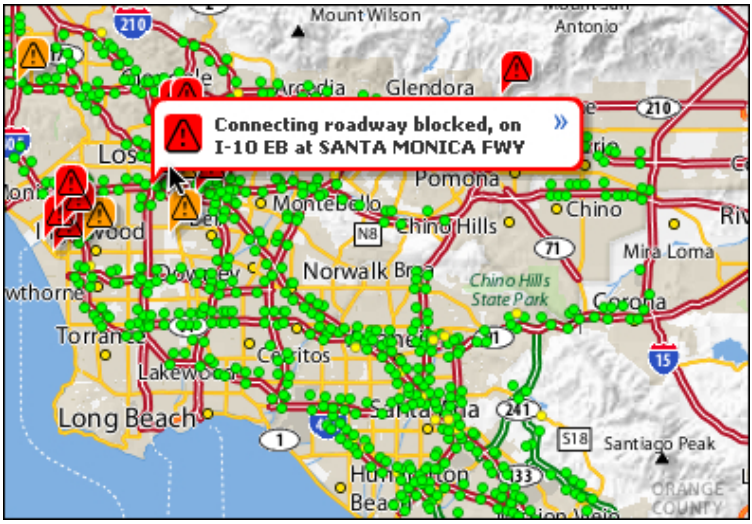

Figure 6.1: Yahoo! Live Traffic

I propose that the integration pattern be used as a structure which includes communication and integration of other models from different sites. Individualization in this area deals with the desired connections to other content providers acting as supplementary models.

A practical example is Yahoo's live traffic and driving direction map [91] that offers local traffic conditions in the United States as shown in figure 6.1. Yahoo! enhanced its existing map service to allow customers to plot a route from one local destination to another, and overlay traffic data such as road speeds and potential delays.

In this example Yahoo! combines the latest information from several different sites. Digital mapping data is provided by two competitors which offer variable qualities depending on the location and richness of detail. Real-time traffic conditions are from metropolitan transportation

departments and private providers, including embedded road sensors, traffic cameras, police scanners, and traffic helicopters.

The integration pattern is closely related to Mashups (see: 2.4.4 on page 23). From a functional point of view the integration pattern is simply another way of describing the behavior of mashups. But mashups are concrete web applications which combine data from more than one source into a single integrated tool while the integration pattern only specifies the behavior of the application, keeping the implementation open.

Table 6.1 shows a summary of the described patterns and the relation to the components of the MVC paradigm.

Pattern	Related MVC Component
Content pattern	The *model* represents application data and the business rules that govern access to and updates of this data. The content pattern additionally contains strategies to decide and discover the origin of the model.
Presentation pattern	The *view* renders the content of a model. The presentation pattern includes the view and functionalities changing the behavior of visual components.
Integration pattern	The *controller* translates interactions with the view into actions to be performed by the model. These actions result in business process activations and changes of the state of the model and selecting an appropriate view. The integration pattern contains controlling of other pattern-based components and integration of supplemental models which contribute to the application.

Table 6.1: Individualization on the web compared to the MVC paradigm

6.2 Manifold – The subsequent intention

The aim of this section is the formulation of requirements of a software architecture which enables individual pattern-based composition of distributed facilities. This is a further granulation of the demands on a generic component-based architecture which were previously stated

(see 4.2). In order to define such requirements the intent is to find and transfer established principles. The accumulated principles will be accumulated in the notion of a manifold system in contrast to monolithic systems (see 2.2).

At the class library level reusable components are common and frequently used but not to the same extent when it comes to interacting, heterogeneous and distributed applications. Building blocks from different providers usually do not automatically fit together.

The subsequent intention is a manifold system with a possibility to compose an application by integrating services from components. Components could come from different partners and should be integrable without programming. Necessary data transformations, protocol implementations and routing decisions should all happen automatically. The connected components should be independent of each other and without tight relation in between. These principles are known as *Plug and Play*, *Adaptation*, and *Loose coupling* and are used in connecting hardware devices. Especially the term "loose coupling" has gained greater popularity in combination with software architectures (see: 2.3).

Another requirement of a manifold system is the possibility to add new components which do not relate to each other directly but allow interaction among them. This concept is known from peer-to-peer networks in which users and their nodes appear *anonymously*.

If several equivalent components are available, they should be substitutable. An ideal implementation should also take advantage of the best available components without compelling interactions. If new alternative components become available, they should be discovered automatically. *Auto discovering* is also a principle from hardware device interfaces such as the universal serial bus (USB) [92]. The term to identify a selection of a favorite component is named *Best of breed* according to stock selection strategies found in portfolio theories.

Communication paths among services should be kept as short as possible. Direct communication paths should always be favorized. This principle can also be found in the field of hardware interfaces such as USB. Table 6.2 summarizes these stated requirements on a manifold system and associates a particular concept with them.

Principle	Explanation
Plug and Play	Component integration without specific knowledge about the component interface.
Adaptation	Automatically adjust to required formats and standards.
Loose coupling	Group components together without affecting each other.
Anonymity	Connect components which are not related to each other.
Auto discovering	Automatically discover available components.
Best of breed	Strategy to select one component out of alternatives.
On the go	Allow the direct interconnection among components.

Table 6.2: Properties of a manifold system

6.2.1 Advantages

Applications enhanced by following the proposed principles will benefit in several ways. Generally, it allows those involved to reduce costs and improve flexibility. Users, developers, administrators and managers are all actors who are playing a specific role along the process from creation to use of an application.

An end user should not notice any circumstances and interacting with the user interface should be habitual. But the user has the ability to adjust the behavior of each component or exchange it entirely if it is supposed to do so. Not only the user themselves could replace components, but also the manifold system could do so. Ideally, the user would automatically profit from components which best fit the user's personal needs. If a new component is made available and the quality is more suitable, the system would replace or add it without further interactions.

Developers can concentrate on integrating already existing components and do not have to program anything from scratch. Abstraction between interfaces and implementation minimizes dependencies. As a consequence new types of clients are more easily supported because the controller

logic is kept independently.

Administrators will get autonomous blocks which can be managed and monitored separately and failing components can be replaced. Managers may decide on using particular implementations on a component based basis.

6.2.2 Requirements on the applications

To make use of the advantages of the proposed principles, applications need to fulfill a certain profile. Presentation oriented information-centric applications which would benefit when combined with information coming from other resources will profit the most. On the other hand, these principles might not be the right guideline when processing speed or cpu-intensive calculations are a central point. Also, if security is critical, a distributed component based architecture naturally decreases a scrupulosity use compared to a monolithic system.

The following list summarizes properties which promotes the usage of an architecture following the described principles:

Diversification: Information from different resources.

Reusability: Integration of reusable components.

Substitutability: Components can be exchanged with alternatives.

Behavior centric: Components are defined by behavior.

Autonomous: Components are autonomous.

Integration over speed: Integration of several resources is more important than processing speed.

6.3 Use cases and basic architecture

A user participating in the Syndicate system can be assigned to a specific role. A role determines a set of connected rights and obligations.

Figure 6.2 shows primary actors and processes participating in the Syndicate system.

A conceptual overview of the upcoming architecture is drawn in figure 6.3. The architecture is based on a classical input-processing-output

Role	Rights	Obligations
Service Developer	Not restricted by the Syndicate system.	Guarantees a consistent, reliable delivering of content or functionality.
Mission Developer	Can create and add new Missions on the Syndicate server. Defines which settings can be modified by a user.	Has to guarantee that integrated services are of no risk to use.
Mission Publisher	Can publish Missions to be used by other users and restrict modifications of Mission properties.	Has to guarantee that Missions and integrated services do no harm a potential user.
User/ Subscriber	Can subscribe to published Missions.	

Table 6.3: User roles

structure. This simplified version is subdivided into concrete components and described in the following sections. The fragmentation of the building blocks is shown in figure 7.1 on page 71.

The arrangement of the units includes two feedback loops. One loop integrates information from the user viewer which is modified by the controlling block. The user viewer is used as the primary output channel but at the same allows inspections by the controller.

The second loop redirects transformed input data back to the data access block. This information chain allows stacked processing.

6.4 Terminology

The description in the following list gives an overview of the terminology of the Syndicate components used in this chapter. Each component will further be described in more detail in this chapter and is later illustrated with practical examples (see chapter 11).

Syndicate – Name of the project and the state of a loaded page respectively. If a page has been syndicated, it is responsive

to Mission triggers.

Syndicator – Element that enables syndication of a web page.

Syndicate Server – External site used to manage, configure and store Missions.

Mission – A task that changes the appearance, the content or the behavior of a web page.

Mission Patterns – Categories that group Mission operating ranges.

Trigger – Event which releases the invocation of a Mission.

Mission properties – Environmental settings which control Mission specific parameters.

Service – A web-based service or a web site which acts as information provider.

Server detour – Calling of an external service via Syndicate server.

Adapter – Channel adapters can receive messages and invoke functionality inside the application.

Transformer – Transforms and links gathered information together.

Scout – Component, responsible for information gathering.

Selector – Unit which implements a service address selection strategy.

Connector – Component which supports connections to external services.

Consigliere – Controlling unit where the application logic is implemented.

Renderer – A component responsible for the presentation of the outcome of a Mission.

Injector – A component which manipulates the current loaded web page.

DOM – Document object model, an interface to the content, structure and style of web documents.

External – External program running outside a web browser.

6.5 Involved servers

In a complete workflow several servers can be involved. The Syndicate server acts as a turntable between the user's web browser and various service providers. In this situation all communications run through the Syndicate server. However, there are other scenarios with fewer servers involved and particular situations exist which only use a Syndicate server to fulfill a task.

The configuration shown in figure 6.4 uses three different external service providers. The number of contacted external domains depend on the operation to be completed.

6.6 Architecture principles

The design of the architecture is a direct consequence of the requirements for a manifold system as previously described in chapter 4. The principle of Loose coupling in terms of time and location demands for a place where requests are buffered and participants addresses are resolved. The existence of the Syndicate server has among other issues loose coupling as its primary goal.

Loose coupling in terms of type and versions of service providers is addressed through the detachment of the Scout. The Scout offers different techniques to accept certain changes in the documents published by service providers and is resilient to variations.

Integrated services must be interchangeable or have alternatives to offer greater reliability. This expectation is resolved by the Selector component, responsible for choosing suitable participants.

Missions must have the ability to change the appearance, the content or the behavior of a syndicated page separately. A component based architecture and a decoupling of the render process from the controlling and data collecting task supports this independence.

The purpose of an anonymous usage of Missions is served with stateless service consumption.

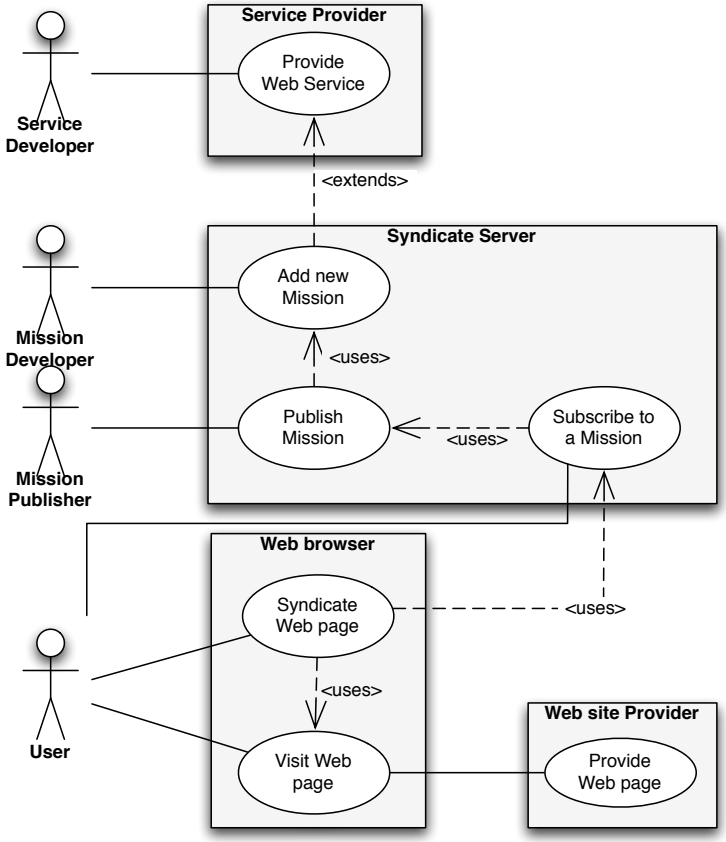

Figure 6.2: Syndicate use case diagram

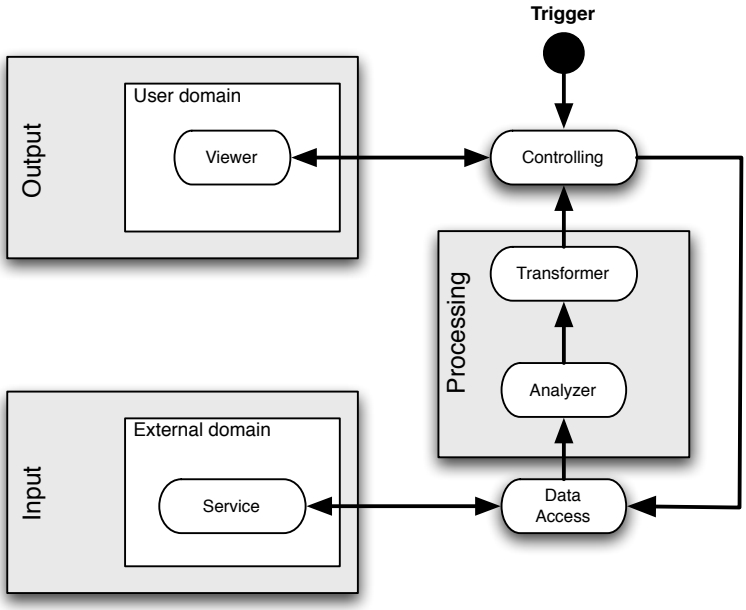

Figure 6.3: Syndicate basic architecture

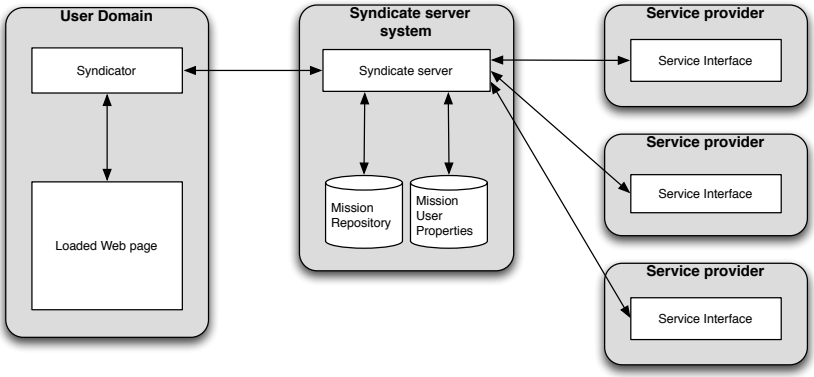

Figure 6.4: Involved servers

In order to allow an individual Mission composition, the subscription pattern is used to make use of personalized services. To address the issue of rapid development of new Missions and make use of a community, a publishing mechanism is used to widen the range of Missions.

Reusability of existing Missions at a component based level is achieved through a pipe concept copied from the Unix environment.

The following chapter describes the architectural approach to achieve these expectations. The interaction of the components and the sequence of control is given in chapter 8.

6.7 Summary

Individualization possibilities in the area of the World Wide Web have been summarized by the term of *personal services* which includes content, presentation and functionality interventions. These personal services have further been linked to the Model-View-Controller paradigm from object oriented programming. As a result, three patterns were described which categorize the individualization areas, namely the content, the presentation and the integration pattern.

Supplementary principles of a manifold system have been described which are required to allow individualization of pattern-based components emerging in the area of the World Wide Web.

7 Syndicate Units

7.1 Syndicate components

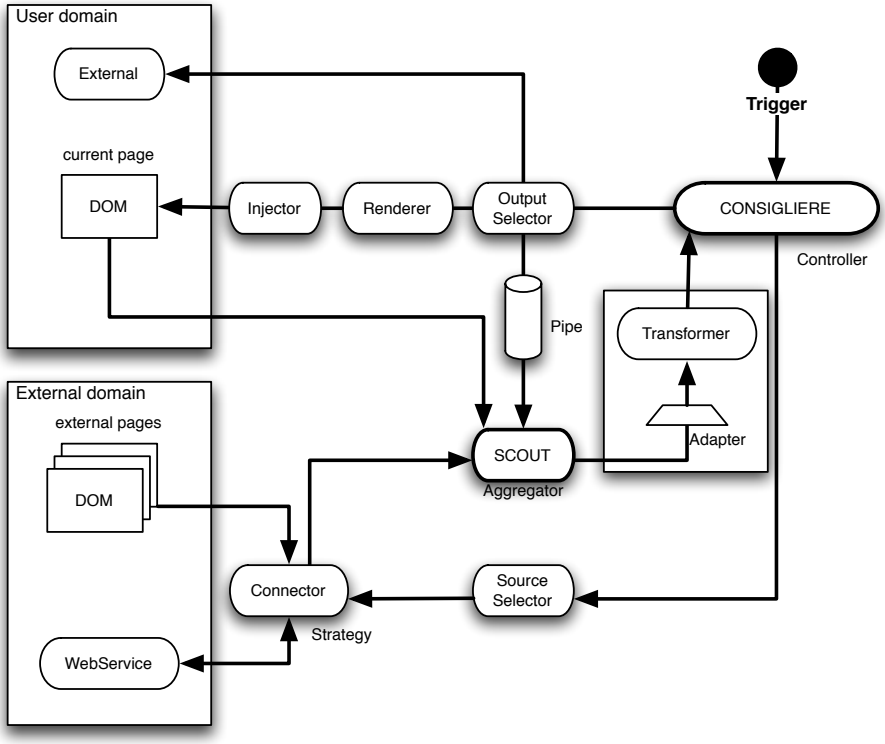

Figure 7.1: Syndicate components

The Oxford dictionary describes the term Syndicate as: "A group of individuals or organizations combined to promote some common interests" [93]. In this context the term is used for two purposes: First, it

is the name of the project and covers the generic principle of several small tasks working together to address a common goal. Second, when used as an adverb, it describes the state of a loaded web page which becomes responsive to Mission triggers as soon as the page has been syndicated.

7.1.1 Syndicator and page states

A Syndicator acts as an enabler and is a term used for an element that allows to syndicate a page. Once a syndicator has been activated the state of a loaded web page changes from *unknown* to *syndicated* as shown in figure 7.2. In order to use a syndicator there is no need of software installations or browser plugins to be loaded.

Once a page has been syndicated the page becomes reactive and Mission triggers are activated and static Missions executed. *Core* is the central state after static Missions are completed. If the static Missions have not been altered, a web page always appears in the same way after syndication at this stage. From that point on the appearance and behavior of a web page depends on user selected Missions.

7.1.2 Syndicate server

A Syndicate server is an external site used as Mission storage with possibilities to manipulate Mission properties. Personalized access allows an individual selection and configuration of Missions. Selected Missions are loaded from the Syndicate server as soon as a page has been syndicated. Once Missions have been loaded onto the user's side, further usage of the Syndicate server depends on the Missions itself. It is possible to continue with syndicated pages without involving the Syndicate server anymore.

Figure 7.3 shows the weaving of a web browser, the syndicate server and connections to the web, seen as a global service-oriented architecture (SOA). Certain services on the web are directly contacted during the visit of a web site while others use a server detour for that purpose. The dotted components and connections become available with the syndication of a page.

Missions, published to the servers repository can make use of services found on the web. It is up to the user to make a selection out of the available Missions. Once a Mission has been loaded and activated, other services may get consulted in order to fulfill a certain task.

