FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Context-aware Security Analysis of Mashups

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Heidelinde Hobel
Matrikelnummer 0625620

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Mitwirkung: Mag. Dipl.-Ing. Dr. Amin Anjomshoaa

Wien, 06.11.2012

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Context-aware Security Analysis of Mashups

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Business Informatics

by

Heidelinde Hobel
Registration Number 0625620

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Assistance: Mag. Dipl.-Ing. Dr. Amin Anjomshoaa

Vienna, 06.11.2012 (Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Heidelinde Hobel
Josef Sirowy Str. 20, 2231 Strasshof

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasserin)

Acknowledgements

I would like to thank Dr. Edgar Weippl for the chance to work under his supervision on this topic. Especially, I want to thank him for giving me the opportunity to work at the SBA Research.

I also wish to thank Dr. Amin Anjomshoaa for his idea of this topic as well as his help and encouragements in my work.

Furthermore, I want to express my gratitude to my family. Without their support, encouragement and trust I would never have completed my studies.

At last, I would like to express my thanks to my colleagues at the university. With them I spent a great time in my life and without them my studies would have been a lot more challenging.

Abstract

Mashups offer various possibilities by providing a simple way to process data according to the needs of the user. Furthermore, from the processed data a new, personalized, and reusable data resource is created. In business context, mashups are denoted as Enterprise Mashups and are designed in a way that staff members could easily develop their own applications, with minimal or even without support of the software engineering staff. The principal purposes of the application of Enterprise Mashup solutions are at the one hand the automation of daily tasks and on the other hand facilitating the implementation of sophisticated applications. Hence, mashup solutions are aimed to exploit the full potential of short-time, situational, ad-hoc, tactical, and individual (Hoyer et al., 2008) software development as well as one of the main strength of the concept is that the main effort of development is shifting into the end-users' hands.

The architecture of mashups follows the paradigm of a pipeline that concatenates operations such as select columns, merge, and filter. Hence, the data is sequentially processed by the designed data processing flow. The operations are predefined, or in the case of Enterprise Mashups, provided by the development staff. This thesis investigates how the data is manipulated with predefined operations from the end-user's perspective. Then even with a single or just a short sequence of simple operations, sensitive data records could be created. Considering that the enterprises' data is one of the most valuable assets and the introduction of mashups into daily business, it follows that this data is exposed to unregulated data processing. Hence, questions in the area of security and data privacy arise. In order to answer these questions, this thesis provides a theoretical survey about mashup fundamentals as well as Enterprise Mashups and proposes a semantical approach that describes and restricts mashups' architectures.

In order to illustrate the whole approach, a Proof of Concept prototype shall clarify the proposed approach based on use cases for a typical peer review system. The implemented Web application provides the capability to upload mashup scripts that are written in the Enterprise Mashup Markup Language (EMML). The mashup script could afterwards be automatically evaluated for compliance with predefined policies by a reasoner. These policies are restrictions that should regulate the mashups' architecture. The evaluation part illustrates experimentally how the reasoner enforces the embedded restrictions and computes an explanation in the case that a rule is violated.

Kurzfassung

Das Konzept von Mashups bietet eine Vielzahl an Möglichkeiten, um auf einfache Weise Daten aus verschiedenen Quellen zu verarbeiten und daraus eine neue, personalisierte und wiederverwendbare Daten-Ressource zu schaffen. Mashups in Unternehmen werden auch Enterprise Mashups genannt und sind dazu optimiert, dass Angestellte einfach ihre eigenen Applikationen entwickeln können, um ihre täglichen Aufgaben zu automatisieren oder zu erleichtern, vorzugsweise ohne oder mit nur minimalem Support von der Software Engineering Abteilung.

Die Architektur eines Mashups bildet die Form einer Pipeline, die Operationen wie zum Beispiel Filtern, Selektieren oder Zusammenfügen miteinander verbindet, sodass die Daten sequentiell verarbeitet werden. Die Operationen sind vordefiniert, oder im Fall von Enterprise Mashups durch die Software EntwicklerInnen des Unternehmens bereitgestellt. In dieser Arbeit betrachten wir aus der Perspektive der EndanwenderInnen, wie sie diese Daten nach ihren Wünschen manipulieren können. Schon durch den Einsatz von einer oder einer Reihe von simplen Operationen, zum Beispiel Filtern, können sensitive Datensätze erzeugt werden. Da die Daten eines Unternehmens wertvolle Vermögensbestände sind, stellt sich bei unregulierter Verarbeitung von Daten die Frage des Datenschutzes. Dazu wird in dieser Arbeit ein theoretischer Background sowohl über die Grundlagen von Mashups als auch von Enterprise Mashups vermittelt und ein semantischer Ansatz vorgestellt, der darauf abzielt, die Architektur von Mashups zu beschreiben und zu restriktieren.

Im Rahmen dieser Arbeit wurde eine Web Applikation entwickelt, die anhand eines Peer Review Systems den vorgestellten Ansatz verdeutlichen soll. Dazu können Skripts von Mashups, welche in Enterprise Mashup Markup Language (EMML) geschrieben sind, hochgeladen und anschließend mit der Hilfe eines Reasoners automatisch auf die Konformität mit den vordefinierten Regeln getestet werden. Diese Regeln sind Restriktionen, die die Architektur eines Mashups regulieren sollen. Die Evaluierung zeigt experimentell wie der Reasoner die Restriktionen durchsetzt und, falls eine Regel verletzt wird, eine Erklärung berechnet und ausgegeben wird.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The mashup concept was introduced with the emergence of Web 2.0 and offers handy possibilities to use public APIs or reuse existing components and share the mashed up data resources with other users. According to (Ogrinz, 2009), the term mashup describes the creation of new contents through a recombination of existing contents of third parties or the combination of external sources with own content. Today, the Web offers a variety of mashup editors of different providers which could be used to create mashups without special programming knowledge in an intuitive and easy way, like Yahoo Pipes. Thus, almost everybody can create a mashup which will then takes over the specific task for what it is designed to do. For instance, the mashup can be designed to gather data from different news feeds, filter them according to the preferences of the user and then create a graphical interface for the data with the help of a predefined style template function or module.

In the organizational context the mashup could also be designed to generate value, particularly in combination with business processes. However, in the business environments such user mashup editors are generally proscribed, because of issues like security, privacy, accessibility, usability and performance. Some approaches like the JackBe Mashup (JackBe) solutions or IBM Mashup Center (IBM) are tending towards using pre-defined widgets, which could be combined by the end user to new situational applications. The general goal is to shorten the long tail of user needs and provide appropriate software for everyone which allows the user to write or design his-/her-self own solution. Permitting the user to design his/her own applications has therewith a lot of advantages, to mention only three of the most common, (1) the specialists in programmers can concentrate on complex application, (2) the staff member who needs a simple automated procedure doesn't have to wait till a programmer has time to take over the programming order and (3) even the most simplest task can be automated so that a user doesn't have to waste time for time consuming and unnerving tasks, like creating a nicely designed balance sheet for each month.

Nonetheless, in order that the user can create his own applications, the user needs access to the necessary data. By unlimited access to the enterprise data the question of security and privacy arises due to the fact that, as a consequence of the mashup architecture, the possibility of confidential data leaks increases (Bader et al., 2010). To emphasize the problem we have to keep the following three facts in mind:

> *"1) Mashups are usually created by non-experts who are unaware of underlying data structure and security threats;*
> *2) Mashups can be shared and reused within organizations and also with users outside;*
> *3) Mashups are created by using data and information from known and unknown sources in- and outside the company."* (Bader et al., 2010)

Following the idea of Bader et al. (2010) the valuable data has to be protected with appropriated security models. Some approaches like from Cherbakov et al. or Trojer et al. (2009) make use of Service-Oriented Architecture (SOA) to prevent security and privacy threats. However, the set-up of the platform as well as design and maintenance of such compensable services is, at the one hand, a great deal of effort for the trained programming staff and, at the other hand, maybe another approach would be better suited to exploit the full potential of the mashup concept.