The selection and basic configuration of Mission properties happens at the syndicate server while the actual Mission activation is started at the

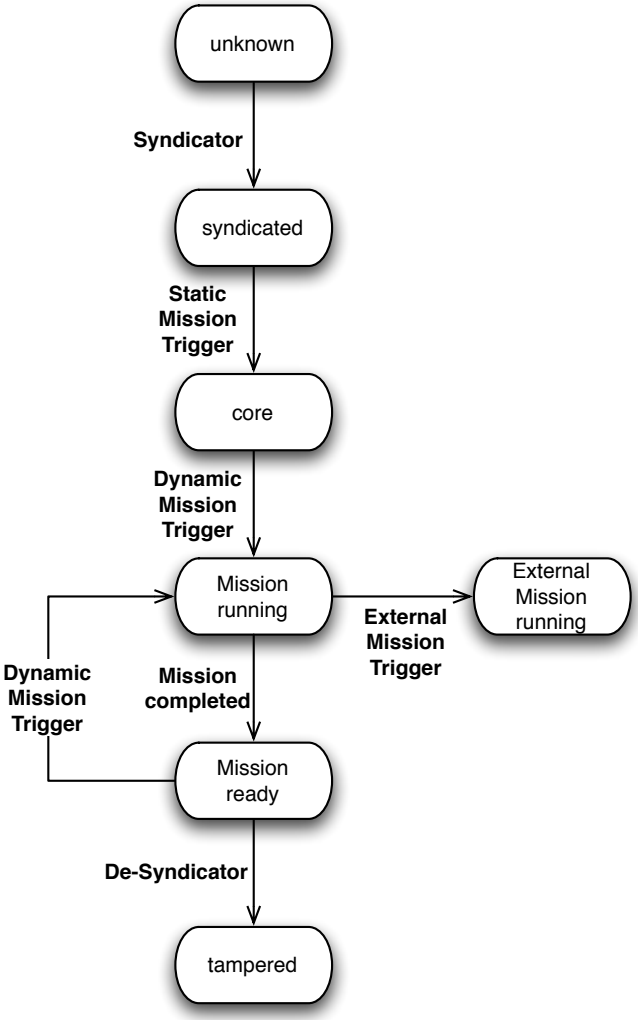

Figure 7.2: States of a web page

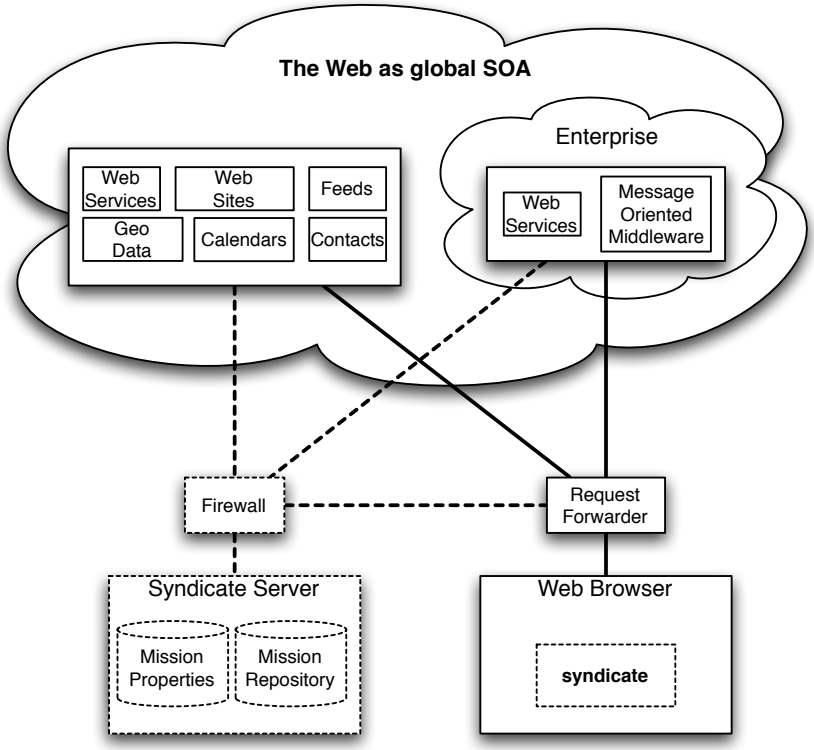

Figure 7.3: Syndicate Client/Server

client's web browser.

7.1.3 Mission

A Mission is a term used to describe a specific task. A task can be a simple modification of the current loaded page or can be composed from invocations of several services before altering the actual page. Missions differ from services in the way that Missions can make use of services but provide additional functionality beside the simple invocation of a service. Missions are not limited to the web browser but can also hand over part of the job to other applications and invoke external programs.

Authorized users create, store and publish Missions on a Syndicate server. It is then up to the user to make an individual selection from the published Missions. Once a Mission has been loaded onto the users side, a trigger activates the execution of the Mission.

According to the behavior of a Mission, a Mission is categorized by one of three patterns: Presentation Missions, Content Missions, and Integrating Missions. Each pattern specifies the major operating range of the Mission. The distinction of the patterns is closely related to the number of the involved external sites. The relation between the Mission patterns is shown in figure 7.4 and patterns located in the outer boxes can always include patterns from the inner ones. At the same time complexity of the Mission increases starting from the innermost ones.

7.1.4 Mission Patterns

Presentation Mission

Presentation Missions are used to change a web site's appearance. Rearranging web content, changing element sizes, fonts or colors or introducing new functionality to enhance the possibilities of a web site are examples of this kind. Enhancements in this pattern only include changes to the visual component of a web site without consulting other web services. Such enhancements can extend interaction components such as formulas or static elements such as images, hyperlinks or table structured data for example.

Content Mission

This pattern describes Missions that make use of external services or information providers to add additional functionality to a web site.

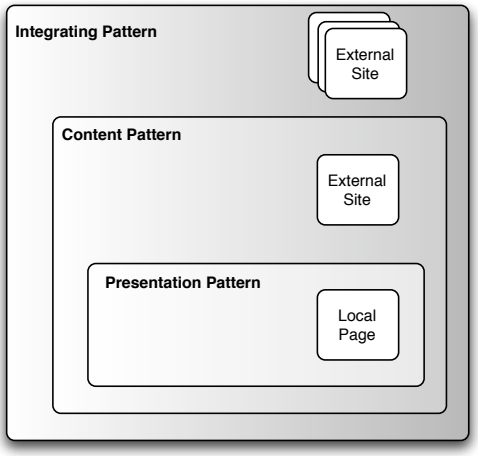

Figure 7.4: Relation of Mission patterns

Missions that implement this pattern make use of mechanisms to contact external sites which are or are not aware of being used as an information provider. Missions implementing this pattern combine data from the current loaded web page with data coming from an external service.

Integrating Mission

The main point of Missions belonging to this group is the fact, that they integrate and combine more than one external site in order to fulfill their task. The focus of these Missions lies on the integration and controlling of the information providers. Since integrating Missions can also include Mission from the other two patterns, their complexity outweighs the one of the other patterns.

7.1.5 Mission triggers

A Mission trigger is an event that starts the invocation of a Mission. This can be a static, periodical, or user-generated event. An event is said to be static if it depends only on the completion of web page loading but not on user interaction. Periodical events are triggered by an internal timer and user-generated events can come from mouse movements or keystrokes.

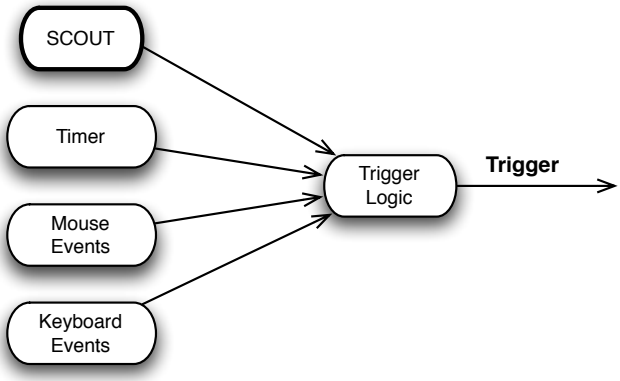

Figure 7.5: Trigger Event

A trigger could also combine user events with scraping information coming from the Scout in a way that an action is only initiated when a user generates an event while the mouse pointer is at a specific position.

7.1.6 Properties

GlsplMissionProperty are environmental settings affecting a specific Mission. Properties are divided in user- and global wide values which depend on the Mission, and properties changing the behavior of Syndicate in general.

Mission specific properties

Each Mission has specific settings. Global values control the general functionality of the Mission. Service addresses, data formats and authorization information are defined globally and are essential for a Mission to work properly. The Source Selector is also configured at a global basis and defines the strategy on how service providers are selected to fulfill a Mission.

User properties allow variations of Mission behavior and can be set once a page has been syndicated. These properties are Mission specific and control the way a Mission is working. Settings can change the process of information gathering with the Scout, the transformation task and the way data is injected into the current page.

Syndicate properties

Properties at this level affect the way syndicate is working. Syndicate can be configured to store the user properties and reload the settings the next time a Mission is activated. Syndicate can also be setup in a way that the syndication of a page happens automatically once a hyperlink referring to another page has been clicked. Once a page has been syndicated, all follow-up pages will be syndicated as well in this mode.

7.2 Service unit

7.2.1 Service

In the context of Syndicate the term *service* embraces all resources available through the internet which offer any form of information. This includes web services with public available access as well as other external web sites or web pages without defined access mechanisms. If a web site does not offer a programming interface to access its information, particular technologies are in place to extract relevant data. Often, a server detour is necessary to gain access to sites that do not offer a public access to the desired data.

The usage of the term *service* in this context differs from web services as described in the architecture description of the W3C [94] in the way, that services are not limited to a particular message protocol as proposed by the working group.

7.2.2 Server detour

A server detour indicates an additional roundtrip of a communication path starting at the user's web browser. This effort of contacting a server on the way to the destination information provider justifies itself because of several reasons:

Security: The server does not necessary need to be located at an external site but can provide an additional layer of security because of central management by experienced administrators.

Authentication to other servers or accessing several other services to fulfill a Mission is made easier on the server-side.

Scalability: The server can distribute the load to several other servers, each server serving its own application area. Beside distribution

the server can also offload the server by caching content or using compression techniques.

Performance buffer: The Syndicate server can serve as a buffer between the client and other web sites and can provide default or cached information to the client. The server may also decide to break down large amount of data and send only portion of the data back to the client.

If requests often arrive in set of bursts, the Syndicate server can smooth out the bursts by using a buffer to reduce transaction load.

Intranet publishing: If the server is located within the intranet, access to an organization's internal data could also be provided without offering the material to the rest of the world.

Richer development environment: Server-side development simplifies the access to other web servers. Different libraries support the processing of requests.

Support client-side development: It is generally easier to transform different data formats into a format more desirable by the client such as JSON instead of SOAP.

Reliability buffer: In choosing alternative sites if a desired server is unreachable, the Syndicate server can offer more reliability.

Quality buffer: The source selector allows the selection of services based on quality priority.

Better information: The server can respond with different status- or error codes and provide more information about performance, reliability, quality or failure about Missions.

Adaption: Requested Services usually do not respond with a data format or functionality that is exactly in the desired format. By using adaptors at the server-side, content can be brought into the necessary format.

7.3 Transformer unit

7.3.1 Adapter

The channel adapter can change the data format coming from the Scout to an appropriate format for the transformer. The purpose of the adapter is to hide heterogeneity and provide a uniform view of the extracted data. The adapter depends on the transformer which is Mission specific.

7.3.2 Transformer

Transformers are the main area where the functionality and the business logic of a Mission is kept. The transformer can use information from various sources and produces an output which can further be processed. Transformers usually enrich or combine the input data with other information and produce an output with additional functionality. It is the responsibility of the Scout to deliver such information to be used as input data. But it is also possible that the transformer changes already processed information without further invocation of the Scout.

The number of involved information suppliers is directly related to the pattern of a Mission. A Mission based on the presentation pattern uses a transformer that does not make use of information coming from external services. The Content and the Integration pattern do require additional data resources.

The output of the Transformer can consist of a modification of the original web page or can be a complete replacement. It is also possible to hand the output and controlling over to external programs running outside of the web browser.

7.4 Exploration unit

7.4.1 Scout

The information Scout is responsible for the aggregation of input data. The source of the input data depends on property values globally set by the Mission or on user-specific setting.

The aggregation of input data is based on one of two strategies: Semantic and structural scraping. Both strategies depend on an analysis of text based input.

Semantic scraping

Semantic scraping covers the area where information has to be processed with knowledge about the content of the input data. The output of the processing can be a subset of the input data, where only relevant information is kept. But the output can also be a new description about the analyzed data without direct structural relation to the input.

In order to come up with a semantic description of the input data, external services can be consulted. Methods such as statistic evaluations can improve the quality of the description, but at the same time a consistent output cannot be guaranteed.

Structural scraping

Structural scraping defines the analysis of the composition of a web page. In contrary to semantic scraping, this information aggregation process does not depend on knowledge about meaningful content. But as within semantic scraping, assumptions and restrictions limit the use to a particular type of web pages. In particular web pages which do not use html as the basis for their content need another processing schema.

The output of the structural scraping process is an eventually rearranged subset of the input data. User actions such as a text selection can influence the functioning of the scraping process.

7.5 Data access unit

7.5.1 Source selector

Source selection for the Scout is based on a strategy pattern. The input source is selected or chosen by priority depending on different criteria. Such criteria include the origin of the resource based on quality, speed or liability. Input sources can also be weighted and put into a priority queue depending on a particular criteria. The use of an input source is transparent in the process flow.

Enabling criteria

Whether a loaded web page allows to be altered by a Mission depends on the enabling criteria. This criteria is a list of web addresses or wild cards of web addresses that specify which web pages are permitted. This

criteria is used to restrict Missions that only make sense on particular web pages to the ones that are built for that purpose.

Quality criteria

Resources can be weighted by a quality criteria. *Response time* is an example of an objective criteria where sites with faster response times are favored against slower ones. *Information quality* is a user defined criteria which cannot be measured directly and requires a graduation by the user.

Best of breed criteria

Best of breed is a strategy where external ratings are used to built a priority list of otherwise equivalent service providers. In order to access ratings it can be necessary to make use of a Mission where the only purpose is to lookup ratings.

Additional functionality

By the use of a Source selector, input sources are decoupled from the rest of the application. This approach leads to greater flexibility in choosing alternative input resources. Decoupling at the same time allows to change physical protocols used for communication or insert logging functionality at the communication channel.

7.5.2 Connector

Unit that provides methods to connect to external services. External services can make use of different mechanisms and require individual access. The Connector gives a unified view onto these service interfaces.

The Connector is linked to the Scout which processes the content of the received data. The Connector itself is only responsible for the connection but not for the analyzation of the content.

7.6 Controlling unit

The Consigliere is the nerve centre of the Syndicate system and acts as controlling unit between the single components. All actions of the Syndicate system are coordinated by the Consigliere and are running at

the client side. The processing sequence of the Consigliere is immutable but the behavior of Missions are modifiable via Mission properties. A local Consigliere runs at the user's web browser and a remote Consigliere can run at the Syndicate server if contacted.

7.7 Output viewer

7.7.1 Renderer

The renderer addresses the presentation of the outcome of a Mission. The behavior of the renderer is given with default implementations but can be changed depending on a specific Mission. The renderer works independently of the other components and relies on capabilities of the output device.

7.7.2 Injector

A component, that manipulates the current loaded web page. This includes opening a new page, appending content to the current page or replacing parts or everything of the current content. The Injector can also manipulate the current page in a way that new content appears only based on user actions such as selecting a particular component or moving the mouse over a specific part of the page.

The injector works independent of the other components but often a logical connection holds.

The necessary information for the Injector are based on the Scout and the result coming from the Transformer. The Scout provides pointers to locations where data parts have been found and the Transformer offers the data to be injected.

7.7.3 DOM

The Document Object Model (DOM) describes a standard object model for representing HTML, XML or related formats. The World Wide Web Consortium (W3C) developed the DOM in response to the development of various proprietary models for HTML, particularly those used in web browsers. The DOM specification is platform- and language independent and constantly being revised by the W3C, with browsers at the same time constantly trying to support the latest recommended version of the DOM.

7.7.4 External programs

Syndication is not limited to the web browser. Beside modifications of the loaded web page, launching of external programs is also possible. Information from the Scout can be used to control the program while startup or affect the way the program is initialized. Once the control has been handed to an external program it is out of reach of the syndication environment.

8 Syndicate Control Flow

Figure 7.1 on page 71 shows the connections between the different components of syndicate. The group of elements emphasized in figure 8.1 represent input and output resources of the syndication process.

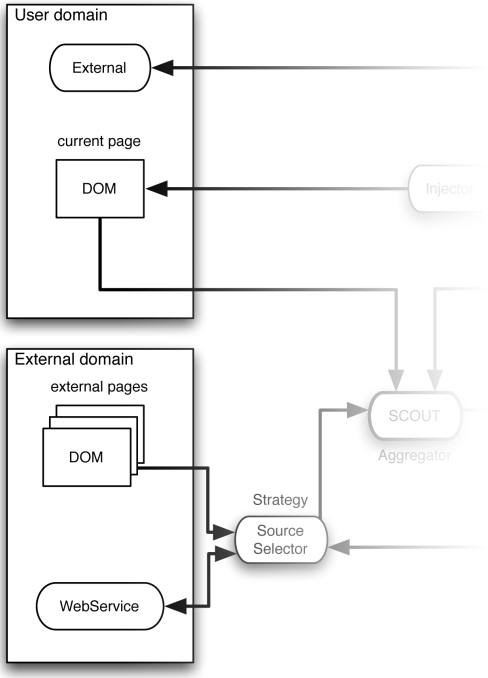

Figure 8.1: Input and output resources

The grouped elements have a logical connection since they are all resources used for input and output of Missions. But they are physically separated and generally live on different machines. The current page is the starting point of the syndication process and other resources such as external pages or web-based services are contacted while a Mission is

running.

If an external component from the user's domain is started up, Syndicate looses control and only initial parameters are passed to the external component.

8.1 Syndication of a page

The action diagram shown in figure 8.2 describes the control flow starting with loading of a web page.

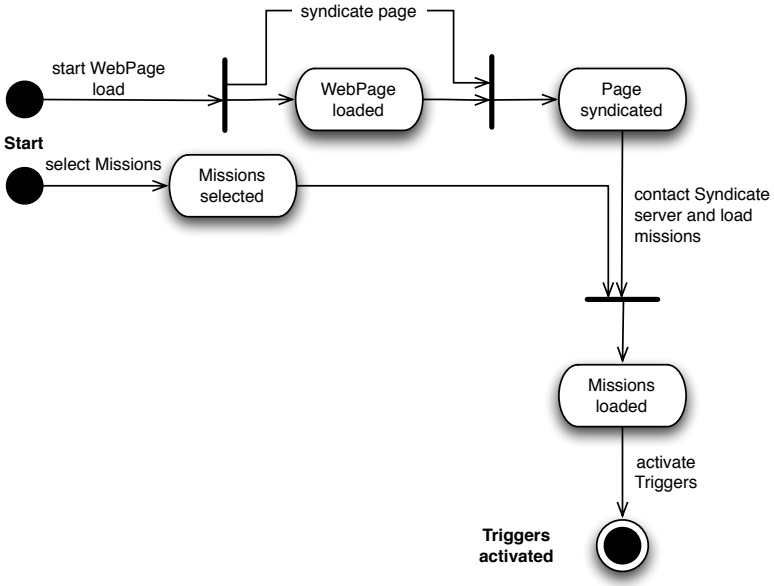

Figure 8.2: Page syndication

Syndicate starts its activities after a web page has been loaded and a Syndicator transformed the web page into a syndicated page. During the syndication process, the Syndicate server will be contacted and subscribed Missions are loaded into the current web page. The activation of the Syndicator is initiated by the user and requires a connection to a Syndicate server. Up to this point the loaded page has not been altered but all necessary Mission components have been loaded onto the user's

side. If the Syndicate server is furthermore contacted depends on the functionality and activation of the loaded Missions.