Thus, the security model has to care of the following according to the previous mentioned facts: according to (1) an unskilled user has to be controlled by designing a mashup due to his/her unawareness of data structures and security threats. Concerning the data structures such as relational databases or Application Programmable Interfaces (APIs) there are basically three conceivable ways: document the data structure so that the user understands how to use it; provide templates or predefined modules for data access; use data structures that are familiar for the specific user group, like the tabular structure of an excel or csv sheet which is familiar to the people who work with it every day. Concerning the security threats, the user has to be forced to comply with predefined security rules because otherwise some people will violate the restrictions. That could be on purpose or unintentionally due to unawareness and the additional work the security concerns will induce. Additionally, according to (2) and (3), the complete context has to be taken into account. Thus, it should be guaranteed that mashups supply data only to the right person with appropriate granted permissions which should also take the trustworthiness of the data into account. Furthermore, the violated rules have to be shown to the user in a way that is understandable so that he/she can correct them. At the same time these rules has to be understandable for a machine so that it can compute when a rule is violated and when not. One approach to solve these issues is the usage of Semantic Web technologies.

1.2 Methodology

The focus of this thesis is to show an approach which befits from mashups in organisational context without the possibility that unauthorized staff members or attackers can exploit sensitive data of the company. For the realization of the model, Semantic Web Technologies are used that allows formalization of the context, mashup structure and organisational policies. Furthermore, by using Semantic technologies, automated conclusions could be drawn from a reasoner. The reason why Semantic Web Technologies are used is basically that they provide an intuitive way for the logical formalization of rules that are understandable for humans as well as machines, thus a perfect interface for defining rules and dependencies from the company itself and then the automated reasoning from the implemented controlling tool.

For the sake of clearance the necessary background knowledge is explained in the theoretical part which should basically introduce the following three topics in such a detail that the later proposed security model could be comprehended and the advantages of the whole mashup application concept in organisations is understandable. At first the fundamentals of mashups in general will introduce how the mashup concept has emerged and a short survey is given about the types, techniques and tools that are available. Secondly, mashups will be considered in organizational context by showing what possibilities mashups provides for companies and why they are so attractive for them. The risks of the mashup's architecture in security and privacy related concerns are especially emphasized and builds the fundamental for the later introduced security model. At last, the Semantic Web Technologies will be introduced and explained due to their importance as formalization and reasoning tool in the proposed security model.

The second part is concerned with introducing the general idea of the whole security model, proposing how the mashup's architecture as well as security related polices can be described and restricted with Semantic Web Technologies. The model introduces a semantic approach that mitigates risk like data leakage and sensitive aggregation. Likewise, the approach introduces a way that describes from the end-user's perspective, which operations of a mashup are allowed, required or forbidden as well as in which sequence these operations have to be executed, so that the mashup distributes only allowed data to the user.

In order to prove the proposed model, the implemented prototype is introduced and three conducted experiments with the implementation are explained. The service layer of the prototype is based on an embedded ontology, that defines the polices for designing mashups. A reasoner is responsible for a human-readable explanation that informs about violated policies in the design of the target mashups. The implementation is designed as a Web application for testing uploaded scripts of mashups and is based on a peer review system.

1.3 Related work

Bader et al. (2010) and Anjomshoaa et al. (2009) address the general security, trust and privacy problems, which come along with the mashup architecture. They describe in detail, the general idea of mashups and their shift from only the casual sector to business supporting applications, the connecting security issues and provided the first hints in the direction of the usage of Semantic Web Technologies in mashup security concerns, that provide the idea for this thesis.

The shift of mashups to enterprise mashups should imply that the mashup security sector should be well equipped with new proposals and recommendations. However, due to the novelty of mashups that emerged with Web 2.0, in mashup security are very few academic publications available. Wang et al. (2007) introduced the idea of Microsoft MashupOS where a browser acts as multiprincipal operating system, in order to prevent *liability* and *overtrusting* concerns in client mashup creation. Miller et al. (2008) point out the possibilities in mashup development with capabilities and isolation of untrusted web applications and present Google Caja for the development of safe mashups, which integrate untrusted JavaScript code. De Keukelaere et al. (2008) introduced a security model for Web mashup applications, where they introduced the idea of isolated components in mashup development that could communicated over regulated and controlled channels.

However, a closely related topic to mashup security is privacy concerns in data-publishing, mostly due to the fact that most mashups are performing automated data processes and return the result to the executer of the mashup. Fung et al. (2010) provides a detailed survey about newest concepts and techniques for data privacy concerns by publishing for public or specified user groups. Likewise, Minami and Borisov (2010) and Liu and Yang (2012) provide detailed discussions and examples about inference attacks that have to be considered by publishing of sensitive information.

The research of ontologies started with representation aspects of ontology design just like in publication "Formal ontology, conceptual analysis and knowledge representation" from Guarino (1995). However, there is a trend towards using ontologies, meta data and semantics for security model and for defining policies. Panagiotopoulos et al. (2010) are using ontology-based privacy policies to enforce, among other things, access control in an intelligent environment.

The same is applied in mashup security research, Barhamgi et al. (2011) which proposes an declarative and privacy preserving approach to mashup data Web services by putting ontologies, meta data and ontology queries into practice. This approach is compared with the first mentioned security concepts similar to the approach that is proposed in this thesis. However, they are using the semantics in the data level whereas the proposed approach is describing the operations of a mashup. Another interesting approach is proposed by Beckman (2012), who discussed composability pattern with Language-integrated Query (LINQ) in order to satisfy, among others, security requirements.

1.4 Structure of the thesis

This thesis is structured in basically two main parts. The first part provides a survey about mashup fundamentals (chapter 2) as well as mashups in organizational context (chapter 3) that are also called Enterprise Mashups. One part of Mashups in organizational context is concerned with the security, privacy and trust implications of the mashups architecture. Additionally, in the first part, one section (chapter 4) provides detailed background information about semantics, ontologies and ontology languages. In the second main part the implemented Prototype will be introduced. The chapter Security model (chapter 5) will explain how the architecture of a mashup could be described and restricted with an ontology. The Proof of Concept (chapter 6) introduces a Web application that should present the approach, based on a peer review system as example. In chapter 7, the result of this thesis will be summarized.

Part I

Theoretical Part

CHAPTER 2

Mashup fundamentals

2.1 Web 2.0

In 2005 Tim O'Reilly coined the phrase 'Web 2.0' in his article *What Is Web 2.0* (O'Reilly). However, a clear definition for this term is missing. According to Murugesan (2007), Web 2.0 is more of *"an umbrella term"* for several *"technologies as well as behaviour models"* of end users and businesses. Although for businesses the term has changed to Enterprise 2.0 (cf. chapter 2.2).

Murugesan (2007) describe the term Web 2.0 as an "usage and technology paradigm". The technology part comprises the whole new and helpful developments, which turn the Web more interactive, collaborative and build a platform for many opportunities. The main components and services are blogs, Really Simple Syndication (RSS), wikis, tags as well as mashups. The usage part describes the changing behaviour of Web users and developers. The users could access content almost everywhere and easily contribute their knowledge and opinion. In fact, the social involvement changes from only consumers to real participants (cf. figure 2.1).

Figure 2.1 shows a classification of adult Web users in six behavioural categories according to Forrester Research. The classes start from "Inactives" that do not use the Internet for any kind of purpose. "Spectators" could also be named as only Internet consumers that uses the full content of the Web but do not contribute anything. "Joiners", "Collectors", "Critics" and "Creators" refers to the real participants of the Web. The classes build on the respectively upper classes. Starting with "Joiners" that join the social community like Facebook or Twitter, "Collectors" that subscribe to RRS Feeds and tag content, and "Criticts" that comment and rate other content like product offers, blog entries and add there opinion and knowledge to forums and wikis. "Creators" then add their own designed Websites, blog entries and upload photos, music and videos.

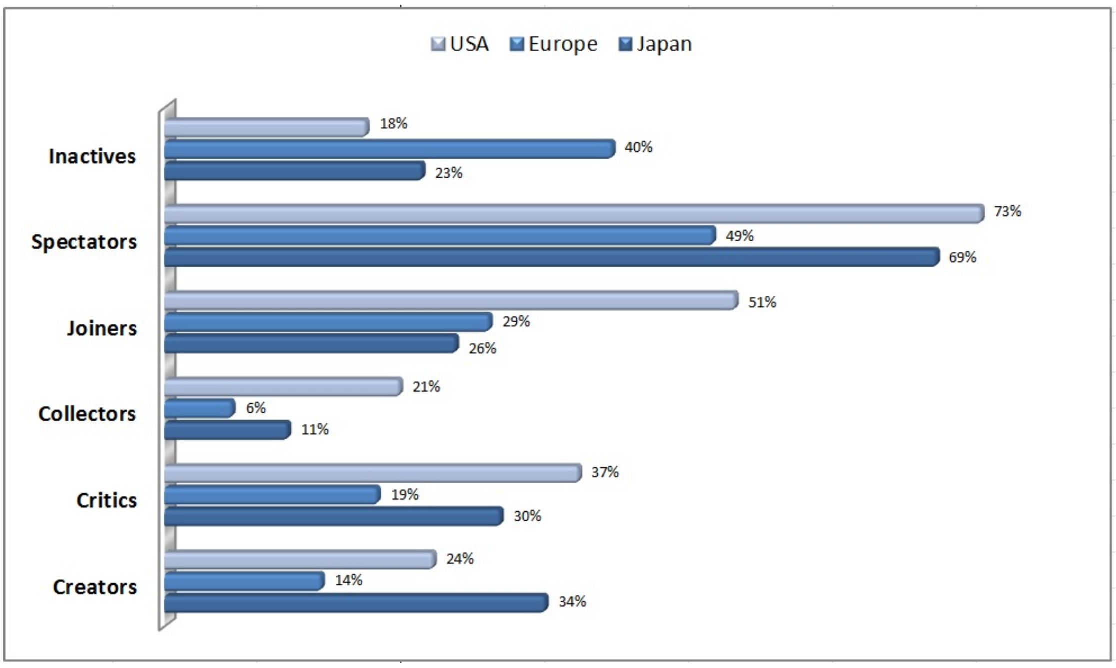


Figure 2.1: A geographical survey of Web behavioural groups. Data derived from (Forrester Research).

The diagram 2.1 shows that especially in USA and Japan the *Inactives* play only a minor role in the whole target group. The spectators take the front rank and also the creators, critics, collectors and joiners already play an important role too. According to Agichtein et al. (2008) the phenomena of the changing behaviour of Web users started already in the early 2000s. Thus, the *user-generated content* has become more and more popular, rather than traditional published content, which is created by only a small group of Web experts. Another term which is introduced is *social media*. So, in other words, the internet gets a communication medium for the social life and is evolving into social communities, which are well know in context of Facebook and Twitter.

An important incentive for the growing usage of the Web is that it is becoming more attractive. Thus, the user interfaces of the Web get more and more a rich design, which interact with end-user in an intuitive way. One important factor for that is the application of Rich Internet Applications (RIAs), which are designed to process the most data at the client's host and use the Internet for the sharing of information. (Bader, 2012)

In summary, Web 2.0 is only an all-embracing term for different technologies and a change in user behaviour, which makes Internet so flexible and so powerful in respect of content amount. However, there is no prospect of an end of this trend. The terms user-generated content as well as social communities are increasingly becoming popular.

2.2 Enterprise 2.0

Enterprise 2.0 could be described with the following sentence of the Association for Information and Image Management (AIIM):

> "Enterprise 2.0 is a system of web-based technologies that provide rapid and agile collaboration, information sharing, emergence and integration capabilities in the extended enterprise." (Frappaolo)

The main message of the citation is that the modern enterprise adopts social software and other Web 2.0 technologies for their own purposes. The first applications were blogs and wikis as an internal communication medium in project groups and departments. (McAfee, 2006) However, according to Silva et al. (2010) the integration of Web 2.0 technologies has no prospect of an end but rather gaining wide acceptance in the enterprise context. The main reason for this fact is that the staff members started demanding the same advantages of the Web 2.0 technologies inside their own company. Additionally, further advantages for the company itself are possible. For instance, the companies could design rich Web applications, increasing their public presence and personal image. Other important factors include general knowledge sharing inside and outside of the company, staff engagement and the building of situational applications in short time.

The Web 2.0 or Enterprise 2.0 age respectively, has created additional challenges for enterprises. Especially the security of valuable data and information is one key factor each company has to consider. Such security vulnerabilities can occur from one or more undiscovered or unresolved risks respectively. The next few points show a survey about some key security factors that emerge with Enterprise 2.0 in the viewpoint of Silva et al. (2010) with some extensions:

- *Trust*: Today it is common to install applications or widgets without consideration of security or legal factors. Additionally, almost everything which is posted on wikis, blogs, forums or in user recommendations is assumed to be trustworthy information. In the enterprise 2.0 environment this user behaviour is raising some concerns on security, trust and legal aspects. Concerning the copyright, at the moment it is simply a matter of trust that nobody removes the copyright information and passes off the used content as their own.

- *Sensitivie information leakage*: As already mentioned, the internet evolves to a comprehensive platform for knowledge sharing. Also employees of a company use this platform to share their personal opinions, knowledge or personal information such as their holiday pictures. People are not always fully aware almost all of this information could be exploited for several use cases. For instance, a secretary of a company could post in a social platform that his/her manager is in a bad mood lately. Such information could be used to work out the financial situation of the company.

- *Data leakage or destruction*: Along with Web 2.0 technologies several vulnerabilities of the network can emerge. According to Silva et al. (2010), "Web 2.0 sites inherently carry more risk than traditional Web sites because they let users upload content and require scripting capabilities - which can run code or carry malware - to function properly ... hackers have exploited Web 2.0 to launch worms that execute harmful operations outside the browsers, leaving the user unaware of their activities." With Web applications and other Web 2.0 technologies becoming increasingly popular, the companies provide a big target for attackers.

2.3 Mashups in general

The term mashup describes the combination of two or more sources. In earlier days the term was only used by the music industry, where the term mashup stands for a new song, which has been created by combination of two or more other songs (TheFreeDictionary). In the IT industry it could be defined by the following sentence:

> "A mashup is a user-driven micro-integration of Web-accessible data." (JackBe Corporation, 2008)

The sources of mashups could be quite different and the mashup's concept is emphasized with figure 2.2.