8.2 Mission triggering

Whenever a Mission trigger event has been received, the correspondent Mission starts up. A trigger event can origin from a user activity such as a key stroke or mouse gesture or can be established by a system event. A system event includes periodic triggers which occur at regular intervals and static events which are produced when a page has been loaded completely. With the activation of a Trigger the preparation of a Mission begins as shown in figure 8.3

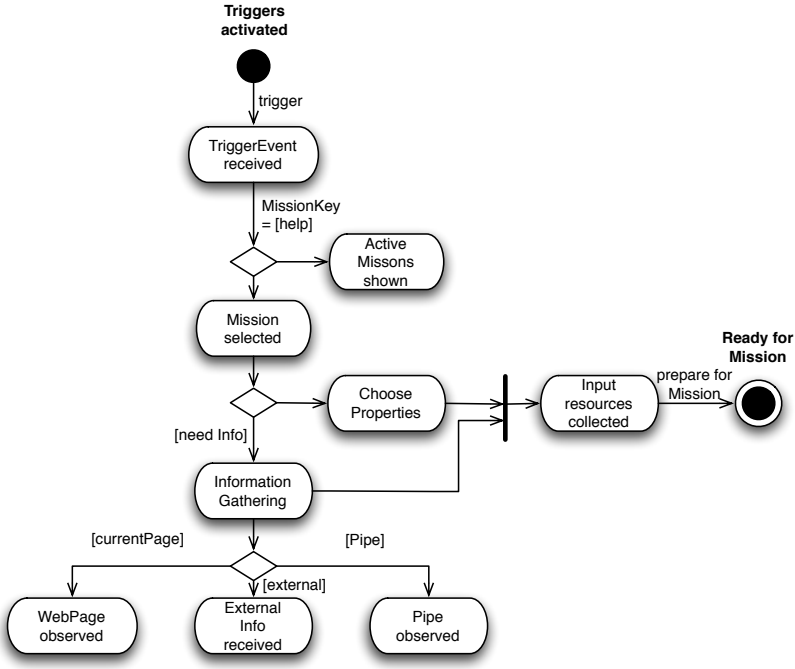

Figure 8.3: Initializing of a Mission

If the Mission needs additional information, it is the responsibility of the Scout to gather necessary data.

8.3 Mission activation

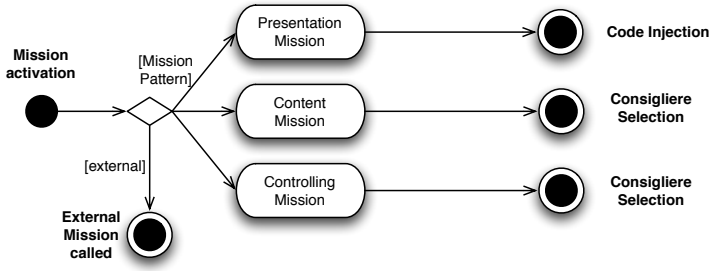

Figure 8.4: Mission activation

When a Mission has been activated the collected input sources are processed further either by the local Consigliere or by the Consigliere at the Syndicate server. The decision about the addressed component depends on the Mission pattern and complexity. If the server is involved, the process is named a server detour.

The invocation of the remote Consigliere includes a service selection process where service partners are selected based on different strategies as described in 7.5.1 and shown in figure 8.5. Whether the remote Consigliere has been involved or not, after the interaction with a remote service the Scout is in charge of further processing.

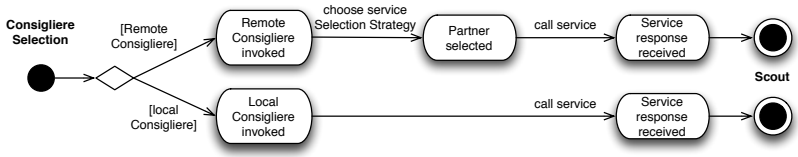

Figure 8.5: Consigliere Selection

8.3.1 Local Consigliere

The local Consigliere is directly consulted for presentation Missions or simple content Missions which do not need complex transformations of resource data. If the control has been handed to the remote Consigliere

to complete a Mission's task, the final processing is still covered by the local Consigliere. This processing consists of the coordination of the Scout and the render and injection action.

8.3.2 Remote Consigliere via server detour

The contacted remote Consigliere selects its service source by the Source Selector based on a specific strategy as explained in section 7.5.1. Once the response has received and eventually adapted for further handling, the Scout takes care of information gathering.

8.3.3 Information gathering with the Scout

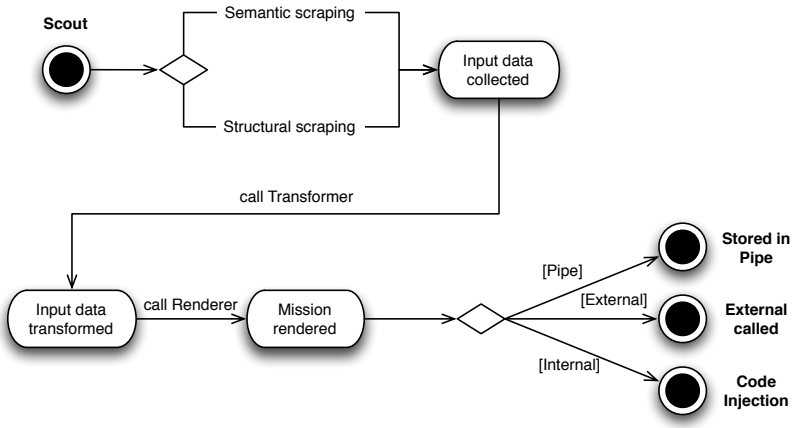

Figure 8.6: Invocation of the Scout

The Scout is responsible for information gathering. Data can be extracted from the current loaded page, from external web pages or from the output of web-based services or from other Missions. Semantic or structural scraping techniques as described in section 13.3 are used to find the desired information.

8.3.4 Transformation and Rendering

The main goal of a Mission is the transformation of one or more input resources into a new structure. The transformed data is rendered by the

Mission renderer and passed to one of three possible output sections as shown in table 8.1.

Mission output	Handler
A pipe can be used as information storage if further processing by other Missions is requested.	Consigliere
An interface for calling external programs offers the possibility to hand the responsibility over to another application.	External Application
Code injection allows transformed data to be placed into the current loaded web page.	Injector

Table 8.1: Handlers for mission output

8.3.5 Injection

The injector takes rendered, transformed Mission data and puts them into the current loaded page. The data can either be appended to the page or be used as a replacement for an area marked during the processing of the Scout. With the finishing of the Injector a Missions activity is completed.

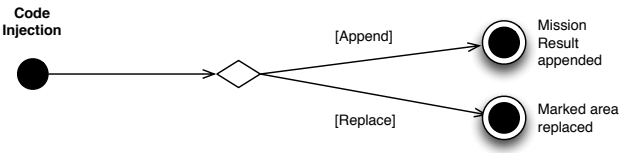

Figure 8.7: Injection

Part III

Syndicate as seen by the User

9 Mission Publisher

This chapter describes tools and interfaces from a user perspective depending on the role the user takes on. Examples of a workflow from creation, accessing, using, editing and removing are given at the end of this part.

9.1 Mission components

Publishing of new Missions is restricted to authorized Users (see 6.3). A Mission to be published consists of a Mission description, the Mission implementation, Property settings, and Resource addresses.

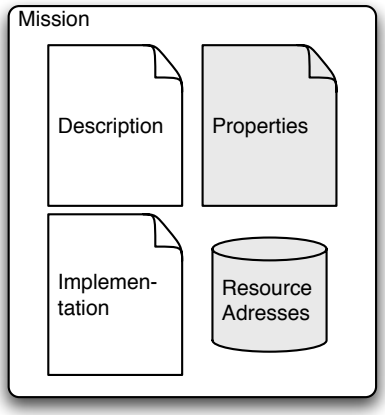

Figure 9.1: Mission parts

9.1.1 Summary

In order to gain information about a Mission's purpose a summary has to be included with each Mission. A summary consists of a Mission pattern

(1), a category (2), an image (3), a textual description (4), and the Missions function (5). Beside these explicitly defined elements, Mission property settings are also contributing to the Mission's summary.

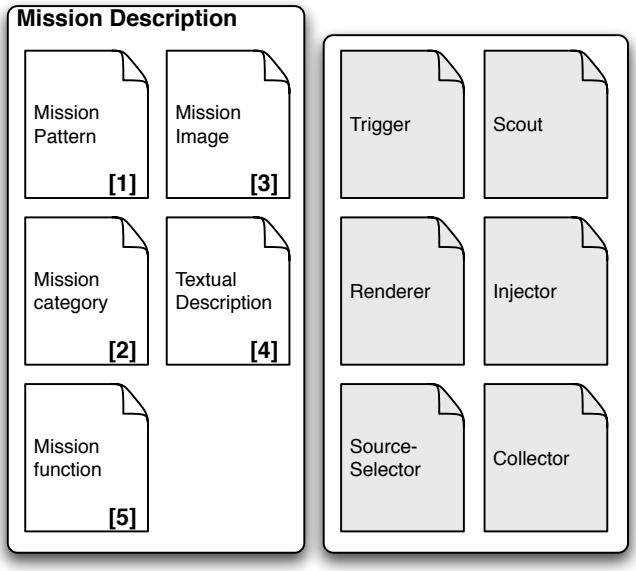

Figure 9.2: Mission summary

The Mission patterns (1) is one keyword out of 'Presentation', 'Content' and 'Integration' and is used to categorize the Mission's purpose but has only descriptive character. The category (2) of the Mission is a tag list of keywords further describing the Mission's purpose. The image (3) is a visual representation of the Mission and together with the textual description (4) serves to select a suitable Mission. Finally, the Mission function (5) explains what it actually does, for example modifying particular elements on the page.

9.1.2 Implementation

The implementation part of a Mission consists of JavaScript code that executes on Mission triggering. This code includes specific activities of the Scout, the Mission's actions and integration code that eventually

combines output from other Missions.

9.1.3 Mission Properties

Mission properties are a set of environmental settings and are initiated with predefined values. The Mission publisher can decide if a subscribed user is allowed to change a particular value.

Trigger properties

An example of a setting that is very likely to be changed is the Mission trigger. The Mission trigger is responsible for the activation of a Mission and can individually be defined by a Mission subscriber. The Mission publisher chooses possible choices for Mission triggers settings, sets up a default environment and defines which setting can be changed by a Mission subscriber.

Trigger settings can be chosen from the list presented in table 9.1. Some of the settings require additional values which can be replaced by a Mission subscriber.

Trigger	Explanation
static	Static triggers fire as soon as the page has been loaded.
periodical	A trigger that periodically invokes the Mission. This trigger requires the setting of an internal timer between each invocation.
user-generated	A trigger that starts firing dependent on user key strokes or mouse movements. This trigger requires the setting of a key combination or a selection of a mouse gesture.

Table 9.1: Trigger settings

Triggers can use scraping information coming from the Scout to decide if a trigger fires or not. For example a trigger can be defined to fire whenever the mouse pointer is located on a table structured text and a specific key has been pressed and released.

Conflicts with Mission trigger settings from other Missions are recognized by the Syndicate server and a change is requested in case of a

conflict. Trigger settings can always be altered by subscribing user and the Mission publisher cannot disallow a change.

Source Selector

The Source Selector is responsible for choosing an appropriate resource for a specific Mission. Settings for the Source Selector include a sorted list of resource addresses and a selection strategy. The selection strategy is a choice of the two possibilities shown in table 9.2 together with a weighting definition.

Criteria	Explanation
Enabling criteria	The enabling criteria consist of a list of possible web addresses used to decide if a loaded web page is allowed to be altered by a specific Mission.
Quality criteria	Response Time is a predefined possibility where equivalent resources are compared to each other depending on the time needed to process a request. Information quality is another criteria where the Mission publisher can rate resource addresses by rearranging the address list.
Best of breed criteria	This strategy uses external ratings to select a resource. Such policies allow complex configurations and have to be configured specifically for each Mission.

Table 9.2: Source Selector settings

In order to increase the chance of a successful Mission completion, test cases need to be defined where failing resources can be eliminated in advance to a Mission activation. Due to the nature of anonymous service integration a 100% guarantee cannot be given but test cases increase prospering.

Collector

Collector settings comprehend of environments necessary to complete requests to the defined resources. Such an environment consist of authorization information, protocol specific definitions and parameters used to involve external services.

Scout

Properties for the Scout depend on the selected Scout strategy as shown in table 9.3. Properties for both strategies are given as a set of instructions to extract the desired information out of a given input.

Strategy	Explanation
Semantic scraping	Semantic scraping requires the specification of a selection method. Such a method often includes textural search patterns such as the definition of a specific number format.
Structural scraping	This scraping mechanism uses simpler definitions and is based on items used to build a web site.

Table 9.3: Scout strategies

Renderer

The settings for the Renderer can be altered by changing style definitions used to specify the presentation of a Missions outcome.

Injector

Injector properties decide where the outcome of a Mission goes to. Table 9.4 shows possible locations for a Missions result and lists corresponding property values.

9.2 Mission Unpublishing

If a Mission has to be unpublished, the Syndicate server controls user subscriptions and requires additional confirmation of the unpublishing

Location	Explanation
Separate	A new window opens with the result of a Mission.
Append	The output of the Mission is appended to the current loaded web page. A Scout definition might be necessary in order to define the exact location if the data is to be inserted within the page rather than appended to the end of the page.
Replace	This selection requires a Scout definition of the area where the resulting data replaces existing information.

Table 9.4: Injector locations

process in case of an active Mission. Unpublishing removes the entire specifications of a Mission as well as all user settings that belong to the Mission.

9.3 Publishing workflow

A Mission developer sets up the environment in order to allow a Mission to function properly. The Mission publisher has the ability to alter those definition and specify which properties are allowed to be changed by subscribing users. Once a Mission has been published to the Syndicate server it is up to the users to choose, alter and invoke specific Missions. The Mission publisher must guarantee a successful setup but it cannot be responsible for altered settings that do not work acceptable.

10 Mission Subscriber

10.1 Mission subscription

Missions are stored on a Syndicate server and can be subscribed by users. Once a Mission has been selected, Mission properties need to be defined. If a specific property is allowed to be altered depends on the settings given by the Mission publisher as described in section 9.1.3. User defined settings are stored at the Syndicate server and are removed when a user unsubscribes from a Mission or the Mission has been unpublished by a Mission publisher.

A required property is the selection of a Mission trigger used for invocation of the Mission. This setting can eventually be forced by the Syndicate server if the default setup conflicts with other Missions. Other properties such as resource selection or render configurations depend on the Mission and can eventually be altered. The Syndicate server can assist user settings such as resource addresses and verify the existence of remote sites, but it cannot prevent constellations that prevent functional Missions.

If a Mission has been subscribed it is only marked to be included but is still stored at the server side and loaded whenever a page has been syndicated. The actual processing of the Mission relies on the firing of the Mission trigger.

10.2 Mission selection

In order to chose a Mission, different search criteria can support the decision process. Some Missions are only built for a small set of web sites while other Missions work generally. Table 10.1 gives an overview about useful selection criteria. The criteria are additively used to narrow the presented Missions to an interested subset.

Mission category	– A list of keywords that are used to demarcate Missions by operating area.
Mission behavior	– A list of keywords that separate Missions according to their behavior.
Mission pattern	– The list of Missions can be limited to a given Mission pattern.
Mission description	– A textual search in the Mission descriptions can be used to isolate interesting Missions.
Web address	– This criteria filters Missions that work for a given web address.
Included services	– Another search criteria to localize Missions that include selective service addresses.
Trigger settings	– By choosing trigger settings, Missions can be filtered out that only respond to given activities.

Table 10.1: Mission selection criteria

10.3 Mission unsubscription

If a previously subscribed Mission is deselected at the Syndicate server, it will not be included in further page syndication. Individual settings for Mission properties will be removed and locked trigger combinations are released for other Missions. The definitions of other Missions are not affected.

11 Examples

This chapter provides illustrations from the user point of view and shows possibilities of Syndicate. The stated examples are grouped into three categories according to the proposed Mission patterns, namely content pattern, presentation and integration pattern (see: 6.1) and increase in complexity in terms of the number of involved components.

Table 11.1 on page 102 gives an overview of the presented examples and their participating components. One example in each category is explained in detail while the other descriptions are limited to a functional explanation.

		Trigger		Scout		Source Selector		Communication				Injector		
	Example	Static	User	Semantic Scraping	Structural Scraping	Proxy	Alternatives	REST	SOAP	JSON-RPC	XML-RPC	Ext.	Append	Replace
Presentation	AdRemover	•			•	–	–	–	–	–	–			•
	TextFieldSize	•			•	–	–	–	–	–	–			•
	TableSort		•		•	–	–	–	–	–	–			•
	FileExtension	•			•	–	–	–	–	–	–			•
	iTunes		•		•	–	–	–	–	–	–	•		
	Stocks		•		•	–	–	–	–	–	–	•		
Content	Question	•				•		•					•	
	AcronymList	•		•		•	•	•					•	
	BookFinder		•		•	•	•	•					•	
	Distance		•		•	•		•						•
	Dictionary		•		•	•				•			•	
	Weather		•		•	•					•		•	
Integration	TextSpeaker		•		•	•	•	•				•		
	Translator		•	•	•		•		•					•
	PhoneCall	•	•	•			•	•				•		
	AskSmart	•	•	•			•	•					•	
	DiscMission	•	•	•			•	•					•	

Table 11.1: Syndicate examples

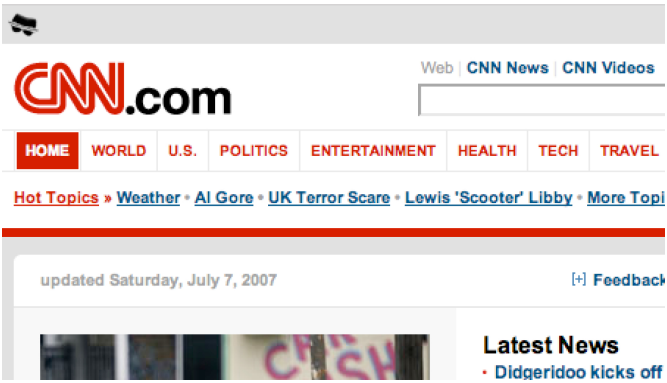

Figure 11.1: A syndicated page

All Examples provided in this chapter are not restricted to certain web sites. The presented web sites have been chose because certain aspects could be demonstrated effectively but all Missions should work on any other site as well. If a web site has been syndicated, a small icon appears on the top of the page as shown in figure 11.1.

11.1 Presentation Mission examples

Mission examples of the Presentation pattern change the appearance or behavior of a syndicated page. These Missions do not use other services or external information for their task. But in order to activate a Mission, some of the examples do make use of user actions to trigger a Mission.

11.1.1 AdRemover

Functional description

This Mission removes advertisements and pop-up's from a syndicated page by making corresponding blocks invisible. The Mission does its job without any user interaction and automatically starts removing advertisement blocks with the syndication of a page. This Mission profits from an ongoing update to recognize new advertisement blocks. This is a good example why it does make sense to reload the Mission with each page load.

11.1.2 TextFieldSize

Functional description

This Mission starts as soon as a page has been syndicated and modifies all text input fields found on the page. Every text field becomes resizable by the addition of a resizable icon in the lower right corner of the text field. Single text fields can be expanded or shrunk to the side and text areas can be resized in their widths and heights. When the user moves

Figure 11.2: Resizable text fields

the mouse pointer close to the resize icon, the mouse courser changes from the standard image to a resize pointer. After clicking and dragging the resize icon, the text field instantly changes its size. Resizing a text field can affect the page layout especially when a web page has been poorly designed and uses absolute numbers to define its layout.

Component based description

After a page has been syndicated the static trigger starts the *TextFieldSize* Mission. The Consigliere, responsible for the coordination of the Mission flow, activates the Scout. In this example the Scout uses structural scraping to find all relevant user interface components and collects them in a list. Since this Mission does not make use of external services, the Source Selector is not involved and the Consigliere directly calls the Renderer for each user interface component from the list. The Renderer creates a modified version of the text field and installs an event handler that takes care of applicable mouse gestures. Finally the Injector replaces the existing text fields with the freshly created ones.

Whenever the user moves the mouse pointer on top of the resize icon, the event handler becomes active and modifies the text field size as soon as the user presses the mouse button and drags the icon around. This example is based on a user script for GreaseMonkey [95] and serves as an example on how to make use of related projects.