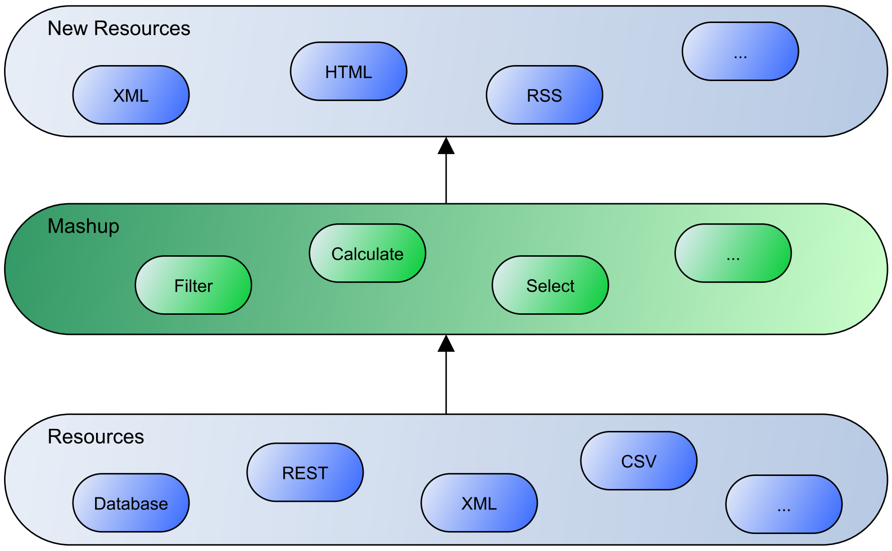


Figure 2.2: Mashup concept: A tool which mashes up data from several sources, process them according to the needs of the user and builds a new data resource. (Inspired by Zhiquan et al. (2011))

Thus, the whole idea of mashup programming is just to simply create something new with already available services or data. The chapter 2.4 will emphasize that approach by explaining the different types of mashups that are classified according to the intended purpose. However, in general the architecture of a mashup follows the paradigm of a pipe (Bader et al., 2010) (cf. figure 2.3). Each of the operations has at least one input variable and exactly one output variable. That approach could be translated in a script where each of the operations are sequentially processed. Following that approach, there already exists a number of different editors (cf. chapter 2.5) where even an inexperienced programming user can design his/her own application by just dragging and dropping predefined modules, which are basically the operations of the mashup, and connecting them with arrows to build the control flow. For instance, JackBe Presto (JackBe), a highly advanced mashup vendor, provides a visual editor for designing the mashup and translating the generated pipeline in Enterprise Mashup Markup Language (EMML) (Open Mashup Alliance) that is an XML script language for mashup development.

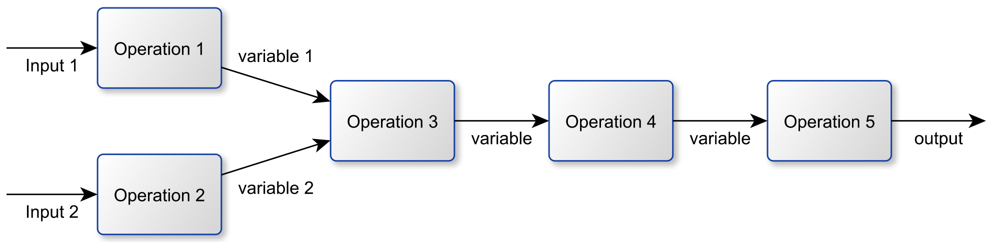


Figure 2.3: Mashup as a pipe paradigm - mashing up data in several operations which are sequentially processed.

Another fundamental distinction, which distinguishes traditional integration applications and mashups, is mainly the fact that mashups are designed for specific situations only (Yu et al., 2008). In general, a mashup will be created every time when some feature is missing or someone identified an improvement or useful combination of existing applications, data or presentation of content. According to Ogrinz (2009) mashups are designed for a "short life, respectively to its purpose, but are also implicitly reusable, which leads to further recombinations and development". That could only be accomplished by using the simple programming principle of mashups. For instance, compared with other approaches, such as SOA, that approach means a lot less effort for the experienced programming staff of a company. The reason for that is because the staff does not have to provide the services, APIs, maintained databases or compensable modules that are needed. Additionally, for using an API or compensable services the applied person has to have some knowledge about programming and how to use the services. The mashup architecture allows a much simpler access to all sorts of data, services or information so that everyone can build an application that is completely personalized to his/her needs in a specific situation (cf. chapter 3).

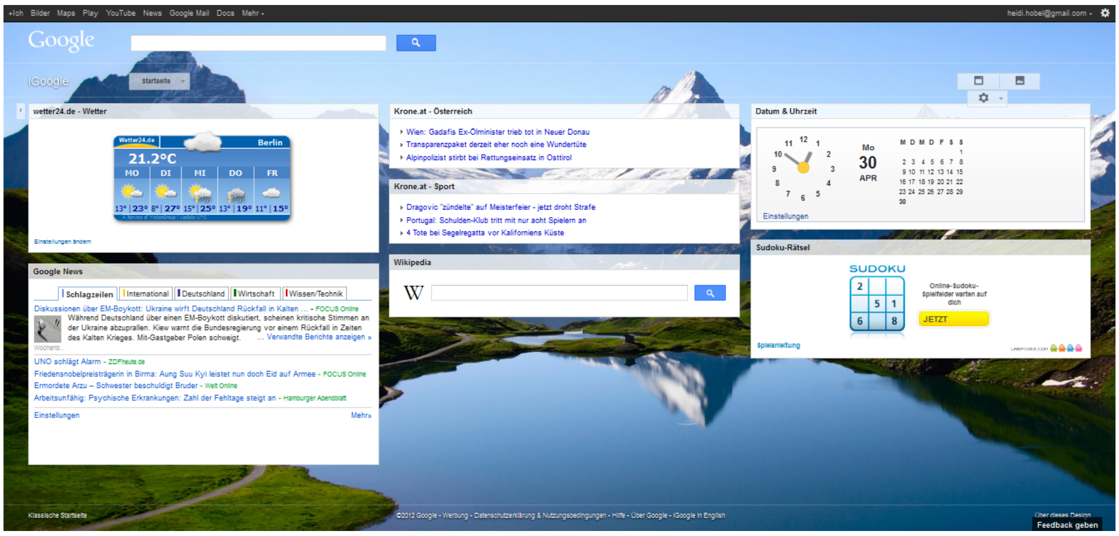
2.4 Mashup types

According to Ogrinz (2009), mashups could be roughly categorized in consumer and enterprise mashups. Consumer mashups are generally built by normal users that use a for-free accessible visual editor like Yahoo! Pipes (Yahoo!). Enterprise mashups are more complex and require deeper programming knowledge than consumer mashups. For instance, if a specific calculation or algorithm is not preprogrammed, then the creator has to build it her-/himself. Another important difference is that enterprise mashups have to take several security aspects into account (cf. chapter 3.3). For enterprise mashups some sophisticated frameworks have been developed, such as IBM Mashup Center (IBM) or JackBe Presto (JackBe).

Further categorizations (Bader, 2012) divide mashups into presentation, data and process mashups, which are explained from his point of view in the following chapters. Basically, these categories distinguish a mashup based on the intended purpose.

Presentation mashups

Presentation mashups are aimed to present something in a personalized or unique way. The whole approach is based on widgets that are predefined visualisation components. These widgets can be combined to a new and personalized view just like the Web sites "iGoogle"[1] (cf. figure 2.4).