11.1.3 TableSorter

Functional description

The TableSorter allows to sort table content by clicking on a text in the table header row. The user has to select a table manually and initiate the sorting possibility with a keystroke. From that point on the table content can be sorted by clicking on the hyperlinked text in the header row. Subsequent clicks toggle the order direction and are marked with a small arrow to the right. The sorting algorithm recognizes simple date formats, decimal numbers and US dollar currencies.

This Mission is a modification of a user script, available for GreaseMonkey [96].

11.1.4 FileExtension

Functional description

After syndication of a page, all hyperlinks that appear on the page are analyzed and extended with a small icon showing the file type to be downloaded. For example Microsoft Word documents have a small icon showing the logo of a Word document. It is also possible to exclude links from the list. Hyper links to zipped files for example, could be removed because of security reasons.

This Mission is an example of a modification from a related project as found in Chickenfoot [72].

11.1.5 iTunes

The next two examples differ from the ones before that an external program is launched. As soon as the Mission has been activated, control is taken over by the external application and the Mission is completed.

Functional description

This Mission starts a lookup for music albums containing the current selected text in their album name in the iTunes music database. The external program iTunes is started for that purpose and a connection to the music store is made which handles the control from the web browser to the iTunes application.

11.1.6 Stocks

Functional description

The currently selected text is used as a search string for stock values. This Mission launches the Sherlock application to look up quotes and provides charts and stock values. Chart selection and modifications are all made in the external application. The Mission completes as soon as the application has been started.

11.2 Content Mission examples

The following examples make use of external information providers with or without published web interfaces. Different communication technologies are in place, mostly following the REST architectural style as described in section 3.4.

11.2.1 Questions

Functional description

If this Mission has been activated a question will show up with the syndication of a page. The question is randomly selected from a question database and can be answered independently of the rest of the page.

11.2.2 Acronyms

Functional description

This Mission selects acronyms from the current page and appends a list of descriptions for them to the end of the page. Figure 11.3 shows an example with acronyms found on a page from New York Times. This Missions makes use of alternative service providers and shows descriptions from sites according to the user's preference.

Component based description

After page syndication, a static Mission trigger starts analyzing all text on the current loaded page. This activity is done by the Scout which uses semantic scraping to select potential acronyms words. The Consigliere takes the extracted words, sorts them, removes duplicates and puts the

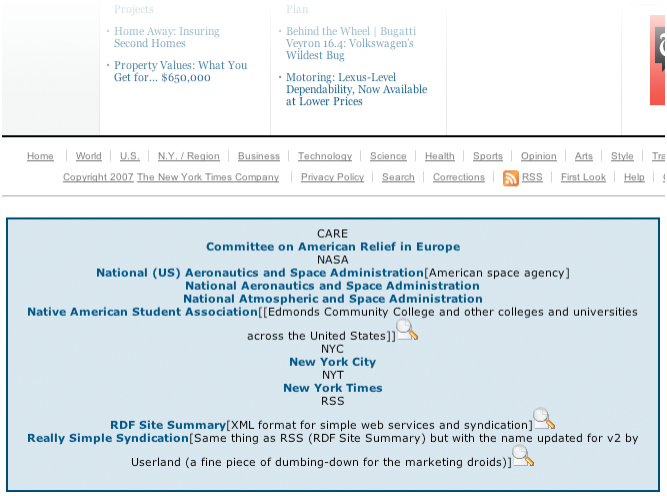

Figure 11.3: List of acronyms

result into a list for further processing. The Source Selector picks service providers according to the user's preference and keeps a reference of each one. The Connector then contacts the selected providers and asks for a description of each possible acronym. The resulting descriptions are passed to the Renderer which builds a visual representation of all the words for which a description could be found. The Injector finally adds the created summary to the end of the page. Figure 11.4 shows the components involved in this Mission. This sketch is a cut out of the concept presented in section 7.1.

11.2.3 BookFinder

Functional description

BookFinder is a Mission where a text selection is used to lookup book titles containing the selected word. External web sites such as Amazon [97] are used to find matching products. A segment of the first books are then concatenated to the end of the page. The Mission is started by a keystroke after the desired text has been selected.

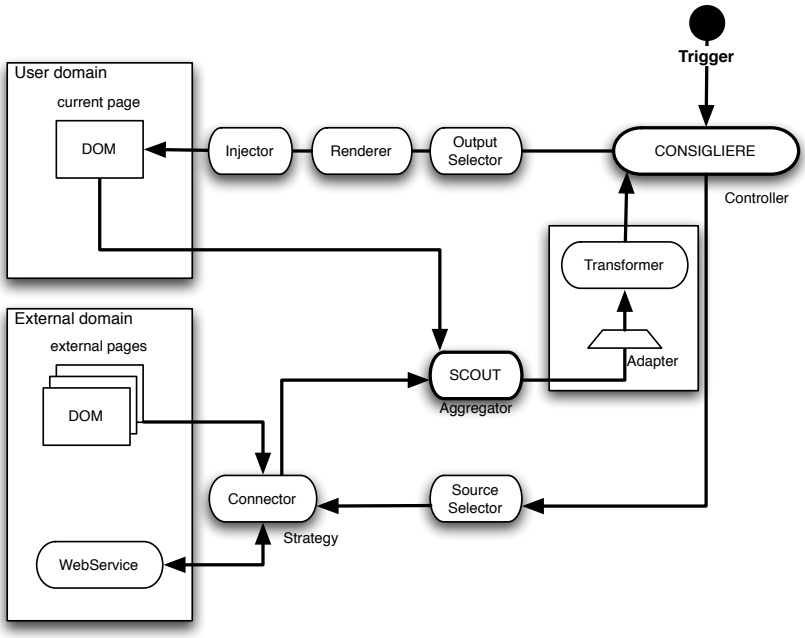

Figure 11.4: Components for Acronyms

11.2.4 Distance

Functional description

This Mission calculates distances from a predefined starting point to a selected address and shows this information in a tooltip window. The starting point is defined once at the Syndicate server and the destination address is chosen by selecting a text area.

11.2.5 Dictionary

Functional description

Dictionary is similar to the *BookFinder* example but in this Mission a selected word is looked up in a dictionary. The consulted description then appears in a small window as long as the user has its mouse pointer positioned on top of the selected word.

11.2.6 Weather

Functional description

If the current text selection contains a name of a city, a region or a country, the latest weather forecast appears in a small window when the Mission is triggered with a keystroke.

11.3 Integration Mission examples

This category contains examples of Missions which use more than one cycle to complete a certain task. The piping mechanism for staggered Missions is also explained.

11.3.1 TextSpeaker

Functional description

This Mission provides the ability to start a synthetic speech for a text selection. The text is chosen either by selecting a text range with the mouse or otherwise the text block which is under the mouse cursor is used. The language of the web page is automatically chosen to use the correct speaking service. Alternative service providers can be defined which are consulted according to the user's priority definition.

11.3.2 Translator

Functional description

A text area under the mouse cursor or a selected text range is translated into a target language defined by the user. The input language is automatically chosen by this Mission. Alternative translation services can be defined where one service is picked based on a priority strategy.

Component based description

After page syndication a static Mission trigger launches a language guessing Mission which tries to figure out in which language the current page has been written. This Mission consults the Scout which starts semantic scraping in order to find this information. The outcome of the Missions is a language string such as "en" for English or "fr" for French.

The outcome data is placed into a pipe for further processing. If the Mission cannot come up with a solution an input field is presented to the user, asking the user for a manual selection of the language for the current page. Figure 11.5 shows the involved components.

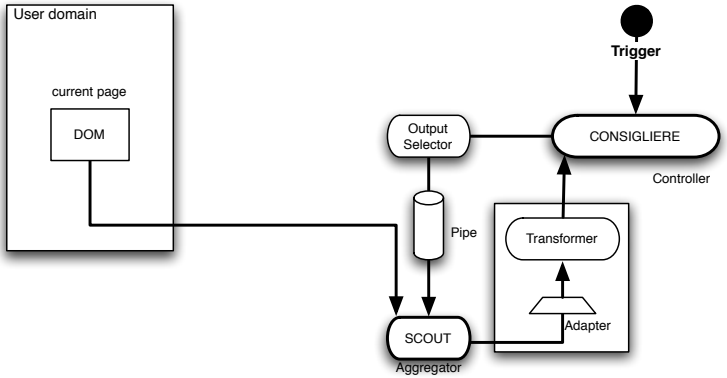

Figure 11.5: Components for language dedection

The user choses a text block by either selecting a text range with the mouse or by positioning the mouse pointer over the desired text area. The *Translation* Mission is started with a keystroke which triggers the Consigliere. The Consigliere communicates with the Source Selector and chooses a service provider out of the list of alternatives. The choice is made based on a priority strategy where each language pair can have a corresponding service provider.

At the same time the Scout analyzes the user selection or the text area at the position of the mouse cursor if no text has been selected. If the mouse position is used to locate a text portion, the affected text area depends on the page structure at that location. The text selection algorithm tries to limit the text area to the surrounding layout structure. The chosen text is then stored in the pipe for further processing.

Next, the Consigliere takes the language information from the pipe and reads the user property about the desired target language. Based on this language pair the source selector selects an appropriate service provider and invokes the Connector. The connector requests a translation for the text stored in the pipe and receives the transformed data. The translated result is passed to the Renderer which directly invokes the

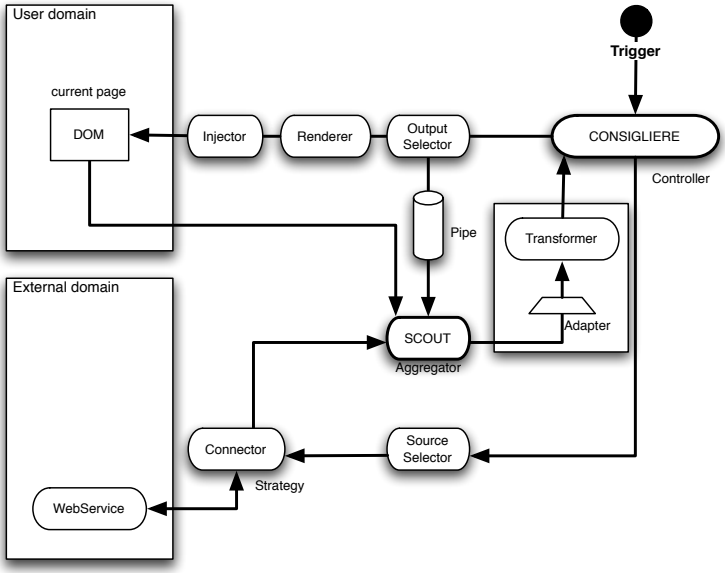

Figure 11.6: Components for Translator

Injector. Finally the Injector replaces the input text with the translated text. Figure 11.6 shows the remaining part of the *Translator* Mission.

11.3.3 PhoneCall

Functional description

This Mission analyzes a syndicated page and tries to extract telephone numbers. Each telephone number is supplemented with a telephone button as shown in figure 11.7.

When the user presses the button a telephone call is initiated either by starting a VOIP connection or by playing a series of dial tones which can be used to place a call with an ordinary telephone.

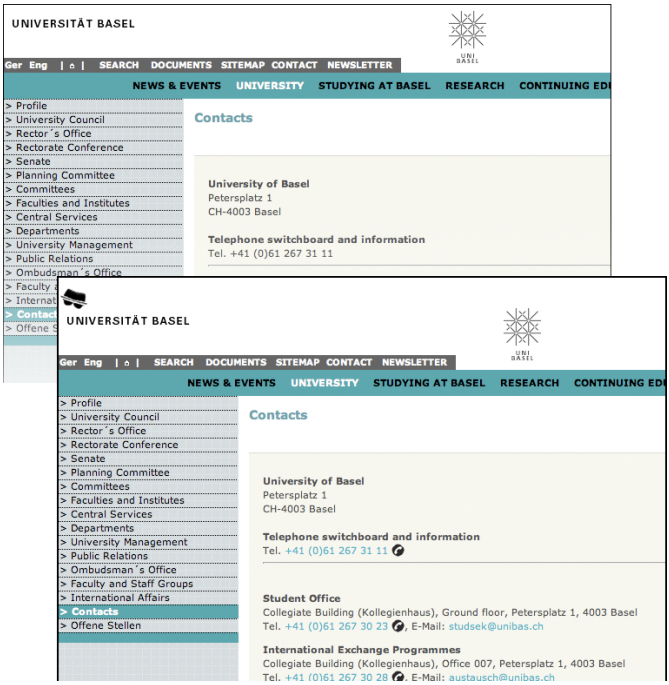

Figure 11.7: Web page with telephone numbers

11.3.4 AskSmart

The following two examples have not been implemented yet, but serve as a good example of the integration pattern.

Functional description

The aim of the *AskSmart* Mission is to embed knowledge questions to a loaded page. These questions can be used as self assessments about the topic of the current page. Standardized databases serve as resource to discover and load predefined questions.

Component based description

At first a *Tag* Mission analyzes the current page and returns a list of keywords that describe the possible content. The output of the *Tag* Mission is used to lookup questions from a set of databases. These databases make use of the IMS Question and Test Interoperability specification (IMS QTI) as proposed by the IMS Global Learning Consortium [98]. The retrieved questions are rendered by an external web-based service [99] [100] and are embedded by the *AskSmart* Mission to the bottom of the current page. These questions are being used as self assessments and are corrected by an external correction service keeping the interaction anonymously [101].

11.3.5 DiscMission

Functional description

The purpose of this Mission is to discover new Missions that might be of interest to the user. Missions are looked up based on two conditions. The first condition is defined by the user who specifies categories or behaviors he is interested in. The second condition is given by another Mission that makes a semantic analysis of the loaded page and tries to find Missions specifically built for this page. If a new Mission has been discovered and has not already been denied by the user, then the Missions description is shown together with a possibility to subscribe.

11.4 Examples of distinguished components

This section uses examples described in this chapter to explain distinguished components with concrete implementations. The purpose of

these descriptions is a better understanding of the operating range of the components.

11.4.1 Triggers

Example: Static trigger

An example of a static Mission triggering is the acronym list as described in section 11.2.2. As soon as the web page has been loaded the Scout is triggered to observe the current page for possible acronyms.

Example: Periodical trigger

An example for periodical updates is the integration of weather data appended to city names found on the current page. Temperatures or the strength of the wind could be other useful information.

Another example would be stock values appended to company names appearing on the current page. Or information about current train schedules for city names contained on a page.

Example: User-generated events

Often a text selection with the mouse combined together with a keystroke can be useful to start a Mission. The *BookFinder* Mission for example lists available books which contain a selected word or phrase when a specific key has been pressed.

Another example is the *TextSpeaker* Mission which starts a synthetic speech of a text selected by the mouse.

11.4.2 Scout

Semantic scraping

An example of semantic scraping, where a subset of the input data is chosen, is the task of extracting telephone numbers found on the current loaded web page. The scraping process can be influenced with different number schemas for different countries. The resulting output is a list of telephone numbers together with position information about the location where the number has been found on the page.

Another example without structural relation to the input is the attempt to discover the language of the current loaded web page. Resulting output

is a textural character description out of a predefined set of known languages. Criteria can be a weighted analysis of the meta description of the page, the origin of the page, the character set encoding, a lookup in the domain name server or a comparison of words found in a specific dictionary.

Both variants depend on the problem to be solved and assumptions about the web page under investigation. For example a textural representation of telephone numbers is a prerequisite to make the analysis possible. If the telephone numbers where shown as images it would need another strategy.

Structural scraping

A possible output of this task could be a list of all hyperlinks or text field areas found on the current page as a simple example. The *TextFieldSize* Mission makes use of the list of text fields.

A more complicated example of a possible output is the header line of the first table-structured data found on a page.

Scraping information generally depend heavily on the page structure and are very fragile to changes of the content. This tentativeness especially holds when external services are being used for scraping.

11.4.3 Properties

Synthetic speech as an example allows the specification of different translation service addresses where ASCII text will be transformed into audio data. These global settings are necessary to make the Mission running. Speech quality can dramatically differ for certain languages. One service provider may offer excellent quality for one language, but another language can come out quite poorly.

On the user-level modifications such as speaking speed or speaking pitch are possible. These configurations may not be available for all defined services addresses and are simply ignored in that case.

11.4.4 Adapter

An example of an Adapter is the transformation of characters into a standard format such as UTF-8. The *Translator* Mission makes use of such an Adapter that extracts textural parts from the current page and changes the encoding of the characters to UTF-8. The adapter should be

as general as possible and in this example be able to change any textual input to one standard output format.

11.4.5 Transformer

In the *PhoneCall* Mission the transformer implements a mechanism to change the input list of telephone numbers in such a way, that each number allows a direct call via voice over ip (see figure 11.7 on page 112).

An example of a modification of the PhoneCall Transformer could be a Mission where additional information is presented. Such information could include the location found in a global dictionary or calling rates coming from a list of carrier providers.

11.4.6 Source selector

Quality criteria

Synthetic speech is an example where a lot of quality variances exists. A service with a good speech quality in one language does not necessary provide the same quality for other languages or might also not exist at all for a specific language. A priority queue based on speech quality for different languages could contain information of several services. If a service is not available, the next service provider found in the priority queue could be contacted.

Best of breed criteria

As an example, a Mission where product prices for a selected item could be looked up might depend on a rating page. The price of the selected item could be checked with several price finder services and then be compared with a list of user ratings for the supplier companies.

11.4.7 Renderer

For example, telephone numbers could be colored differently depending on the type of a number such as mobile phone numbers, local numbers, and foreign countries. But the telephone numbers could also show that additional information by the use of small icons appended to it for example. The behavior of the Mission does not change, only the presentation appears differently.

11.4.8 Injector

Figure 11.7 on page 112 shows a web page where telephone numbers are replaced by a hyperlink together with a small image.

The *Acronym* Mission in section 11.2.2 is an example where the outcome of a Mission is appended at the end of the page.

12 Workflow Example

This chapter deals with the necessary steps to introduce a new Mission, locate the Mission based on different criteria, subscribe to the Mission and finally invoke the Mission for different web pages. These steps include different actors chained together as shown in figure 12.1 which is a different view from figure 6.2 presented on page 67.

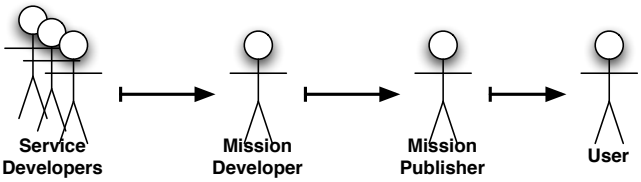

Figure 12.1: Workflow example

The Mission which serves as subject under investigation is the *Distance* example explained in section 11.2.4. In this example a web site without published interface is used as a service provider.

12.1 Involved servers

Three servers are included in the *Distance* example as shown in figure 12.2: The Syndicate server, an interface provider and the distance service provider. The distance service provider is the location where the main work is accomplished. This server offers a service where a distance between two addresses can be calculated. But the server does not offer a public available programming interface to access its service without the use of a web formula.

The interface provider fills this gap of the missing programming interface and acts as a gateway between the Syndicate server and the distance service provider. The interface provider transforms structured data coming from the Syndicate server to formula based data accepted by the distance server.

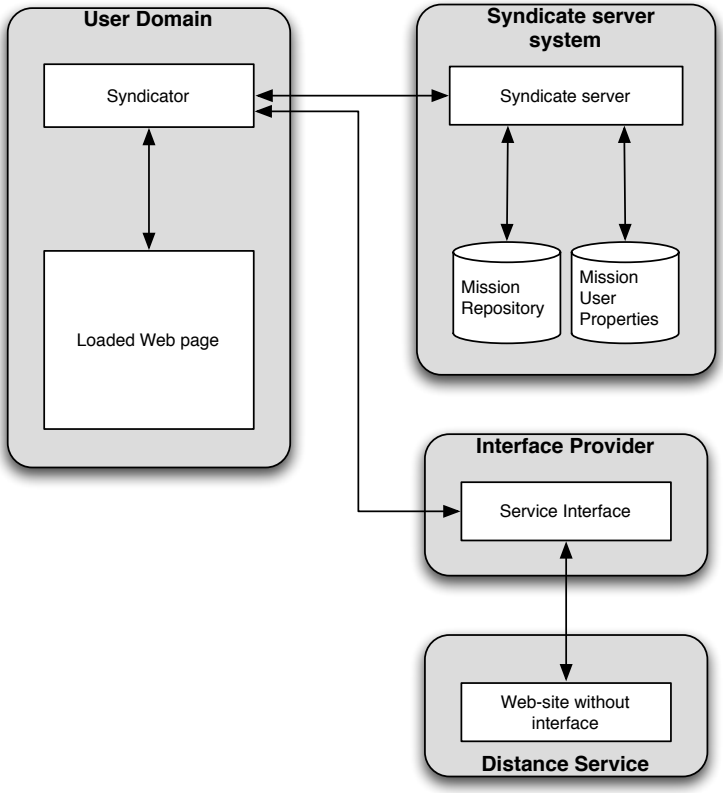

Figure 12.2: Involved servers for the *Distance* Mission

The Syndicate server acts as a turntable between the user's web browser and the distance service provider. Configuration data and access information are stored at the Syndicate server.

12.2 Workflow steps

The service of distance calculation is already available on the web but the service cannot be reached without the use of a web-based formula. This circumstance requires a way to transform structured data coming from the Syndicate server into a format accepted by the distance service

provider. In this example an interface gateway is used to make the distance service available for the Syndicate server.

12.2.1 Mission creation

In the situation of the *Distance* Mission the creation of the Mission consists of two parts: First, the Mission itself has to be created with the assumption that a calculation service exists that accepts structured data. Second, because a distance calculation service will be used that does not accept structured data, an interface gateway has to be set up to fulfill this demand.

Interface gateway configuration

In the *Distance* Mission example, *Dapper* [102] is used as an interface gateway. The Dapper site offers the possibility to use a formula based web site and generates a programming interface where structured data is accepted and transformed into the desired formula. As long as the distance service provider does not essentially change its formula and issued data format, the interface gateway can do its job. But it is good practice to define certain input values and their expected calculated result in order to control the interface behavior.

Mission setup

A Mission developer creates a new Mission by providing methods to complete a desired task as described in the implementation chapter 14. A Mission description, Mission properties and default values have to be set up by the Mission developer and if a test case can be provided additional function guarantee can be given.

The *Distance* Mission makes use of the Content pattern and is categorized with the following keywords: *distance, map, address*. The following text is used as a description for this Mission:

> Calculates the distance between a predefined departure address and a selected text area serving as target address. The distance calculation can be setup to be calculated in Miles or Kilometers.

The default trigger for this Mission is set to the key combination *ctrl-d* and a selected text area. The Mission is not restricted to certain web

sites and can always be activated. The outcome of the Mission is defined as a tooltip window floating at the position where the mouse pointer is located and shows the calculated distance between the departure address and the selected address.

Once a Mission has entirely been defined and packed into a Mission package, it can be uploaded onto a Syndicate server which makes the Mission available for the Mission publisher, but not yet for the Mission users.

12.2.2 Mission publishing

The next step is provided by the Mission publisher as described in chapter 9. The Mission publisher is responsible for the correctness of available Missions and has to examine a Mission closely to detect malicious code. It is very important to mention that Missions have an enormous power when they have been loaded by the user. The Mission publisher is the last instance that can prevent the distribution of Missions if the processing is suspicious.

12.2.3 Mission localization

Potential Mission users connect to a Syndicate server and select Missions based on different criteria as explained in chapter 10. In this example a search for keywords *map, distance* limit the available Missions and the *Distance* Mission appears in the resulting list.

12.2.4 Mission subscription

Once a Mission has been located, the user subscribes to the Mission by selecting it and altering Mission properties as explained in the next section. Subscribing to the Mission only marks the Mission to be loaded with the next syndication of a page but in this example it does not automatically activates the Mission.

12.2.5 Mission individualization

The degree of individualization depends on the Mission itself and on the possibilities the Mission publisher permits. In the *Distance* Mission example the user can define a departure address and can choose the unit of the resulting calculation between Miles and Kilometers. If the

Mission trigger does not conflict with other Missions the user has also the possibility to change the

12.2.6 Mission usage

After syndication of a page the subscribed Missions are loaded from the Syndicate server and installed at the user's side. The loaded Missions are responsive to the Mission trigger if the Mission is allowed to alter the current loaded page which is always the case in this example. The user selects an address with the mouse and presses the key combination *ctrl-d* that triggers the *Distance* Mission. The Syndicate component at the client side contacts the interface gateway provider with the departure and target address. The interface gateway translates the arguments into formula accepted values and communicates with the distance calculation provider. The returned distance is translated back into structured data and sent to the client. The client renders the resulting value into a tooltip window and shows the information as long as the user keeps the mouse pointer at the location of the selected address.

12.2.7 Unsubscribe of the Mission

The User connects to the Syndicate server and unsubscribes from the *Distance* Mission by deselecting the Mission from the list. The Syndicate server then removes user property settings. The next time the user syndicates a page, the *Distance* Mission is excluded from the list of loaded Missions.

Part IV

Syndicate Implementation

13 Technical Challenges

A manifold system builds the basis for a pattern-based composition of distributed facilities. The requirements of a manifold system combined with the claims by the Syndicate concepts lead to technical challenges and force to use particular technologies. The following list summarizes the resulting aspects as a more concrete version of the generic demands on a manifold system:

Installation-free syndication: In order to syndicate a web page, it should not be necessary to install software (On the go).

Mission composition of different web sites: Data from different web sites can be combined into a single Mission (Loose coupling).

Browser independence: Syndicate is not limited to one web browser (Adaption).

No site restrictions: Missions can access content origin from different sites (Loose coupling).

Service substitutability: Missions can depend on a set of alternative service providers which are dynamically selected (Best of breed).

Semantic data structures: Structured data can be integrated in a meaningful way (Adaption).

Access to sites without API: Functionalities from sites that do not offer public API's can be integrated (Plug and Play, Anonymity, Auto discovering).

Interaction with page structure: Missions can be triggered by selecting specific parts of a web page (Adaption).

Dynamic runtime configuration: It is possible to control the behavior of Missions while the application is running (Plug and Play).

Technical challenge	Manifold principle
Installation-free syndication	On the go.
Mission composition of different web sites	Loose coupling.
Browser independence	Adaption.
No site restrictions	Loose coupling.
Service substitutability	Best of breed.
Semantic data structures	Adaption.
Access to sites without API	Plug and Play, Anonymity, Auto discovering.
Interaction with page structure	Adaption.
Dynamic runtime configuration	Plug and Play.

Table 13.1: Technical challenges in relation to manifold principles

In order to come up with these points Syndicate combines particular technical solutions. Syndicate uses the following sequence of events: To start up the user selects a set of desired Missions on the Syndicate server. From that point on the user can syndicate any page that has been loaded by the web browser. As soon as a web page has been syndicated all static Missions are executed and the browser continues listening to Mission triggers. The control flow shown in figure 8.2 on page 86 outlines the sequence of events to syndicate a page.

To allow any page to be syndicated there are essentially three different possibilities. The first one uses the existence of a proxy-server which can modify all traffic going on between the client web browser and another target web site. There are several other projects related to Syndicate which make use of this technology (see 5.2).

The second option uses a flavor of certain browsers which allow the extension of the browser application. This variant allows a complete modification of loaded web pages and a few projects implement such a strategy as shown in section 5.1.

The third alternative which Syndicate is based on uses simple hyperlinks to inject code into the current loaded page.

If a hyperlink is enough to change the behavior of an already loaded web page, it must be possible to execute a program code without reloading

the page. The execution of a JavaScript code within a loaded page is possible with the use of Bookmarklets.

But as later described in section 13.1.5, Bookmarklets cannot directly be used to run Missions. They require another technology to inject code into the current page. This is known as cross-site scripting (XSS). Cross-site scripting has advantages when it comes to possible modifications but at the same time weakens security. Security issues are later covered in section 15.1. Beside cross-side scripting some browsers also allow the implementation of signed scripts (see 13.1.6).

The restricted possibility of connecting several sites together is also known as 'same origin policy' and comes from the Netscape Navigator 2.0. Cross-site scripting includes a way to get around this limitation.

To make sure the application is not restricted to one particular browser and eventually achieve browser independence, cross browser scripting strategies have to be implemented. The following section contains details as well as the preferred solution as implemented in Syndicate.

Service substitutability requires connections to other systems that can be exchanged. This demands Loose coupling service partners as explained in section 13.4.

Whenever structured data occur in the process of interfacing a service on the web, the advantage of a clear description should be taken into account.

To access functionalities of sites that do not support a public interface, a way of taking advantage of the potential is needed. One strategy to get to the desired information is the use of scraping techniques.

One Mission trigger variant is based on the selection of specific parts of a page structure. Visual feedback has to be setup to let the user be notified of the selected part.

Runtime configuration is a comfortable way to control the behavior of an application. Within Syndicate it is used to select desired Missions and specify their individual environment settings, named Mission properties in this context.

13.1 Browser scripting

While browser scripting is a general term without fixation on a specific language, it will be cut down to the usage of JavaScript as the preferred

scripting language. Executing JavaScript in a browser assumes that the user did not prevent such an execution due to security or other reasons. The usage of browsers or devices that are not capable of running JavaScript needs to be handled with special care as well.

13.1.1 Cross browser scripting

As described in section 3.3.1 JavaScript is a standardized language. But writing scripts that make use of the DOM interface – which is usually the case when running a browser – lack of standardization among various browsers. The implemented DOM interfaces don't always match the W3C DOM standards. The availability of functions and properties must be tested during execution and handled differently for each browser. Especially in connection with AJAX programming as mentioned in section 3.4.1, differences exist and make development more complicated.

Functionality	Plugin
Modal dialogs	jqModal
Tool tips	Tooltip
Additional interface components	interface
Support for audio data	jQuery Media

Table 13.2: Included plugins for JQuery

Meanwhile object oriented programming libraries exist that completely hide browser differences and support AJAX programming and DHTML effects. DHTML is a shortcut for dynamic HTML programming which includes manipulation of the DOM interface with JavaScript and is often used for visual effects of extended interaction possibilities. *Prototype* [103] [104], *JQuery* [105], *Dojo* [106] and *Script.aculo.us* [107] offer comfortable methods and a rich function library. All of these libraries can be extended by third parties offering even more functionality. The amount of available extensions makes the main difference among these products. It is not necessary to make use of a library, but it simplifies the development and cuts down the number of lines of code. Syndicate uses *JQuery* together with a set of extensions. Extensions are called plugins within *JQuery* and are loaded at runtime. Table 13.2 lists plugins originally included in Syndicate.

Syndicate is not limited to *JQuery* and the used plugins. Missions can be implemented together with any JavaScript library or even no library at all. But *JQuery* offers a convenient way to modify the loaded page and make use of AJAX functions.

13.1.2 Cross site scripting (XSS)

XSS is originally a type of vulnerability in computer security. Malicious web users try to put code on web pages viewed by other users to let the code be executed within their web browser. This code can pass access controls and restrictions originally given by the user's browser.

Three distinct types of cross site scripting vulnerabilities are known [108]. These types all deal with possibilities to insert malicious code into the user's web browser with a vulnerable web site. Syndicated pages make use of the cross site scripting technology because of the reason explained next, but they do not allow users to insert arbitrary code.

Cross site scripting can also be used to get around a browser restriction known as *the same origin policy*. The same origin policy prevents documents or scripts loaded from one origin from getting or changing properties of a document of a different origin. Two web pages are said to have the same origin if they use the same protocol, the same port number and the same hostname in their URL. For example the following two URL's would pass the origin comparison:

```
http://syndicate.unibas.ch/Missions/MissionList.html
http://syndicate.unibas.ch/Properties/PropertyList.html
```

But another comparison to

```
http://www.serviceprovider.ch/service/store.html
```

would result in a failure. With this restriction in place it would not be possible to contact the Syndicate server and communicate with other service providers at the same time.

13.1.3 Callback functions

Callback functions are a method of reaching asynchronous communication. The basic principle is also known as the *"Hollywood Principle"*, or *"Don't call us, we'll call you"*. Sending a message is usually a non-blocking operation and objects can continue processing after sending a message

without waiting for a response [109]. But receiving a message on the other hand is more difficult because receiving objects have to wait and listen to the communication channel if the response has already arrived. With callback functions and a message queue, the message queue handler invokes such a function whenever a new message has been received. This modality naturally offers an asynchronous behavior of the communication system. Syndicated pages make intensive use of callback functions with Missions calling external services.

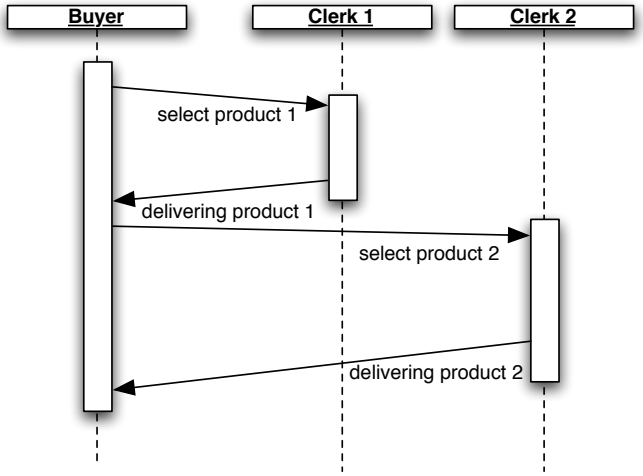

Figure 13.1: Synchronous message queuing

An example of the distinctive behavior in real life is shopping in a shopping center and buying two goods as shown in figure 13.1. Figure 13.2 shows the shopping process with the usage of electronic ordering where the two orders can be submitted without waiting for the delivering of the first product.

13.1.4 Browser extensions

Syndicate claims to allow any page to be syndicated and run missions within that page. In order to fulfill this goal it must be possible to run certain code within a page without being able to change the page beforehand.

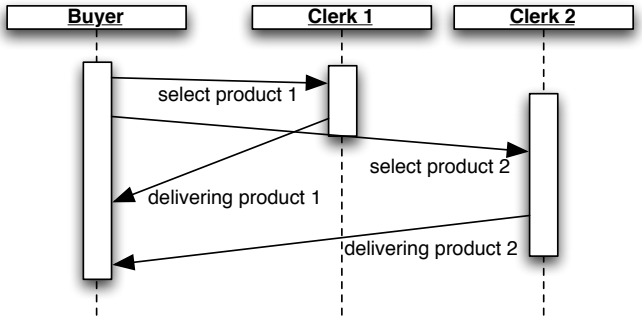

Figure 13.2: Asynchronous message queuing

The most common solution is the installation of a program on the client side that allows changes of the current loaded page. Firefox extensions as explained in section 2.5.1 are frequently used to solve the problem of running code inside arbitrary pages. Chickenfood, Greasemonkey, iMacros for Firefox and Stylish as described in chapter 5 typical examples of this family. As already mentioned, are these products all have the drawback that a program needs to be installed first, and they all depend on a specific web browser, mostly Firefox.

13.1.5 Bookmarklets

Another possibility are Bookmarklets [110], also known as Favelet. The term itself is a combination of a bookmark and an applet in the sense of a small application. It does not relate to Java Applets as described in section 3.3.4.

Bookmarklets are a way to run JavaScript programs for the current page. A Bookmarklet is a small JavaScript program stored as a URL within a bookmark, or stored within a hyperlink on a web page. The JavaScript program is restricted to one line of code but can otherwise make use of any function available at the client side. The major web browsers do support the usage of Bookmarklets.

Bookmarklets offer interesting possibilities of a web page within the browser. The following list shows examples what is possible with Bookmarklets:

- Extract data from a web page.

- Modify the appearance of a web page.
- Use the current page or a selected text on the page as a parameter and request another web page.

Syndicate uses Bookmarklets in combination with cross site scripting as shown in the last section to syndicate an arbitrary page. Therefore the user only needs a stored bookmark to syndicate a page, without installing software in advance. The Bookmarklet injects a small piece of JavaScript into the current page which loads additional code from the Syndicate server. From the point of the JavaScript injection the control has been handed over to the Syndicate server. This offers functionality but at the same time weakens security. An analysis of security issues are given in section 15.1.

A direct modification of the web page by the Bookmarklet is not practicable since all the running code must fit into one line of code and the number of characters are limited browser specific (see table 13.3).

Browser	Number of allowed characters
Netscape	more than 2000
Firefox	more than 2000
Opera	more than 2000
IE 4	2084
IE 5	2084
IE 6	508
IE 6 SP 2	488
IE 7 beta 2	2084

Table 13.3: Number of allowed characters in Bookmarklets

13.1.6 Signed Scripts

Signed scripts are a technology used in Mozilla web browsers to access expanded privileges in JavaScript applications. The JavaScript security model for signed scripts is based upon the Java security model for signed objects from Communicator 4.x. In order to use signed scripts a digital signature has to be generated and assigned to the JavaScript it uses. A

certification authority such as VeriSign [111] proves the identity of the owner of the script and that the script hasn't been modified by someone else. A signed script can request expanded privileges and allow access control at a detailed level. When the user access a document containing a signed script he may allow or deny the execution of the script. If the user denies the execution of the script, an exception is thrown which can be handled programmatically.

The security philosophy behind this policy is based on trust. When the user has proof of the author's identity, and the guarantee that the script has not been modified since its original content, it is up to the user to allow or disallow the execution. If the user trusts the author and grants permission, the script may perform sensitive actions. Such actions can include accessing the browser's preferences, or reading or writing from or to a file on the user's hard disk.

Because all of the JavaScript code on a HTML page runs in the same process, a distinction of different signatures should be possible to protect the scripts from each other. But Mozilla does not currently support multiple signatures.

Although restrictions exist, signed scripts allow secure environments at a fine-grain level and are a way to ensure scripts haven't been affected by a malicious code. Nevertheless signing scripts is combined with additional effort and – as it can be seen by browsing through the web – the majority of web pages does not make use of this technology.

13.2 Microformats

The easiest way to make use of data formats happens whenever a semantic description of structured data exists. One approach of semantic descriptions includes the existence of a standardization committee such as W3C. But this often leads to a time consuming process ending up in new, complex structures which are complex to use. The gap between SOAP and REST as a way to access web-based services is such an example and is described in section 3.4.3.

Another approach is based on a pragmatic way where already existing technologies aid to solve a particular task. In the case of semantic information, cascading style sheets (CSS) are currently being used to describe structure and meaning of data. The microformats-wiki community have led to de-facto-standards such as the semantic description of calendar entries (hCalendar), address information (hCard), reviews (hReview) or

human relationships in social networks (XFN). This bottom-up approach will eventually lead to an implementation of the semantic web [112].

As soon as a description of structured data exists, technologies such as xml schema description and dependent programming libraries, for example XmlBeans [113] are of great assistance.

13.3 Scraping techniques

If a web site does not offer a programming interface to access its published information or if the data structure has no semantic definition, scraping techniques are a way to extract required information.

Scraping means the analysis of a web site and finding the location of the data part on the page in order to be able to pull it out. Analyzation of a page means parsing its HTML source code and XPath expressions are a great way to support this task. Beside implementing the parser by hand, libraries or web-based services can assist in this process. For example *Dapper* [102] offers a way to train its integrated browser to find desired information on a page. Once the information has been found, it can be exported into different data formats and processed further. HTML, JSON, RSS, and XML are possible outcomes and especially the latter three are all favorite formats. *Dapper* offers a comfortable web interface to simplify the creation process.

JScrape is a Java library that supports the extraction of information from a web site without service interface. *JScrape* uses the XQuery language to do most of the hard work and makes finding information on the web page much simpler and contained to a single query. Several existing third party libraries have been included in the *JScrape* package in order to be able to analyze a web page with its DOM structure. The combination of commonly used software improves the overall stability of *JScrape* which is still in an alpha state by the time of writing. A similar product is available for the Ruby language with scRUBYt! [114].

Chickenscratch, a language based on the appearance of the user interface in web sites could be another possibility to access data from a site. Section 5.1 explains this related project in more detail.