Diskussionen über EM-Boykott: Ukraine wirft Deutschland Rückfall in Kalten ... - FOCUS Online
Während Deutschland über einen EM-Boykott diskutiert, scheinen kritische Stimmen an der Ukraine abzuprallen. Kiew warnt die Bundesregierung vor einem Rückfall in Zeiten des Kalten Krieges. Mit-Gastgeber Polen schweigt. ... Verwandte Berichte anzeigen »
UNO schlägt Alarm - ZDFheute.de
Friedensnobelpreisträgerin in Birma: Aung Suu Kyi leistet nun doch Eid auf Armee - FOCUS Online
Ermordete Arzu – Schwester beschuldigt Bruder - Welt Online
Arbeitsunfähig: Psychische Erkrankungen: Zahl der Fehltage steigt an - Hamburger Abendblatt

Wien: Gadafis Ex-Ölminister trieb tot in Neuer Donau
Transparenzpaket derzeit eher noch eine Wundertüte
Alpinpolizist stirbt bei Rettungseinsatz in Osttirol

Dragovic "zündelte" auf Meisterfeier - jetzt droht Strafe
Portugal: Schulden-Klub tritt mit nur acht Spielern an
4 Tote bei Segelregatta vor Kaliforniens Küste

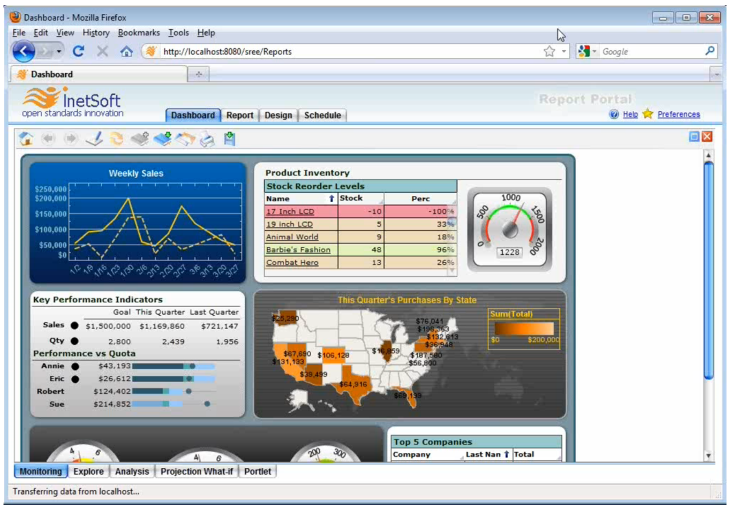
©2012 Google - Werbung - Datenschutzerklärung & Nutzungsbedingungen - Hilfe - Über Google - iGoogle in English

Figure 2.4: iGoogle: Google's user interface, which could be designed by the user by adding and removing widgets to the interface.

[1] `http://www.google.de/ig`

Data mashups

The purpose of data mashups is basically to collect/gather data from one or more sources and process them according to the needs of the user. When talking about data mashups it is helpful to apply terms from a database to describe the structure of the collected data and the generated output of the mashup. Thus, we are talking about tables that are structured in rows (tupel) and columns (attribute). When collecting data from different sources, that data is mostly joined, at the attribute axis, or merged, at the tupel axis. Then basic operations are performed to filter, select, append, change, replace, etc. attributes, tupels or single values. An example of use is that somebody wants to calculate a statistic from data that is spread over the Internet. That data has to be collected from all the sources and then evaluated according to the purpose of the statistic. Instead that the person collects the data itself, cleaning up the data and finally calculating the statistics that could be automated by a simple mashup. For example InetSoft (InetSoft) provides a platform that is based on data mashups and ultimately presents the calculations in a personal dashboard (cf. 2.5).



Stock Reorder Levels

Name	Stock	Perc
17 Inch LCD	-10	-100%
19 inch LCD	5	33%
Animal World	9	18%
Barbie's Fashion	48	96%
Combat Hero	13	26%

Key Performance Indicators

	Goal	This Quarter	Last Quarter
Sales ●	$1,500,000	$1,169,860	$721,147
Qty ●	2,800	2,439	1,956

Performance vs Quota

Annie ●	$43,193
Eric ●	$26,612
Robert	$124,402
Sue	$214,852

Top 5 Companies

Company	Last Nan	Total

Figure 2.5: InetSoft: Practical example of a data mashup - automated statistical evaluation of a huge data amount. (InetSoft)

Process mashups

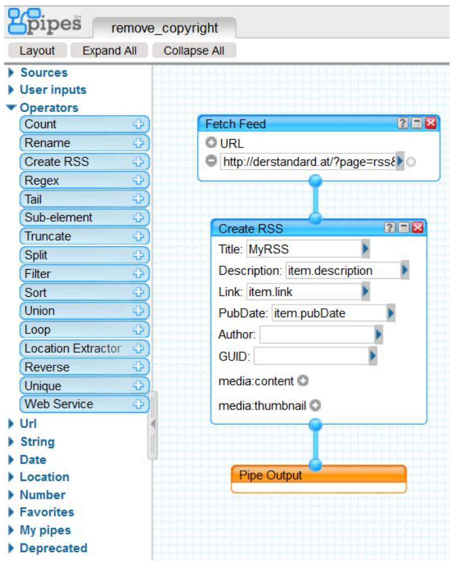
Process mashups are actually meant to gather information and process the data to fulfil a certain task. These mashups fall into the most complex category of mashups and require adequate programming knowledge according to the purpose. Process mashups are also often called enterprise mashups, due to the fact that this concept is widely used in an enterprise environment. Although a variety of mashup development tools are available, process mashups almost always depend on manual programming and the realization or implementation of a work flow respectively. (Bader, 2012)

2.5 Mashup frameworks and editors

Platforms, frameworks, editors and other development tools simplify the development of mashups in such a way that even inexperienced end users could mash up their own applications. The range of available products reaches from free Web Applications where an end user can design his/her own mashup by dragging and dropping predefined modules and connecting them to create a control flow. The next sub-chapters deal with numerous tools and emphasize the variety of possibilities that are offered from the editors.

Yahoo Pipes

Yahoo Pipes (Yahoo!) is a free Web application and uses the concept of a pipeline to design and build a mashup. The Web application provides a user interface that allows the user to drag and drop pre-defined pipeline elements with a visual editor and connect them to a data processing pipeline. The generated pipe is basically designed to coalesce a new Web Site with different data sources like Feeds, Websites, Web Services, Databases, JSON and XML data.



URL: http://derstandard.at/?page=rss&

Title: MyRSS
Description: item.description
Link: item.link
PubDate: item.pubDate
Author:
GUID:

Figure 2.6: Yahoo Pipes Editor: A Web application to design a new data processing pipeline to create a new personalized data resource.

The pipeline elements allow the user to use features like filtering, term or location extraction.

Intel Mash Maker

Intel Mash Maker (Intel) is a Web application in combination with a browser plug-in, which lets users annotate Web pages while browsing. In the Web application, the annotated Web pages could be integrated subsequently in a personalized new Web site. For the sake of efficiency the mashed up Web pages could be stored as mashup or widget respectively so that the actual widget could be shared and reused by other users.

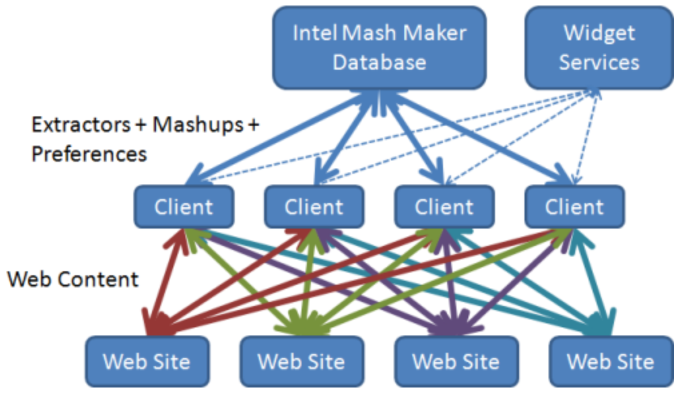


Figure 2.7: Intel Mash Maker: Basic functional principle of Intel Mash Maker. (Intel)

Figure 2.7 illustrates the functional principle of Intel Mash Maker. A client gathers the web content from different sites and the extracted content is stored in the database together with the user information. Additionally, the preferences and self-created mashups are stored. The preferences are used by the platform to suggest integrable, and, to the actual content suitable, mashups and widgets. In the design phase of the new site, the stored content as well as the mashups and widgets that are freely accessible can be integrated. The whole functionality exhibits that Mash Maker is specialized in content presentation or presentation mashup editors only, while data processing and security requirements are not considered. The software is not commercial and completely freely available.

IBM Mashup Center

IBM Mashup Center (IBM) is a commercial Mashup framework that is designed to mash-up different data sources from an enterprise, convert them in a Feed format and combine/process the whole data according to the personal needs of the end user. The framework is able to access data from the common Back-End-Sources such as databases, SAP, different applications and internet sources like Web Services, Web pages or RSS Feeds. The personalized Feeds serve as new data sources which could be converted into XML, ATOM or RSS. When combined with other IBM products, Mashup Center is also able to provide a platform for widget creating that can be used in the Mashup Center to build Web applications.

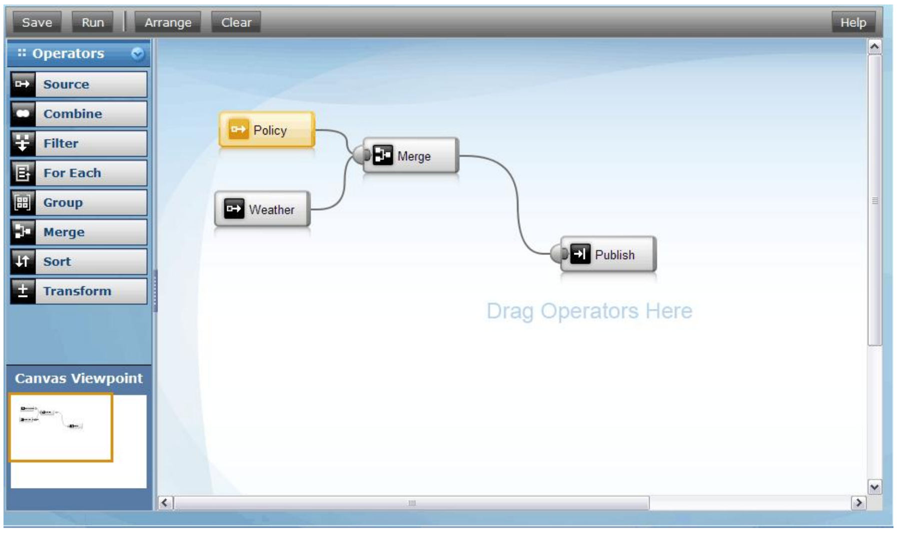


Figure 2.8: IBM Mashup Center: A editor for designing mashup pipelines. (IBM)

An important enhancement compared with non commercial frameworks is that IBM Mashup Center implemented a security mechanism for the data sources. Additionally, it includes its own user management so that the whole system as well as the single data sources could be provided with access rights. The framework has to be installed at the enterprise's server landscape.

JackBe Presto

Presto (JackBe) is one of the most sophisticated enterprise mashup solutions at the moment. It provides a wide selection of predefined possibilities to integrate different data sources. If a data source is not supported, it could be easily embedded by developing a new connector.

The whole mashup development process in Presto is based on three artefacts that could be created and finally connected. *Services* are basically data sources that have well defined connector interfaces so that it is accessible from a mashup. *Mashups* are the logical level that gathers the data with usage of services and transforms the data according to the defined rules. *Mashlets* are basically widgets and provide the logic to present the results of the created mashup. For instance, figure 2.9 illustrates the result of a mashup by presenting four mashlets in the browser.

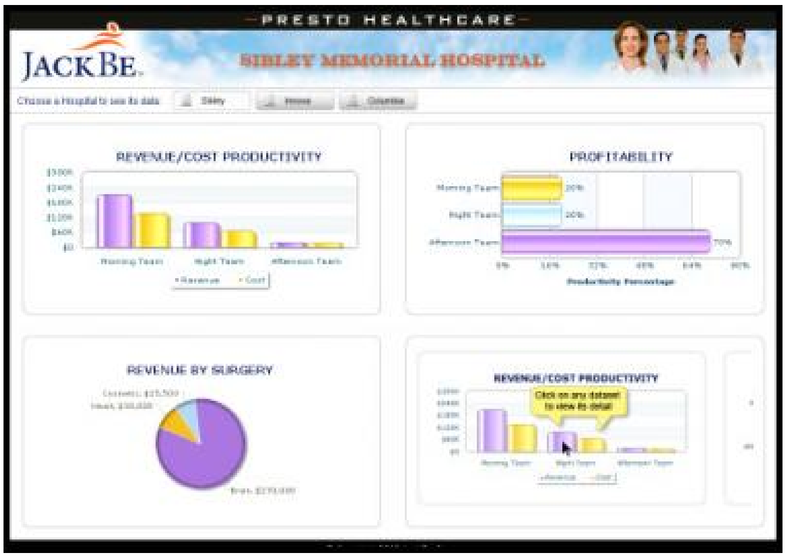


Figure 2.9: JackBe Presto: Result of a mashup that had been designed with Presto. The figure shows the computed results with four mashlets. (JackBe)

In general, Presto consists of three components. A *platform* where the mashups are executed. A visual editor and a developer environment for more complex tasks that need personal programming in EMML. And a software development kit (SDK) for the integration of clients. Due to the server-side execution of the mashups it is possible to provide access rules for mashups as well as data sources.

Provided security mechanisms of mashup vendors

Hoyer and Fischer (2008) provide a market overview of different mashup tools. According to them, mashup tools are in the early stages of development, especially concerning typical enterprise requirements such as security and reliability. By looking at Yahoo Pipes and Intel Mash Maker, which are not specifically designed for enterprise application, we do not find any security or privacy mechanisms. Although non-commercial tools provide some predefined security solutions, there are still open requirements (Hoyer and Fischer, 2008).

CHAPTER 3

Mashups in organizational context

3.1 Enterprise Mashups

Enterprise mashups are in general mashups solutions that are used in organisational context. The mashup solutions are designed from single staff members whenever an automated process or application is missing for simplifying their work. The operations of enterprise mashups depend on the purpose of the solution, thus data processing applications will just like normal mashups execute steps such as data gathering, looping and filtering. However, according to Hoyer some of the building blocks of the enterprise mashup's control flow are much more complex and sophisticated and need the support of skilled development staff. Additionally, enterprise mashups focus more than normal mashup solutions on User Interface support, integration of other existing concepts like Service-Oriented Architecture (SOA) or compensable services and also as much end-user support as possible. For enterprise mashup solutions thus the following definition will be used:

> "An enterprise mashup is a Web-based resource that combines existing resources, be it content, data or application functionality, from more than one resource by empowering the end users to create and adapt individual information centric and situational applications" (Hoyer and Stanoevska-Slabeva, 2009a)

In order to fulfil the requirements of user support, usability and technology integration, there is a trend of using a platform for enterprise mashup support and development in form of stack. The from Hoyer presented stack basically has three layers and is designed to build a collaborative platform that lets different user roles work together. The structure of such an enterprise mashup stack is shown in figure 3.1 and consists of the *resource, gadget and mashup* layer. The basic user roles for the platform based mashing-up are *end-users, key-users, consultants and developers*. *Resources* are basic operations, data access modules and sophisticated application logic that is created and maintained by the developing staff. *Gadgets* are just a synonym for widgets that are created by consultants and key Users that have basic programming knowledge. These gadgets provide a graphical user interface for the data that is processed by the resource (building blocks) based workflow. Key Users design the actual *mashups* by wiring the predefined gadgets that can finally be executed by the end users.