Syndicate can make use of any scraping technique. The Mission where distances are calculated uses *Dapper* for example.

All scraping techniques depend heavily on the source code of a web page and make development and usage very fragile.

13.4 Loose coupling

Since loose coupling is a design goal and not a specific problem to be solved it cannot be reached with a choice of a product. As mentioned in section 2.3 and section 6.6 loose coupling can mean independence in terms of time, data formats or information knowledge.

Time independence means asynchronous communication; and AJAX, or method callbacks to be more specific, is a great way to achieve it. Syndicate uses callback methods with dynamically injected code based on cross site scripting technologies to enable asynchronous communication among service providers (see 13.1.3).

Loose coupling among service partners can be enhanced when a flexible data format is used for message transmission. XML, JSON, and RSS are formats that allow message recipients to publish definitions on how they extract information. XPath expressions or XML schema could be used for example in order to show how a data format is to be expected.

Loose coupling of services can generally be improved by keeping message transmission at a minimum level and reduce information to key elements. This technique comes from the information hiding principle known from object oriented programming.

13.5 Visual feedback of selected page structures

One type of Mission trigger is based on a selection of particular parts of a web page. It is very helpful to have a visual feedback of the current selection and be able to expand or narrow the selection to the neighborhood of the chosen element. The DOM structure of a page is used as an indicator for possible picks.

Firebug [115] is a Firefox extension and an excellent example where visual feedback is used to inspect the structure of a loaded web page. *Firebug* offers a wealth of web development tools to edit, debug, and monitor a loaded web page. Since *Firebug* is an application, it can only serve as an example.

A similar product, although with less functionality is *Aardvark* [116] which is a Firefox extension to clean up unwanted advertisement banners and examine a loaded page. Syndicate uses and extends code from *Aardvark*.

13.6 Dynamic configuration

The topic of dynamic configuration is twofold. Dynamic interaction should be possible at the level of the user's browser on a syndicated page and at the syndicate server where Missions can be configured and subscribed.

13.6.1 Configurations in the web browser

Once a page has been syndicated and Missions have been loaded from the Syndicate server, interaction mechanisms must be in place in order to control and trigger the loaded Missions. These interactions are initiated by triggers coming from a user activity or from other trigger variants as introduced in section 7.1.5.

In order to respond to user activities such as mouse gestures or keystrokes, it must be possible to attach event handlers to elements of a loaded page. *JQuery* greatly supports event handlers and is used for taking care of user interactions (see 13.1.1). Configuration support at the client-side is currently limited to Mission activation.

13.6.2 Configurations at the Syndicate Server

Dynamic configuration of Missions requires two distinct functionalities. First it demands a personalized controlling mechanism of Missions and second, it needs a way to store such information if settings should be kept permanent.

If a Mission is currently running, JavaScript methods must be locally available in order to change the Mission's behavior. If the Mission makes use of the Syndicate server in proxy style, then a web-based service should provide an interface where settings can be changed. This means at the same time that the access to the Syndicate server must be personalized, and access control mechanisms should be in place. Several variants exist at different security levels. At the moment Syndicate does not implement an authorization possibility.

Storing of user settings can be done at the client- or at the server-side. Client-side settings would result in saving data to the user's filesystem, which are considered to be a security risk, or in the use of cookies. Cookies have the advantage that users do not need a personalized access to the server, and cookies can be controlled by the user's browser settings.

Server-side storage requires the implementation of a database where information is kept user specifically and therefore needs a way to authorize at the server-side.

Currently server-side settings are not stored permanently in Syndicate and therefore a personalized user access at the server-side is not required.

14 Implementation Architecture

The previous chapter describes all necessary techniques that are used to build the Syndicate system. This chapter introduces distinguished parts of the implementation but leaves out programming details at code level. The explanations given should glue the loosely described components together and provide an insight into the hidden mechanisms.

14.1 Data

Syndicate makes use of two different types of data: Structured data, being used whenever possible as communication and configuration storage and JavaScript containing behavior of components. The following sections introduce important implementations of the two variants.

14.1.1 Missions

From the implementation point of view a Mission consists of descriptive data stored as Mission properties and behavior specifications. Both characteristics are specific to a particular Mission. Properties contain static information relevant to run a Mission and are covered in the next section. Behavior of a Mission is kept in JavaScript functions and stored as a single script containing all necessary methods. If a Mission makes use of a server detour another code portion is stored at the Syndicate server. This code can be implemented in any server language but Java servlets are favored.

The code for Mission trigger recognition and handling of a loaded page is common to all Missions and stored separately. Page handling includes detections of page structures and visible feedback of such. The general functionality of the Scout and the Transformer are also kept in a commonly used file.

The syndication process and the transfer of the Mission behavior part to the client is outlined in section 14.2.2.

14.1.2 Mission properties

Mission properties contain static information needed by Missions to resolve provider addresses and configuration options. Properties are stored as JSON structured data and are divided into system and user properties. The distinction only lies in the way the properties are treated and allowed to be changed by the user. System properties are stored together with the Mission's behavior definition and user properties are stored individually to each user.

14.1.3 Communication data

Various communication channels are used within the Syndicate system. The following table 14.1 gives an overview about the different communication partners and their common data format used to exchange information.

Communication Partners	Communication method
User interface - Client Consigliere	JavaScript and JavaScript callback functions.
Client Consigliere - Syndicate Server Consigliere	Cross site scripting, JavaScript callback functions
Client Consigliere - Syndicate Server Mission	AJAX
Client Consigliere - External service provider	REST, SOAP, XML-RPC, JavaScript callback functions
Client Consigliere - External interface provider	REST
Syndicate Server Mission - External service provider	AJAX, REST, SOAP, XML-RPC

Table 14.1: Communication partners and methods

Figure 14.2 on page 146 shows an overview about participating components. The connection between the client's web browser and an external interface provider that connects to an external web server without public available programming interface is omitted. All communications are kept asynchronous whenever possible. Access to external service providers

can take place synchronously if an asynchronous access is not available.

Communication between components make use of self contained documents or other techniques to achieve context bound interactions which is a requirement for asynchronous messaging. Another prerequisite is the usage of message queues to guarantee that sending components do not exceed the capacity of recipients. The communication layer that makes AJAX possible already includes message queuing with its XMLHttpRequest object.

14.2 Processes

All processes appearing in the Syndicate system are event driven. Event driven processes are a premise for a Loose coupling system which is one of the key requirements of the Syndicate system. Processes are running at several locations, namely at the client's web browser, at the Syndicate server and at external sites such as service and interface providers.

This sections explains major activities appearing during the usage of the Syndicate system. The cooperation of the code at the client side and the corresponding objects at the Syndicate server side makes out the most complex part.

14.2.1 Syndication

The Syndicator is responsible for the syndication of a loaded web page and acts as the kick-off of the whole Syndicate system. The Syndicator is implemented as a single line of code that makes use of Bookmarklets and cross side scripting technology. (see 13.1.5 and 13.1.2).

After activation of the Bookmarklet by the user, a JavaScript method pulls more JavaScript code from the Syndicate server and installs the received code into the current web page. The received code contains all Mission specific elements in order to trigger, activate and run them. The next section describes the mechanism of Mission loading.

Once the code has been integrated into the loaded web page, the control lies at the client side and Syndicate server has completed its job for now. The client continues with Mission trigger installations as described in section 14.2.3.

14.2.2 Mission loading

Mission loading is a pull mechanism initiated by the user at the client side. The JavaScript code activated by the user contacts the Syndicate server to receive more code from a specific address. The address does not host just a simple file but is actually a Java Servlet that gets activated. The servlet provides the functionality of configuring an individualized JavaScript code that contains all Mission specific definitions and property settings. This compiled code is then sent back to the calling client and inserted into the script section of the current loaded page.

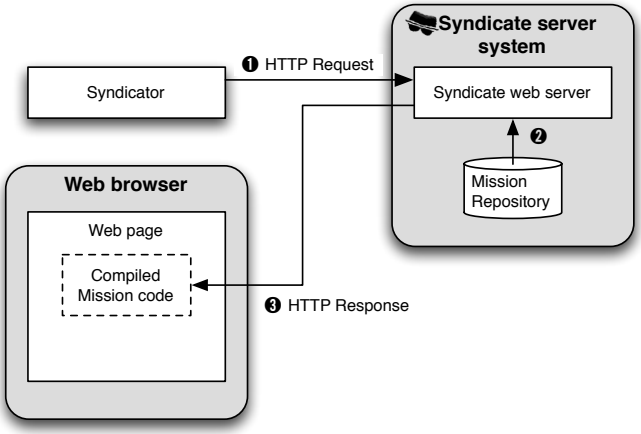

Figure 14.1: Mission loading

1. The user activates the Syndicator which contacts the Syndicate server to load JavaScript code.
2. The Syndicate server consults its Mission repository and compiles an individualized code portion for the requesting client.
3. The server sends the compiled code back to the client who includes the data into the code section of the current loaded page.

14.2.3 Mission triggering

The Mission triggering process consists of two parts: Static and dynamic Mission triggers. In the first part, static Missions are immediately acti-

vated as soon as all Mission definitions have been loaded. This is achieved through calling a local JavaScript method, that invokes all statically defined Missions.

In the second part, dynamic Mission triggers are installed with the registration of event handlers. The event handlers are already defined, only Mission specific definitions are added to the predefined ones. This includes inscribing of additional keystrokes or installing periodic interval timers.

14.2.4 Mission activation

When a Mission has been activated by a Mission trigger the corresponding JavaScript method of the Mission is invoked at the client's web browser. Figure 7.1 shows the interactions between the different components that take place at this stage. Missions that are based on the Content or Integration pattern will contact external service providers to fulfill their task. Figure 14.2 shows an overview about the interactions between the different components.

The following description state the chronological steps taking place from the syndication of a web page to the invocation of static Missions up to the time when the user triggers another Mission.

1. The user syndicates a web page which starts a request to the Syndicate server.

2. The Syndicate server consults its Mission database and selects the users Mission setup. The server responds with JavaScript code that makes an activation of the Missions possible.

3. Static Missions are activated, resulting in an analysis of the loaded web page. Depending on the Mission further actions such as a direct connection to another web server or to the Syndicate server are taking place.

4. A service provider at another web server is called. This is called a server detour.

5. The service provider delivers the processed request back to the Syndicate server.

6. The server Scout analyzes the service response and extracts needed information. The Transformer combines different input sources

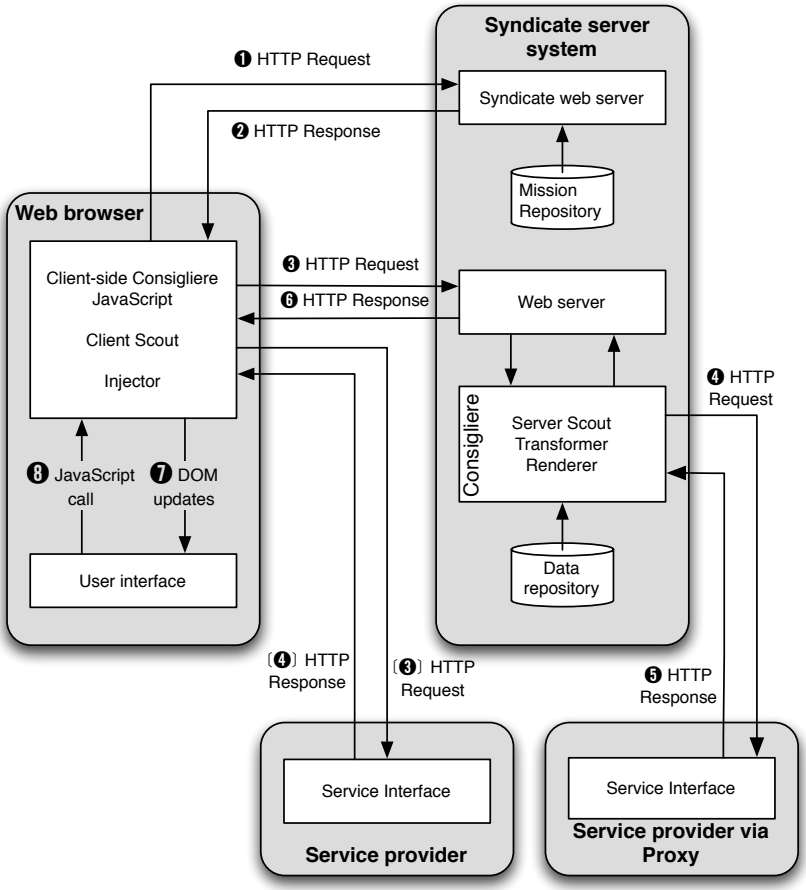

Figure 14.2: Syndicate architecture

and produces the result of the Mission. The Renderer creates the Mission specific presentation format. The Consigliere finally makes a call to the Injector at the client's web browser.

7. The Injector modifies the current web page.

8. Mission triggers are activated. The user can start another processing which results in a new control flow shown in figure 8.3.

Figure 14.2 shows two possible invocations of service providers. The first possibility is a direct call from the client's web browser to another web server that offers a web service. This option is the preferred variant when a service provider allows a direct connection to its web service and makes its services available with a public available programming interface.

The server detour as second option uses the Syndicate server in a proxy style manner to call another web service. While this version at first seems to use an unnecessary round trip at the syndicate server it has several advantages as stated in section 7.2.2.

Part V

Future – Perspectives of Syndication

15 Syndicate Risk Management

The intention of this chapter is to identify potential risks and analyze strategies to manage a controlled environment where the capabilities of Syndicate can unscrupulously be used. Mitigation of risks include technological and organizational aspects.

15.1 Security

Syndicate Missions are dynamically picked up components from a Mission repository that can modify visited web pages. These components are stored on a remote Syndicate server and can contain programs to be executed within a user's web browser.

Enabling Syndicate Missions to be executed within a user's domain poses questions about security aspects as soon as the origin of the Mission is dubious. When Missions are initialized from a Syndicate server, it therefore hands responsibility for security policy to those who have installed the Missions. This leads to a chain of trust of all involved entities as shown in figure 15.1. A Mission developer can include other

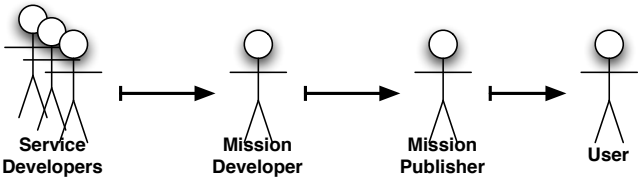

Figure 15.1: Chain of trust in Syndicate

available services and for that reason service developers conducive to the participants.

In general Syndicate Missions have full access to a loaded web page on the client side and personal user data stored locally. Sensitive information on a web site like account numbers or personal data is visible to Syndicate. The dependence on JavaScript which has control of features like event

handlers can enable syndicated pages to snoop on anything the user is doing within a page, such as capturing user input in a password field for example. Additionally, malicious Missions can attempt to trick the visitor into providing information, or performing an action that will enable an attacker to gain more privilege. Without constraints, Syndicate can potentially allow JavaScript code written by other people to be executed within the user's web browser. If one of those scripts is malicious, the local system could be at risk. It is therefore indispensable to establish a security policy.

The alignment of involved contributers shifts all the power in the chain of trust towards the Mission publisher. This focus on a specific actor is an advantage against comparable products. For example Greasemonkey, a browser extension that can be installed locally, allows the execution of installed programs, but requires the user to take responsibility. Currently Greasemonkey is in the list of the top ten Firefox extensions to avoid exactly because of this reason [117] [118]. Greasemonkey is a handy extension to use, as long as a user knows what he is doing with it. But it does not offer a possibility for people concerned about security to limit the usage to certain scripts. The only way to prevent such usage is by completely disallowing the installation of the Greasemonkey extension.

In a first approach of security policy, suspicious parts in the participating elements of a Mission can be neglected by the Mission publisher. Mission developers may build new features to be used in syndicated pages and maybe integrate web services from third parties to complete its job. These web services may already contain unreliable or undesirable components. But it is up to the Mission publisher to allow or deny Missions on a Syndicate server.

To accomplish code execution downloaded from untrusted sources across a network and at the same time offer protection of the user, the concept of a sandbox security model is a possible solution [119]. A sandbox restricts code from taking any actions that could possibly harm a user's system. The advantage of using a sandbox is the independence of the decision what code can be trusted and what cannot be trusted. In order to make full usage of a sandbox within Syndicate, it needs to be implemented on the Syndicate server [120] [121].

The described security aspects are of a general nature and apply to any system exposed to the public. Installing a downloaded plugin into a user's web browser or opening a document received from suspicious

origin already gives up privacy. Security concerns have to be addressed at a more global level and I propose a *web of trust*, built upon a trusted peer to peer community. Such a network would connect trusted Syndicate servers and contributing users with each other. A peer structure does not require the existence of a complex public key infrastructure [122] [123].

15.2 Service guarantee

At the moment there are no standard ways to guarantee the quality of a service (QoS). The Web services flow language (WSFL) [124] might include a way to describe how well a service should perform. But until today the industry has not agreed on such a workflow standard.

Syndicate depends heavily on other information providers without letting them know that they are being used. This is a weakness of Syndicate nevertheless it offers great flexibility and simplifies the connection to other services. The reliability of Syndicate depends on different suppositions which can be grouped into the following areas of a Mission: Mission patterns, resource access methods, and resource gathering methods.

Missions based on the presentation pattern are less likely to fail and Missions based on the integration pattern are the most unsteady ones. This is due to the fact that the number of external services involved increases with the content pattern and the integration pattern. However it does not necessary mean that it is unreliable to use integration pattern based Missions. Especially when several alternative service providers are registered, the chance to prosper increases.

Resource gathering methods of the Scout are another tentative area. The more assumptions the Scout makes about the structure of a page, the more likely that a change in that structure will lead to an unexpected result. But technologies exist that support the analysis of a page and are robust enough to resist page changes (see section 13.3).

Automatization techniques to control Mission availability and success are planned.

15.3 Performance

The way the Syndicate system is designed, it allows the making of several round trips and connections to different sites in order to fulfill a certain task. This can lead to a performance problem if several connections have

to be made before a Mission has been completed. Nevertheless, server detours allow the Syndicate server to make use of caching or compression techniques. In case of a performance issue, scalability can be reached by distributing the workload to several other servers, each server serving its own application area.

Most functionality is executed on the client side. Scraping, rendering or injecting of the transfered information uses the local power available on the client side. This narrows performance problems to communication.

15.4 Legal issues

The remixing possibilities of content from different sites in Syndicate pose the question of legal restrictions. This question has already been addressed in the context of web 2.0 and mashups (see section 2.4.4) but needs to be considered, since Syndicate has another target audience. Mashups are web sites or applications that combine content from one or more sources and publish the created remix on the web. For example, Cellreception.com combines Google Maps with a database of 124,000 cell phone tower locations to help users determine where mobile coverage is strong and where it isn't. Information from two different companies, namely Google and Cellreception appear on the same site and must be free of copyright issues in order to be presented on a public available web site. Since the application comes from Cellreception itself and Google offers its mapping data under the Google terms of service which allows personal, non-commercial use only, it does not call for legal issue. But if a third company would remix the output of Cellreception with another data source – the carrier costs of phone companies for example – it becomes an issue.

A growing number of internet users allow the usage of their data on a freely basis or after a cost free registration. Creative Commons is a non-profit organization devoted to expand the range of creative work available to others to legally build upon and share. Several copyright licenses known as Creative Commons licenses exist that permit certain rights of the work. The intention is to avoid problems with current copyright laws for the sharing of information [125].

The situation in Syndicate is different since functionality and not content is being published. Missions can combine content of certain web sites but the remix is not directly visible on a public web page. The remix itself happens on the user's web browser and therefore avoids the

license problem of public available mashups.

The power of Syndicate does not prevent the user from paying attention when he continues with the outcome of a Mission. Due to the mixing facilities of several sources the origin of the data may vanish. While this is not a problem for personal usage, it becomes a copyright issue when the combined data is published again on the web.

16 Conclusion and Outlook

A summary of the main points of this thesis will be presented. Secondly, the conclusion of this study is shown by recapitulating the answers to each of the research questions. In the last section of the chapter I will discuss the limitations of the research and finally, I will also present suggestions for future research on this theme and present my personal vision for the future of the web.

16.1 Summary of the research

The motivation of this research has been the desire of more flexibility and customization of web applications. Areas that allow personalization have been analyzed and categorized.

An investigation of requirements for individualized applications resulted in the usage of a service-oriented architecture. As a subsequent intention, universal principles of a manifold system were described and conditions of applications that are in the focus of manifold systems were stated, related projects were summarized and evaluated against these conditions.

In the second part the indicated categories were compared to the Model-View-Controller paradigm of object oriented programming. This has led to three patterns that were identified as the content pattern, the presentation and the integration pattern. Subsequently, a new terminology was introduced and concepts were presented serving as a theoretical background for a software architecture which enables pattern-based individualization.

The third part covered a concrete implementation in the form of the Syndicate framework of the suggested concepts and several examples were illustrated, both from the user and developer point of view. Implementation issues and technical challenges were addressed in part four, which led to a detailed description of data and processes.

The practical implementation allowed investigation of the effectiveness of the proposed concepts. It could be demonstrated that Syndicate is a

useful and compelling way of individualized service compositions in the area of the World Wide Web.

16.2 Conceptual framework and conclusion

Syndicate is an architecture designed to allow individual service composition on the web. Without the need for software installation, Syndicate empowers users to change their web experience based on their needs. With its possibility of social software contribution on the Syndicate server, the system can take advantage whenever users publish new functionalities. Syndicate focuses on ease of use for end-users to apply customizations to visited web pages, but at the same time offers developers the possibility to create new tasks without restricting them to a specific browser or development environment.

Many organizations have realized what possibilities are offered by the web and use the web as the preferred platform for application development. The shift from applications that were conventionally developed for the desktop to applications that are now being migrated to the web, offers interesting possibilities in conjunction with Syndicate. Applications, when used on a desktop, allow customization possibilities that were previously planned by the programmers. With the existence of web versions, the applications suddenly become an open playground that can be extended and altered according to the user's need.

In the following sections I will present the main findings of this study. The conclusion is discussed by answering each of the preluded questions stated in the introduction.

Categories of individualization on the web

The first research question was defined as 'How can existing web sites be categorized, and what are the resulting requirements of a software architecture allowing individual composition of these categories?' This research question was further defined through two sub-questions: 'How can personal services be categorized?' and 'What are general requirements of a software architecture enabling individual composition of these categories?'. Thus, the first research question dealt with the essence of personalization areas, both in terms of web pages and web applications. This question was answered by supplying a pattern based categorization, borrowed from object oriented programming technologies. The second

question was posed as a result of a closer analysis of service-oriented architectures. A set of principles, describing the requirements of a manifold system useful in the context of personal services, concluded the analysis. Thus, as a first conclusion, the following outcomes were formulated:

> *Three patterns, namely presentation, content, and integration patterns categorize individualization areas in the context of web pages and web applications.*
>
> *Principles required of a software architecture which supports individual composition of the stated patterns are summarized by the term of a manifold system.*

Architectural design for individual service composition

The second research question was defined as 'How does an application architecture have to be designed in order to accomplish the stated requirements of a manifold system?' This question relates to the categories and requirements concluded from the first question. The answer to this question was sought by introducing a new terminology and describing concepts that support the stated principles of a manifold system. Thus I present the following conclusion:

> *The Syndicate framework includes concepts that comply with the demands on a manifold system that allows individual service composition in the area of the World Wide Web.*

Practical implementation of the Syndicate framework

The third research question focused on a practical implementation in order to give use of a prove of concept. The question has been posed as 'How can the architecture be realized in practice and what are the experiences when applied to different fields?' It has been demonstrated that the proposed concepts can be realized in practice with the use of standard technologies. The Syndicate framework has been implemented and is accompanied by several examples supporting the different categories which were formulated as a result of the first question. Thus I can make the following conclusion:

The Syndicate framework proved to be an effective way to make use of individual composition of existing services and applications on the World Wide Web.

16.3 Contributions

I have introduced an architecture that allows end-users to change or extend loaded web pages into customized versions. Concrete requirements have been defined that would offer a convinient way of finding, selecting and using individualized services on arbitrary web pages.

A new terminology was introduced and a theoretical concept was formalized based on the previously defined requirements. Abstract data structures and processes were described to serve the stated purpose.

Related projects were investigated and demarcated. Whenever possible, beneficial elements were taken into the proposed concepts.

The theoretical concepts have been implemented as a prototype in the form of the Syndicate framework. Standard available technologies were used and several examples have been realized in order to give a proof of concept. Examples of related projects have successfully been transferred into the Syndicate system.

16.4 Limitations of the study

This work does not claim to provide a complete error-proven, robust environment that is resistant to security attacks or other inconveniences. But it allows the studying of a new range of possibilities and gives pragmatic examples of customized application on the web. The opening of the system to social contributions has the potential of growing and expanding into new areas which have not yet been fully comprehended or foreseen.

16.5 Avenues for future research

The core of Syndicate has been implemented as a prototype. Besides necessary enhancements to guarantee robust, secure performance in daily usage, there are other possible extensions to the system. This section lists fields where further research could improve the overall usability.

16.5.1 Community aspects

Service composition

Missions based on the integration pattern, by definition, make use of several web services. Due to the lack of web service middleware, a combination of such services currently needs to be programmed manually. Service composition models and their accompanying languages are an approach to standardizing and supporting the creation of service composites. Although the area of service composition languages is still rather immature, the Business Protocol Execution Language for web services (BPEL) seems to be becoming the leading standard in describing services at a higher level [126] [127] [128] [129]. Such service composition languages allow the description of workflow systems that define the order and conditions of operations to reach a desired goal.

In an ideal world it is imaginable that web service compositions could be defined on an abstract level, which is known as an orchestration model. With standards in place it would be possible to graphically create complex Missions without the need for programming. An investigation in that area could be worthwhile [130].

Network of Syndicate servers

The Syndicate server could be extended to a network of servers. These servers could work by mirroring their published Missions, but they could also exchange information about available Missions. If these servers act independently this could lead to a situation where a user connects simultaneously to several Syndicate servers. Security considerations are necessary, especially when additional servers could be added dynamically.

Integration of service patterns

Missions that are based on the content pattern or on the integration pattern make use of external services. Such services often work on similar principles and could be categorized by service patterns.

A standard example is the use of a formula where a user enters requested values and after submitting, receives a corresponding result. This behavior of entering data, submitting and receiving could be generalized and addressed with a service pattern.

Another example is the use of only a portion of a regular web site. For example web sites with constantly changing content such as weather

information, could be taken apart and only the area of interest, such as the weather image, could be kept for further processing.

These patterns could lead to an environment where developers could elide code programming but use interactive facilities to create new Missions.

Programming by demonstration

A more generalized version of the suggestion above is programming by example [79]. Repetitive operations such as clicking through a set of pages or formulae in order to receive the desired result could be recorded and a Mission could be generated automatically with that information. Or, in a more sophisticated version, a Mission could be provided that could play back such a recording of activities when the Mission has been activated. This could greatly assist the development of other devices such as mobile phones or personal digital assistants and condense a large number of Missions that act in a similar way.

A carefully designed "Playback Mission" and "Action Recorder Mission" would allow users to create and store their personal Missions with little effort.

16.5.2 Security enhancements

Syndicate is implemented as a prototype and security is implemented on a elementary level. More meticulous security policies can be realized in future work. The following suggestions introduce technical prospects.

Web of trust

An authorization environment is definitely necessary to control who is submitting new Missions and to supervise the quality of handed in code. Encrypted checksums could guarantee that a Mission has not been tampered with, which is essential when Syndicate is to be used in the public internet.

An implementation of a trusted peer to peer community between Syndicate servers and users contributing to the Syndicate community could enhance security. With a peer to peer implementation it would be possible to omit central administration of security information. In areas where security is critical, it would be possible to limit the usage of Syndicate to a local environment.

Syndicate sandbox

A Syndicate Sandbox could protect users from potential malicious code. The sandbox could tightly control the set of resources which Missions can have access to. This allows Missions from untrusted users to be run safely [131].

16.5.3 Reliability

Automatic test cases

The specification of an interface for test cases could lead to automatic integrations of system tests. These could include Mission behavior and performance tests as well as strategies for choosing alternatives when contacting external service providers. Automatized testing at regular intervals would considerably improve the overall quality of the system.

Logging

In order to realize where the Syndicate system can take advantage of improvements, a fine grain logging mechanism could provide detailed information on a service level basis. Such a logging mechanism could serve performance and usage information which can be essential to register statistics about implemented Missions. Statistics could influence service selection strategies and most popular Missions could be noted at the community page and attract other users.

Caching strategies

The implementation of caching strategies on the Syndicate server could improve performance, especially when several users make use of the same Mission.

16.5.4 Usability

Support for other devices

Other devices such as mobile phones or personal digital assistants could be better supported. Due to the small screens it would be a great enhancement if Missions could be run seamlessly on such devices. Users of mobile devices usually do not browse the web, but they want instant access to information such as timetables, schedule calendars, product

or address information and so on. This desire is often tied to repetitive operations until the final output is delivered.

A Mission that executes several necessary steps at once would be a great relief and make mobile devices much more productive.

Semantic web

The Semantic Web targets for a more meaningful web, with the intention to make it easier for programs to process information. The result should be a web which is more useful and approachable.

The semantic web and semantic web services themselves are arriving but a meaningful description of the web is still far from being used globally. Nevertheless, it could be very interesting to take advantage of standardized metadata and generate more automation in the development and usage of new Missions [132] [133] [134].

For example, Missions that make use of external web services could profit from automatically tracked down alternative service providers. A recurring process on the Syndicate server could constantly update the list of service providers and store additional information such as response time. This operational behavior would be invisible to Mission users but they would profit from better service quality.

If the scope of the semantic web is taken a step further and if data and processes could be described in a more natural, standardized way, it would open up new windows of opportunities. Not only could the process of finding decentralized Missions for a particular purpose be simplified, but the creation of Missions could also be brought into another sphere. Missions that integrate other web services could be composed by describing what purposes should be included and in which way they are weaved together.

Ongoing Syndication

Syndication of a web page currently only holds for the page which the Syndicator has been activated for. As soon as the web browser loads another page, for example through clicking on a hyperlink, the freshly loaded page has to be syndicated again to make use of Missions. The Syndicator could be changed in such a way that with the first loading of a page all hyperlinks on the page were replaced. The replacement would include another activation of the Syndicator again after a new page has

been loaded. This behavior could also be realized through a static loaded Mission.

A feasibility study has already successfully been implemented, but it needs further refinements to work properly with all variations of hyperlinks.

Mission notifier

Users should not have to be concerned with constantly checking if new Missions are available on the Syndicate server. A Mission could be implemented that notifies the user when new Missions have been published to the Syndicate server. The Mission could be personalized in order to limit the notification to Missions based on a specific type of interest.

16.6 Vision for the future of the web

The World Wide Web has changed significantly since its formation in 1990. Latest developments show that the semantic web has aroused public interest and might occupy a central position in the foreseeing future. The technical foundation has already been laid, as Tim Berners-Lee commented in 2001: "The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [135]

Beside the undoubted continuation towards a more meaningful version of the current web, my personal vision is twofold: First, I see a trend towards three-dimensional structures, and existing virtual worlds already point out the possibilities. The web, in its three-dimensional version could be shaped constantly by participating users.

Secondary, I see the future of the web in conjunction with new devices and infrastructures much in the sense as the mobile phone has captured the modern world. The web will be everywhere and permanent access to the web will be guaranteed. Along with the extension of the web, a distinct location of applications and data will vanish. This would give everyone the opportunity to use ones personal environment independent of device and location.

I will finish with a quote from Google CEO Eric Schmidt, participant at the Seoul Digital Forum 2007, where he was asked to define web 3.0 by an audience member:

"My prediction would be that Web 3.0 would ultimately be seen as applications that are pieced together [and that share] a number of characteristics: the applications are relatively small; the data is in the cloud; the applications can run on any device — PC or mobile phone; the applications are very fast and they're very customizable; and furthermore the applications are distributed essentially virally, literally by social networks, by email. You won't go to the store and purchase them. That's a very different application model than we've ever seen in computing ... and likely to be very, very large." [136]

Bibliography

[1] Hitwise Intelligence, "Myspace moves into first position for all internet sites," Hitwise, 300 Park Avenue South, New York, Tech. Rep., July, 11 2006.

[2] R. Gonda, "Web 2.0: Web applications vs. desktop applications," *AjaxWorld*, January, 26 2006.

[3] A. Kadiyala, "Are You Ready for Mashups? Their real value is in the enterprise," *SOA World magazine*, June 2007.

[4] H. Adams, A. Arsanjani, G. Booch, S. Bose, D. F. Ferguson, D. K. Jackson, C. Lawrence, C. Lau, A. R. Szakal, D. Wolfson, and B. Woolf, "Why and when should you choose SOA?" *Insight and outlook*, November 2005. [Online]. Available: http://www-128.ibm.com/developerworks/library/ar-itio1/. cited 22.10.2007

[5] S. Greenberg, *What is Thin Client Computing?*, 3801 Schuylkill Road, Spring City, PA 19475-1529, July 2000.

[6] T. S. Perry, "John Gage: He IS The network," *IEEE, Spectrum Cateers*, May 2004.

[7] *Web services: beyond the hype*, vol. 35, 2002.

[8] Y. V. Natis, "Service-oriented architecture scenario," Gartner, 56 Top Gallant Road, Stamford, CT 06902-7700, U.S.A, Tech. Rep. AV-19-6751, April 2003.

[9] B. Roch, "Monolithic kernel vs. microkernel," TU Wien, Tech. Rep., 2004.

[10] A. S. Tanenbaum and A. S. Woodhull, *Operating Systems Design and Implementation*, 3rd ed. Prentice Hall, January 2006.

[11] Y. V. Natis and M. Pezzini, "Predicts 2003: SOA to stir up application server market," Gartner, Tech. Rep. SPA-18-8377,

December 2002. [Online]. Available: http://www.gartner.com/DisplayDocument?doc_cd=111837&ref=g_fromdoc. cited 22.10.2007

[12] J. Childers, "Monolithic software decimates it budgets," September 2003. [Online]. Available: http://business.newsforge.com/article.pl?sid=03/09/19/032238&tid=33&tid=3&tid=31. cited 22.10.2007

[13] M. Liebow, "Service oriented architectures conference." The Open Group, October 2005.

[14] R. W. Schulte and Y. V. Natis, "'Service Oriented' Architectures," Gartner, Tech. Rep. 1, April 1996.

[15] C. Lapham, "The cutting edge," *Computer-Mediated Communication Magazine*, vol. 2, no. 7, p. 4, July 1995.

[16] R. Kalakota and A. B. Whinston, *Electronic commerce: a manager's guide.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[17] B. Acohido, "Amazon moves to front line of shaping 'web services'," *USA Today*, September, 30th 2003.

[18] S. Haag, M. Cummings, and D. J. McCubbrey, *Management Information Systems for the Information Age*, 4th ed., ser. 0072819472. McGraw-Hill Companies, 2003.

[19] T. O'Reilly, "Web2.0 compact definition." [Online]. Available: http://radar.oreilly.com/archives/2006/12/web_20_compact.html." cited 20.6.2007

[20] D. Nickull, D. Hinchcliffe, and J. Governor, *Web 2.0 Design Patterns: What entrepreneurs and information architects need to know.* Adobe Dev Library, 2007.

[21] T. O'Reilly, "What is web 2.0. design patterns and business models for the next generation of software." [Online]. Available: http://www.oreillynet.com/lpt/a/6228#mememap." cited 20.6.2007

[22] F. Westphal, "Mashups: Remix me!" *OBJEKTspektrum*, vol. 2, 2007.

[23] E. Ort, S. Brydon, and M. Basler, "Mashup styles, part 1: Server-side mashups," *Sun Developer Network*, May 2007. [Online]. Available: http://java.sun.com/developer/technicalArticles/J2EE/mashup_1/. cited 22.10.2007

[24] D. Hinchcliffe, "Making the Most of the Web: Creating Great Mashups." [Online]. Available: http://web2.wsj2.com/making_the_most_of_the_web_creating_great_mashups.htm." cited 22.10.2007

[25] D. Merrill, "Mashups: The new breed of web app," IBM, Tech. Rep., August 2006.

[26] Wikipedia, "Mashup (web application hybrid)." [Online]. Available: http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)." cited 18.7.2007

[27] J. C. Perez, "Vint Cerf on google's challenges, aspirations," *Computerworld*, November 2005.

[28] "Mashups, apis and the web as platform." [Online]. Available: http://www.programmableweb.com/." cited 18.8.2007

[29] P. Web, "List of new mashups." [Online]. Available: http://www.mashupFeed.com/." cited 28.9.2007

[30] "Google maps." [Online]. Available: http://maps.google.com/." cited 17.8.2007

[31] "Housingmaps." [Online]. Available: http://www.housingmaps.com/." cited 18.7.2007

[32] "Traffic map of new york city." [Online]. Available: http://local.alkemis.com." cited 17.8.2007

[33] "Chicago crime map." [Online]. Available: http://www.chicagocrime.org/map/." cited 17.8.2007

[34] Microsoft, "Active server pages." [Online]. Available: http://www.asp.net/." cited 18.7.2007

[35] A. Systems, "Coldfusion." [Online]. Available: http://www.adobe.com/products/coldfusion/." cited 18.7.2007

[36] Sun, "Javaserver pages technology." [Online]. Available: http://java.sun.com/products/jsp/." cited 18.7.2007

[37] L. LassoSoft, "Middleware and application server." [Online]. Available: http://www.lassosoft.com/." cited 22.10.2007

[38] "Php, free software for producing dynbamic web pages." [Online]. Available: http://php.net/." cited 18.7.2007

[39] "Php usage statistics." [Online]. Available: http://php.net/usage.php." cited 18.7.2007

[40] Mozilla, "Spidermonkey javascript-c engine." [Online]. Available: http://www.mozilla.org/js/spidermonkey/." cited 18.7.2007

[41] Caucho, "Java/php application server." [Online]. Available: http://caucho.com/." cited 18.7.2007

[42] Mozilla, "Rhino: Javascript for java." [Online]. Available: http://www.mozilla.org/rhino/." cited 18.7.2007

[43] Microsoft, "Jscript .net." [Online]. Available: http://msdn2.microsoft.com/en-us/library/72bd815a(vs.71).aspx." cited 18.7.2007

[44] "Smx: Server macro expansion." [Online]. Available: http://www.smxlang.org/." cited 18.7.2007

[45] Apache, "Server-side includes." [Online]. Available: http://httpd.apache.org/docs/1.3/howto/ssi.html." cited 18.7.2007

[46] D. Thomas, D. Hansson, L. Breedt, M. Clark, T. Fuchs, and A. Schwarz, *Agile Web Development with Rails*, 2nd ed. Pragmatic Bookshelf, December 2006.

[47] D. Flanagan, *JavaScript: The Definitive Guide*, 5th ed. O'Reilly & Associates.

[48] D. Goodman and B. Eich, *JavaScript Bible*, 3rd ed. Wiley, John & Sons, March 1998.

[49] T. Powell and F. Schneider, *JavaScript: The Complete Reference*, 2nd ed. McGraw-Hill Osborne Media, 2004.

[50] LaszloSystems, "Openlaszlo – the premier open-source platform for rich internet applications." [Online]. Available: http://www.openlaszlo.org/." cited 22.10.2007

[51] N. Klein, M. Carlson, and G. MacEwen, *Laszlo in Action*. Manning Publications, 2007.

[52] J. J. Garrett, "Ajax: A New Approach to Web Applications." [Online]. Available: http://www.adaptivepath.com/publications/essays/archives/000385print.php." cited 12.8.2007

[53] J. Eichorn, *Understanding Ajax: Using JavaScript to Create Rich Internet Applications*, 1st ed. Prentice Hall PTR;, August 2006, ch. 2, pp. 15–40.

[54] J. Snell, D. Tidwell, and P. Kulchenko, *Programming Web Services with SOAP*, 1st ed. O'Reilly & Associates, December 2001.