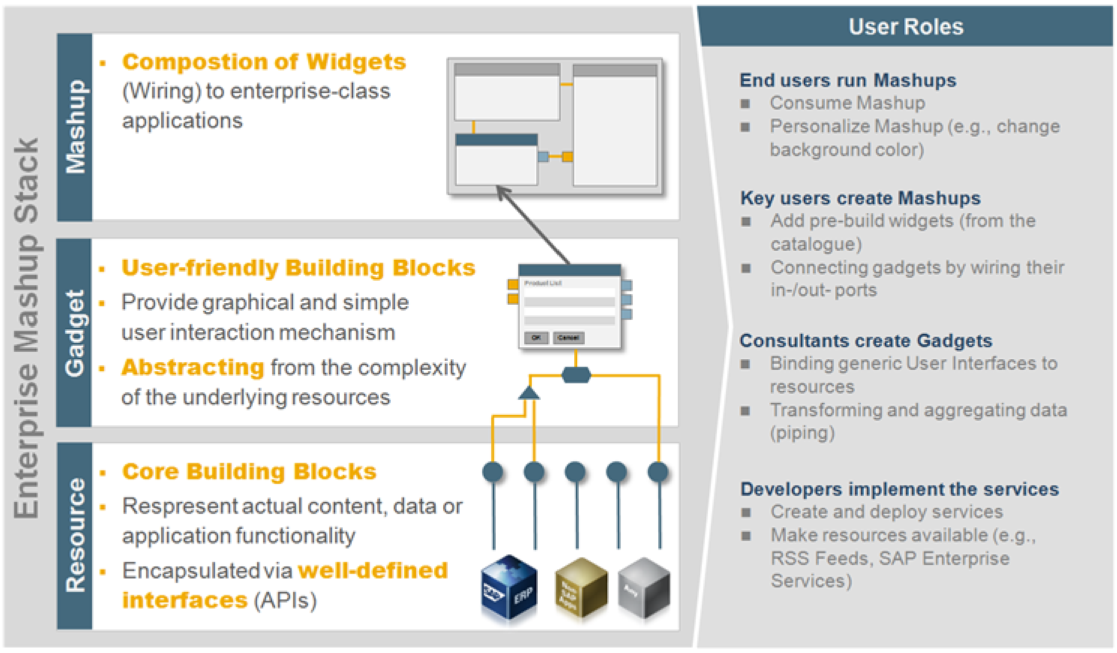

Enterprise Mashup Stack
Mashup
Compostion of Widgets
(Wiring) to enterprise-class applications
Gadget
User-friendly Building Blocks
Provide graphical and simple user interaction mechanism
Abstracting from the complexity of the underlying resources
Resource
Core Building Blocks
Respresent actual content, data or application functionality
Encapsulated via well-defined interfaces (APIs)
ERP
User Roles
End users run Mashups
Consume Mashup
Personalize Mashup (e.g., change background color)
Key users create Mashups
Add pre-build widgets (from the catalogue)
Connecting gadgets by wiring their in-/out- ports
Consultants create Gadgets
Binding generic User Interfaces to resources
Transforming and aggregating data (piping)
Developers implement the services
Create and deploy services
Make resources available (e.g., RSS Feeds, SAP Enterprise Services)

Figure 3.1: The Enterprise Mashup Stack: Design principle for enterprise mashup environments. (Hoyer)

According to Hoyer and Stanoevska-Slabeva (2009b) the usage of comprehensive solutions from mashup vendors like IBM Mashup Center or Kapow Mashup Server is rising. Actually, a trend emerges of enterprise mashup environments that are building the technical fundamentals for the proposed concept and principle. For personalization and adjusting the environment to the enterprises needs, the proposed mashup stack can be adapted. However, challenges for such implemented systems are to cope with the adoption of complex, new or old technology and adjust the system to every new challenge that came along with time due to the novelty of the whole concept. But as typical Web 2.0 technology (cf. chapter 2.1) the behaviour of the users is changing too and has great influence on the enterprises and mashup environment. People are sharing created mashups with others and provide feedback and improvements for these solutions. Mashup catalogues are created that comprise the amount of created mashups so that for each problem statement one or more existing solutions can be found trough ratings and recommendations. The whole systems leads users with different knowledge, skills and roles to participate and build their own community. However, in organisational context these users are also staff members that have to comply with rules and policies that are established by the enterprise. Thus, the fact alone that these users hold different roles that lead to different access rights of resources, gadgets and mashups causes an enterprise to consider possible countermeasures against unauthorised data or mashup access as well as sharing of created mashups. As mentioned in chapter 2.5 current mashup vendors' reactions are to restrict the access rights of the users. But access right restrictions are not sufficient to enforce all of established business policies for mashup solutions. The next chapters will discuss the impact of mashups in enterprises and the security and privacy concerns of mashup solutions in more detail.

3.2 Impact of Mashups on organization

In principle the usage of Enterprise 2.0 technologies such as Wikis and Blogs in enterprises is in the ascendant (cf. chapter 2.2). Mashup solutions for enterprises are a relatively new trend, however they have great potential in a number of application areas of an enterprise. The next sub-chapter discusses some of these impacts and drawbacks in organisational context.

Impact on limited Resources

The value of mashup solutions in enterprises lies mostly in their simplicity. Almost everyone can design a control flow by drag and drop of predefined modules. Thinking of limited enterprise resources like the time of software developers or the IT support staff, mashup solutions can diminish consumption of these resources. Thus, the employees of an enterprise will not be dependent on software developers. They will build their own applications that automate some of their daily work, therewith also having a positive effect on the time resources of the whole staff. Concerning the argument that most employees do not have enough background knowledge or IT skills, a counterargument is that some employees already do a kind of mashuping (Bader, 2012). Thinking merely of the amount of time that is necessary to generate daily or monthly spreadsheets by gathering data from different sources and composing that data into a well-defined form, simple mashups can automate these processes and reduce the total work time of an employee. The huge amount of tools that can support the mashup development enables a user that he/she just has to model the control flow of the mashup of the actual workflow of the user's task.

Impact on the Long Tail of User Needs

According to Hoyer et al. (2008) the usage of enterprise mashups is the answer of the phenomena, where employees need increasingly individualized applications that exactly meet their daily needs and that can easily be adjusted or reused respectively.

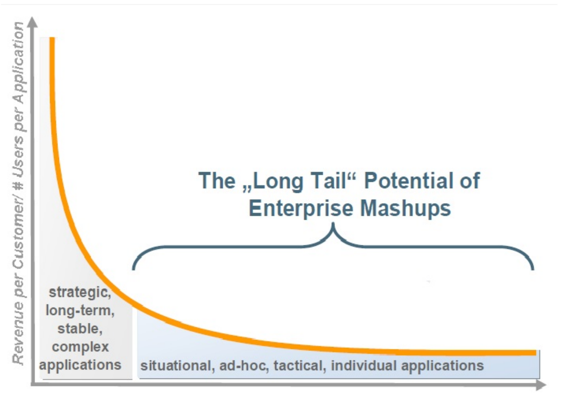


Figure 3.2: The "long tail" potential of enterprise mashups. Mashup solutions have the potential to cover all the software that cannot be developed due to unproportional amount of needed software solutions. (Hoyer et al., 2008)

The phrase "the long tail" addresses a common problem in software engineering in organizational context. The business software development is mostly occupied to satisfy only 20% of the user needs. Thereby the applications development is oriented at the largest projects with the highest revenue per customer (cf. figure 3.2), even if the efforts do not lead to a gain in efficiency. The mashup concept and lightweight composition approach is in these days an attempt to reduce the amount of unmet user needs in software development. (Ogrinz, 2009)