[55] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services. Concepts, Architectures and Applications*. Berlin Heidelberg New York: Springer, 2004, ch. 6, pp. 156–189.

[56] T. Frotscher, M. Teufel, and D. Wang, *Java Web Services mit Apache Axis2*, 1st ed. entwickler.press, April 2007.

[57] Sun Microsystems, "JAX-RPC." [Online]. Available: https://jax-rpc.dev.java.net/." cited 22.10.2007

[58] M. Casati, "Javascript SOAP client." [Online]. Available: http://www.codeproject.com/Ajax/JavaScriptSOAPClient.asp." cited 18.7.2007

[59] S. A. LePera, "Using the mozilla SOAP API." [Online]. Available: http://www.oreillynet.com/pub/a/javascript/synd/2002/08/30/mozillasoapapi.html?page=1." cited 18.7.2008

[60] R. Whitmer, "SOAP scripts in mozilla." [Online]. Available: http://lxr.mozilla.org/mozilla/source/extensions/webservices/docs/Soap_Scripts_in_Mozilla.html." cited 18.7.2007

[61] J. Snell, "Call SOAP web services with ajax." [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-wsajax/." cited 18.7.2007

[62] "PHP soap." [Online]. Available: http://sourceforge.net/projects/phpsoaptoolkit/." cited 22.10.2007

[63] "Pocket SOAP – web services on the move." [Online]. Available: http://www.pocketsoap.com/." cited 22.10.2007

[64] "Soap::lite for perl." [Online]. Available: http://soaplite.com/." cited 22.10.2007

[65] B. Sleeper, "The Evolution of UDDI," The Stencil Group, Inc., San Francisco, CA, Tech. Rep., July, 19 2002.

[66] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[67] S. P. Otto and T. Day, *A Biologist's Guide to Mathematical Modeling in Ecology and Evolution*. Princeton University Press, 2007.

[68] *Bringing speech acts into UMM*. Department of Computer and Systems Sciences, Stockholm University and Royal Institute of Technology. Forum 100, SE-164 40 Kista. Stockholm, Sweden: REA Technology Workshop, April 2004.

[69] K. Jucyte, K. Kevelaitis, and S. W. Park, "Web service implementation with soap and rest," Roskilde University, Tech. Rep., January 2006.

[70] G. Alonso and J. Koehler, "Introduction to the special theme "service-oriented computing"," *ERCIM News*, no. 70, pp. 14–15, July 2007.

[71] C. Y. Baldwin and K. B. Clark, *Design Rules: The Power of Modularity*. The MIT Press, March 2000, vol. 1.

[72] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller, "Automation and customization of rendered web pages," in *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM Press, 2005, pp. 163–172.

[73] A. Boodman. [Online]. Available: http://www.greasespot.net/. cited 12.8.2007

[74] M. Pilgrim, *Greasemonkey Hacks*, 1st ed. O'Reilly & Associates, November 2005.

[75] "userscripts.org." [Online]. Available: http://userscripts.org/." cited 12.8.2007

[76] "Greasemonkeyuserscripts." [Online]. Available: http://dunck.us/collab/GreaseMonkeyUserScripts." cited 12.8.2007

[77] K. Kazuyoshi, "Creammonkey – user scripting for safari." [Online]. Available: http://creammonkey.sourceforge.net/." cited 12.8.2007

[78] M. Solomon, "Pithhelmet – an ad-blocker for safari." [Online]. Available: http://www.culater.net/software/PithHelmet/PithHelmet.php." cited 28.8.2007

[79] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan, "Koala: Capture, share, automate, personalize business processes on the web," in *Proceedings of ACM CHI 2007 Conference on Human Factors in Computing Systems*. ACM Press, April 2007, pp. 943–946.

[80] J. Wong and J. I. Hong, "Making mashups with marmite: towards end-user programming for the web," in *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2007, pp. 1435–1444.

[81] J. Barnabe, "Stylish." [Online]. Available: http://userstyles.org/stylish/." cited 28.8.2007

[82] "Monkeygrease – the server-side greasemonkey." [Online]. Available: http://monkeygrease.org/." cited 12.8.2007

[83] "Mousehole, the scriptable proxy." [Online]. Available: http://code.whytheluckystiff.net/mouseHole." cited 22.10.2007

[84] "Privoxy web proxy." [Online]. Available: http://www.privoxy.org/." cited 12.8.2007

[85] T. Reenskaug, "The original MVC reports," Dept. of Informatics, University of Oslo, Blindern NO-0317 Oslo Blindern , NO-0317 Oslo, Norway, Tech. Rep., December 1979.

[86] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *ject Oriented Program.*, vol. 1, no. 3, pp. 26–49, 1988.

[87] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., ser. Addison-Wesley Professional Computing Series. Addison Wesley, 1994.

[88] R. M. Frank Buschmann (Author), H. Rohnert, P. Sommerlad, and M. Stal, *A System of Patterns.* Baffins Lane, Chichester, West Sussex PO19 1UD, England: Wiley, John & Sons, April 1999.

[89] W. Pree, *Design Patterns for Object-Oriented Software Development.* ACM, 1515 Brodway, New York: Addison Wesley, 1995.

[90] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison Wesley Signatur Series, 2004.

[91] Yahoo!, "Yahoo! Maps, Diving Directions, and Traffic." [Online]. Available: http://maps.yahoo.com." cited 26.10.2007

[92] "On-The-Go Supplement to the USB 2.0 Specification," USB Implementers Forum, Tech. Rep. 1.3, December 2006.

[93] C. Soanes and S. Hawker, Eds., *Compact Oxford English Dictionary of Current English*, 3rd ed. Oxford University Press, 2005.

[94] W3C, "Web services architecture." [Online]. Available: http://www.w3.org/TR/ws-arch/." cited 2.10.2007

[95] F. Jared, "Greasemonkey script." [Online]. Available: http://userscripts.org/scripts/show/9176." cited 16.8.2007

[96] L'OcuS, "Table sort." [Online]. Available: http://userscripts.org/scripts/show/2118." cited 18.7.2007

[97] "Amazon.com: Online shopping." [Online]. Available: http://www.amazon.com/." cited 22.10.2007

[98] IMS Global Learning Consortium, Inc., "Open specifications for interoperable learning technology." [Online]. Available: http://www.imsproject.org/." cited 22.10.2007

[99] S. Rizzotti and H. Burkhart, "Web-based Test and Assessment System: Design Principles and Case study," in *Web-Based Education*, V.Uskov, Ed., IASTED. CA, USA: ACTA Press Anaheim, January 2006, pp. 37–42.

[100] ——, "(208-0887) WEB-BASED TEST AND ASSESSMENT SYSTEM: DESIGN PRINCIPLES AND CASE STUDY," *Advanced Technology for Learning*, vol. 3, 2006.

[101] ——, "528-098 There is More than One Correct Solution: From Simple Questions to Explorative Simulations with Progress Maps," in *Computers and Advanced Technology in Education*, V.Uskov, Ed., IASTED. ACTA Press Anaheim, October 2006.

[102] "Dapper: The data mapper." [Online]. Available: http://www.dapper.net/." cited 12.8.2007

[103] S. Stephenson, "Javascript framework." [Online]. Available: http://www.prototypejs.org/." cited 12.8.2007

[104] B. Perry, "Prototype: Easing ajax's pain." [Online]. Available: http://www.xml.com/pub/a/2006/04/05/prototype-javascript-ajax.html." cited 12.8.2007

[105] J. Resig, "jQuery: The Write Less, Do More, JavaScript Library." [Online]. Available: http://jquery.com/." cited 12.8.2007

[106] "Dojo – the javascript toolkit." [Online]. Available: http://dojotoolkit.org/." cited 12.8.2007

[107] "script.aculo.us – web 2.0 javascript." [Online]. Available: http://script.aculo.us/." cited 12.8.2007

[108] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws.* Wiley, October 2007.

[109] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services. Concepts, Architectures and Applications.* Berlin Heidelberg New York: Springer, 2004, ch. 2, pp. 63–64.

[110] T. Calishain, "Bookmarklets boost web surfing," *PC Magazine*, vol. 23, no. 2, February 2004.

[111] "VeriSign, Inc. – SSL Certificates." [Online]. Available: http://www.verisign.com." cited 12.8.2007

[112] J. Allsopp, *Microformats: Empowering Your Markup for Web 2.0*. friends of ED, March, 26 2007.

[113] Apache, "Xmlbeans." [Online]. Available: http://xmlbeans.apache.org/." cited 12.8.2007.

[114] P. Szinek, "Web extraction framework written in Ruby." [Online]. Available: http://scrubyt.org/." cited 11.11.2007

[115] Parakey, "Firebug – web development evolved." [Online]. Available: http://www.getfirebug.com/." cited 15.8.2007

[116] R. Brown, "Aardvark firefox extension." [Online]. Available: http://karmatics.com/aardvark/." cited 22.10.2007

[117] P. Smith, "Top 10 firefox extensions to avoid. just because an extension is popular doesn't mean it belongs in your web browser," *Computerworld*, April 207.

[118] M. Pilgrim, "Avoid common pitfalls in greasemonkey." [Online]. Available: http://www.oreillynet.com/pub/a/network/2005/11/01/avoid-common-greasemonkey-pitfalls.html?page=1." cited 29.10.2007

[119] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, "Going beyond the sandbox: an overview of the new security architecture in the javatm development kit 1.2," in *USITS'97: Proceedings of the USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, 1997, pp. 103–112.

[120] O. Hallaraker and G. Vigna, "Detecting malicious javascript code in mozilla," in *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*. Los Alamitos, CA, USA: IEEE, 2005, pp. 85–94.

[121] V. Anupam and A. Mayer, "Secure web scripting," *IEEE Internet Computing*, vol. 02, no. 6, pp. 46–55, 1998.

[122] A. Oram, "From p2p to web services: Trust," *O'Reilly xml.com*, April 2004.

[123] W. Jonker and M. Petkovic, Eds., *PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web*, vol. 3178, VLDB. Heidelberg: Springer, August 2004.

[124] F. Leymann, *Web Services Flow Language (WSFL 1.0)*. IBM Software Group, May 2001.

[125] "Creative commons." [Online]. Available: http://creativecommons.org/." cited 10.11.2007

[126] T. Winterberg, "BPEL wird erwachsen..." *Javamagazin*, vol. 7, pp. 22–29, July 2007.

[127] F. Leymann, D. Roller, and S. Thatte, "Goals of the BPEL4WS specification," *CoverPages*, August 2003.

[128] OASIS, "Web Services Business Process Execution Language version 2.0." [Online]. Available: http://www.oasis-open.org/committees/download.php/23665/wsbpel-v2.0-OS.htm." cited 12.7.2007

[129] Eclipse, "Eclipse-based BPEL 2.0 Designer." [Online]. Available: http://www.eclipse.org/bpel." cited 10.7.2007

[130] C. Pautasso, T. Heinis, and G. Alonso, "Jopera: Autonomic service orchestration," *IEEE Data Eng. Bull.*, vol. 29, no. 3, pp. 32–39, 2006.

[131] R. S. Cox, J. G. Hansen, S. D. Gribble, and H. M. Levy, "A safety-oriented platform for web applications." Oakland, CA: Proceedings of the 2006 IEEE Symposium on Security and Privacy, May 2006.

[132] N. Chase, "The ultimate mashup – Web services and the semantic Web, part 1: Use and combine Web services." [Online]. Available: http://www-128.ibm.com/developerworks/edu/x-dw-x-ultimashup1.html." cited 22.10.2007

[133] T. B. Passin, *Explorer's Guide to the Semantic Web*. Manning Publications, March 2004.

[134] D. McGuinness and F. van Harmelen, "W3c working draft: Feature synopsis for owl lite and owl." W3C, Tech. Rep., July 2002.

[135] T. Berners-Lee, "The semantic web," *Scientific American*, May 2001.

[136] E. Schmidt, "Web 2.0 vs. web 3.0," Seoul Digital Forum, Korea, Tech. Rep., May 2007. [Online]. Available: http://www.youtube.com/watch?v=T0QJmmdw3b0. cited 21.11.2007

[137] C. Fallon and S. Brown, Eds., *E-Learning Standards: A Guide to Purchasing, Developing and Deploying Standards-Conformant E-Learning*. St. Lucie Press, 2002.

Glossary – Web 2.0 and Beyond

AJAX
(Asynchronous JavaScript and XML). Ajax is a cross-platform web development technique used for creating interactive web applications. The intent of Ajax is to make web applications more responsive by exchanging data asynchronously in the background. 35, **35**, 130, 131, 137, 142, 143

API
An application programming interface (API) is the specific method prescribed by an application program by which a programmer can make requests of another application. 23, 24, 26, 127

ATOM
The Atom format is a standardized XML language used for web feeds. 35

B2B
B2B is a term used to describe electronic commerce transactions between businesses. 19

B2C
B2C describes activities of electronic commerce serving end consumers with products and/or services. 19

BEEP
A framework for creating network application protocols including mechanisms such as: asynchronous communications, transport layer security, peer authentication, channel multiplexing on the same connection, message framing, channel bandwidth management, and other network features. 40

Bookmarklet
A bookmarklet is a small JavaScript program stored within a bookmark in a web browser, or stored within a hyperlink on a web page. [110]. 52, 129, **133**, 135, 143

BPEL
Business Protocol Execution Language for web services (BPEL) is an XML-based business process modeling language that is executable. 161

C2B
C2B describes electronic commerce in which consumers offer products and services to companies. This business model is a reversal of traditional business model where companies offer goods and services to consumers (B2C). 19

C2C
C2C describes electronic commerce where transactions between consumers run through a third party. 19

Consigliere
Controlling unit where the application logic is implemented. **82**, 88, 89, 110

CRISPY
Communication per Remote Invocation for different kinds of Services via ProxYs . Provides a single point of entry for remote invocation for a wide number of transports: eg. RMI, EJB, JAX-RPC or XML-RPC. 40

DHTML
DHTML is a combination of technologies used to create dynamic and interactive web sites. It consists of a static markup language, a client-side scripting language, a presentation language, and the Document Object Model. 32, 34

DOM
The Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML and related formats. **83**, 130, 137, 138

EAI
Enterprise Application Integration is the process of integrating multiple applications that were independently developed, may use incompatible technology, and remain independently managed. Syndicate can bee senn as a dynamic EAI. 36

Favelet
see Bookmarklet. 133

Global XML Web Services Architecture
Extension of SOAP with security and routing mechanisms. 40

Hessian Binary Web Service Protocol
Binary protocol. 40

IMS Question and Test Interoperability specification (IMS QTI)
An open standard that describes a data model for the representation of question and assessment data and their corresponding results [137] [98]. 113

Injector
The injector is responsible for the manipulation of the current loaded web page. 65, **83**, 97, 104

JavaScript
JavaScript is a scripting language most often used for client-side web development. It is a dynamic, weakly typed, prototype-based language with first-class functions [47]. 31, 32, **32**, 47, 49, 51, 52, 94, 129, 130, 141, 143, 144, 151

JavaScript security
JavaScript has a long and inglorious history of atrocious security holes. Beside implementation errors there are numerous ways in which scripts can affect the user's execution environment without violating any security policies [49]. 135

JSON
JavaScript Object Notation (JSON) is a lightweight data-interchange format with the purpose to be easy to read and write for humans and easy to parse and generate for machines. 35, 79

JSON-RPC
Lightweight remote procedure call protocol which handles only a few data types and commands and allows bidirectional communication between the service and the client. 40

Loose coupling
Loose coupling is an architectural principle to build applications that promote loose coupling among components. Loose coupling can mean independence in terms of time or format. 16, **17**, 37, 44, 66, 130, 143

Manifold
Manifolds are known as abstract mathematical spaces. In this context a manifold, in contrast to a monolithic system, signifies a further specification of a service-oriented software architecture whith additional restrictions on participating components. 61, **61**, 62, 66, 69, 127, 157, 159

Mashup
A web application that combines data from more than one source into a single integrated tool [22]. **23**, 24, 25, 50, 60

Microformat
A microformat (uF) is a data markup language to semantically annotate existing HTML or XML data. This approach is intended to allow automated extraction and processing of data portions stored within web pages (see also: RDF). Microformats are defined for particular subjects [112]. 136

Mission
A task that changes the appearance, the content or the behavior of a web page. 65, 72, **75**, 93, 103, 141

Mission patterns
A mission patterns is either a presentation, content, or integration pattern. 58, 65, 75, 94, 101

Mission properties
Environmental settings which control Mission specific parameters. 64, 65, 72, 83, 94, **95**, 99, 100, 121, 130, 141, 142

Mission trigger
Event, which initiates the invocation of a Mission. Triggers can be static, periodical, or user-generated. 65, 72, **76**, **95**, 99, 106, 114, 123, 128, 138, 141, 144, 147

Monolithic system
The term derives from an architectural style meaning a construction out of a single piece of material. In this context, a monolithic system or monolithic architectures signifies that all processing and data reside on the same system. **15**, 18, 43, 61, 63

MVC
The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller [85] [86] [87]. 57

REST
Representational State Transfer. An architecture style of networked systems based on the idea to see Web applications as a virtual state-machine [66]. 35, 39, **39**, 40, 106, 136, 142

RSS
(Really Simple Syndication) is an XML-based document format for the propagation of web content so that it can be republished on other sites or downloaded periodically. News headlines, weblogs, and the exchange of other timely information such as software release notes are usage examples. 26, 35, 53, 137, 138

Sandbox
A sandbox signifies a security mechanism for a shielded environment where programs can safely be executed. It is mainly used to execute untested or untrusted code. 163

Scout
The Scout is responsible for information gathering and can make use of semantic or structural scraping techniques. 65, 77, **80**, 95, 97, 104, 141, 145

Semantic Web
The Semantic Web approach develops languages for expressing information in a machine processable form. Semantic Web technologies include RDF, OWL, SWRL, SPARQL, GRDDL. 164, 165

Server detour
A server detour signifies an additional roundtrip of a communication vi the Syndicate server. 65, 72, **78**, 141, 145

Service-oriented architecture (SOA)
A design philosophy to achieve loose coupling among interacting software agents [14]. 3, **16**, 18, 57, 72

Signed script
Technology used in Mozilla Web browsers to access expanded privileges in JavaScript applications. 135

SOAP
Simple Object Access Protocol, also known as the service-oriented architecture protocol. An XML-based messaging framework that allows messages being exchanged over a variety of underlying protocols [54]. 35, 37, **37**, 39, 40, 79, 136, 142

Syndicate
Syndicate is first, the name of the thesis and covers the generic principle of several small tasks working together to address a common aim. Second, when used as an adverb, it describes the state of a loaded web page. 2, 71

Syndicator
The Syndicator is responsible for the syndication of a loaded web page and acts as the kick-off of the whole Syndicate system. 65, **72**, 143, 144, 164

UDDI
Universal Description, Discovery and Integration protocol. The basic goal of UDDI is a framework for describing and discovering services and service providers. 37, **38**, 40

VoIP
Voice over Internet Protocol, also called VoIP is the routing of voice conversations over the Internet or through any other IP-based network. 116

Web-based service
Web-based service, or web service is a software system designed to support interoperable machine to machine interaction over a network. **19**, 65, 85, 89, 113, 136, 137, 139

Web 2.0
A term created by Tim O'Reilly describing the ways software developers and users use the web. Web 2.0 does not describes any technical specifications but refers to services instead of packaged software, participation architectures, and remixable data sources [21]. **22**, 23, 25

WSDL
Web Service Description Language (WSDL). WSDL is an XML document used to describe web services and specifies the location of the service and the operations or methods the service exposes. 37

WSFL
Web services flow language. An XML language for the description of web services compositions [124]. 153

XINS
Open-source Web Services technology, supporting SOAP, XML-RPC and REST that includes a Java-based implementation framework. 40

XML-RPC
Specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. 40

XPath
(XML Path Language) is an expression language for addressing portions of an XML structured document, or for computing values based on the content of an XML document. 137, 138

XSS
Cross-site scripting (XSS) is technology which allows code injection into the web pages. Examples of such code include HTML code and client-side scripts. 129, **131**