The "Long Tail" covers all the situational, ad-hoc, tactical or individual applications Hoyer et al. (2008) that will not be build due to the lack of resources in software development (cf. chapter 3.2). In the following these kinds of software will be explained as well as the possible influence of enterprise mashups:

- *Situational* means all the software that is build for one specific situation of task respectively. For instance, a software that composes one specific chart sheet for only one project that runs only a specific period of time. When the project is finished the software will be useless but during the runtime of the project the software could entail great value. Common software is mostly build for more than one specific situation or application area. For example, a software that is build for project management can be used for other projects too. In these cases the project team has to configure the software according to their needs. Building such comprehensive software means long-term development cycles and a lot of effort. For only one project that effort is not manageable. On the other side, the programming staff will not build a software for this short usage timeframe, especially due to their limited time resources. Mashups build on this idea and can be used for such software due to their simplicity and short development time. Moreover, mashups are mostly designed for one specific task due to the high grade of individualization that is explained in the following. For more general solutions, the development cycle of the application would be too time expensive and the software has to be maintained afterwards, patched as well as fixed in the case of a software bug. It is also important to note that some useful situational solutions have the potential to be shared and reused by other users.

- *Ad-hoc* refers to software with a short development time that can be build immediately when a situation occurs that requires a specific, new or adapted software solution. Normally, building software requires a series of steps like project planning, requirement analysis, implementation, testing and some of them have to be repeated. For instance, if some failures in the test phase are discovered, the phase of the project will switch back to implementation. A more detailed discussion about the complexity of software projects is provided by Burke (1999) for example. Thus, reacting on unexpected situations where a software is useful is difficult to comprehend. Mashup solutions, on the contrary, can be easily build in a limited time frame. Especially if a solid fundamental structure already exists so that the modules are pre-implemented and they are only stringed together to an appropriate control flow.

- *Tactical* applications are often neglected by the developing staff. That is due to the nature of the tactical applications which are aimed to support the decision process from short time decisions in contrast to strategic applications that support long time decision processes. Such Decision Support Systems (DSS) work by evaluating a great amount of data and are normally very time consuming in their processing phase (Turban et al., 2004). The data from the evaluation phase are used for decisions in the distant future or strategical focus in the next few years. Tactical applications are rather build for daily decisions and have a mutable character in data sources, control flow and user area. Thus, building tactical is mostly a difficult task. Similar to situational and ad-hoc implementations mashup solutions have great potential in this area of application. Moreover, the character of data mashups is to be geared to gathering and processing data from different sources that is fundamental for DSSs.

- *Individual* and exactly appropriate software for everyone can hardly be accomplished, mostly due to two factors. At first, there is often a communication problem between software developers and people from other branches and thus they do not always understand exactly what the other side means and how the software should work and look like precisely. Second, building for everyone exactly appropriate software involves a great deal of effort for the developing staff. Mashup solutions with their mutual character and their different types and applications areas (cf. chapter 2) can provide a solid base for this problem.

Impact on the Enterprise's Agility & Flexibility

According to Pahlke et al. (2010), Bader (2012) and Hoyer et al. (2008) an appropriate platform for enterprise mashups could have a value increasing influence on all kinds of companies that use IT-solutions for value generation or tactical, statical and administrative applications. Everywhere where characteristics like mutability and application area diversity are needed, mashups have the potential to transfer those properties to the using companies. As a result, the enterprise will be more flexible in their internal workflows due to the fact that the staff are not dependent on their given and statical software. Instead, if some changes in the organization are necessary, software can be easily adapted and changed along with the changes in the company. An important factor of adaptation is the necessary time that is needed to accomplish the change. That has influence on the agility of the company to react on change requests. Change requests are mostly influenced by the environment of that respective company, like the customers, business competition or parter companies. In summary, mashup solutions most likely have a positive effect on agility and flexibility to react on changes like market demands and thus also a positive effect on the competitiveness of a company.

Impact on the Enterprise's Knowledge Management

Concerning the amount of data of an enterprise, it has to be considered that the data is from different times, sources and distributed over the internal hierarchy of the company, all the partner enterprises as well as the comprehensive internet. According to the size of the company the amount of data could be overwhelming and new data and data sources are added each day. As much of that data as possible have to be cleaned up, transferred into a union structure and documented so that the data could be used by DSSs or other enterprise applications. Mashup solutions transfer these steps in the control flow of the applications so that the whole clean-up process is not necessarily required. That favours the amount of possible sources that could be used by enterprise applications and the time to evaluate that data from the first access possibilities to the actual usage of that data.

Drawbacks of Enterprise Mashups

With all the potential, possibilities and advantages of enterprise mashups, the question of possible drawbacks and risks which have come along with Enterprise 2.0 technologies also arises. For all we know, not everyone has knowledge about performance, security or other related aspects. According to the purpose of this thesis, some important security issues will be discussed that came along with the enterprise mashup concept.

3.3 Security, trustworthiness and privacy concerns

Security, trustworthiness and privacy are important but hard-to-accomplish tasks in Web 2.0 and Enterprise 2.0. The mashup technology is no exception and along with the purpose, simplicity and architecture of mashups, new risks in many respects have to be considered.

Bader et al. (2010) provides a comprehensive survey about the challenges and risks that come along with the usage of mashups. In the following chapters the proposed security, privacy and trust concerns will be explained.

Resource trustworthiness

Independent from the kind and purpose, mashups mash-up one or more sources like data or services. With regard to enterprise mashups, the resources are provided by the resources layer that are created by the developing staff. The internal data is often cleaned-up and documented. But data could become obsolete or due to other reasons it is not assured that the fundamental data is really trustworthy. External data like information that are gathered from the Internet, e.g. a HTML parser that gathers news from competitors, is much more equivocal due to the fact that everyone can add new content. To summarize, the trustworthiness of data is not assured every time. By manually processing the data, mostly doubtful information could be recognized, due to the ability of humans to understand the data and to infer hidden relations of it, e.g. that in a name table there normally are no numbers. A human is also able to cross-check the gathered information and rate it according to given parameters like accuracy, correctness or exactness. A mashup would face many severe difficulties during that task. That is due to the fact that the data is already processed, filtered, etc. For instance, the to mashup filters only one data row of a table. Now it is impossible to conclude from other rows about the validity of that one row. In a more specific use case that could be the revenue of one year that could be compared with the revenue of the previous year.

Services as resources or operational steps bear also the risk of trustworthiness. These services are basically black boxes that do not disclose anything about their internal procedures. It is only known the input and output of the whole service. Thus, what these services are doing in all detail is just impossible to determine and the user depends on the documentation of the service. Additionally, enterprise mashups are build by normal staff members that do not always understand about the code even if it would be available, or understand complex procedures or algorithms. Mashups also follow the paradigm of a black box, often not even the data resources of it are known. Combined with the fact that mashups can be shared, it is hardly possible to consider the generated output as valid or evaluate the sources of the mashup on their trustworthiness. Additionally, mashups can be used as resources too, thus the effects of an untrustworthy resource could be inherited by the mashup solutions that are based on that mashup.

Content and feed copyright issues

Considering that gathered data is processed by a mashup, people have to consider the copyright of the used information. Especially in enterprise context the usage of copyright protected data could lead to unwanted consequences. Sometimes it is only necessary to reference the sources correctly or take over the copyright information that is included in the text, content or meta information, like the copyright tag from RSS Feeds. However, there is no generally accepted standard how to mark copyright of data. Additionally, with mashups it is simple to fetch data from a RSS Feed and remove the copyright information from the content. An enterprise has to control these operations by establishing rules that forbid such operations or enforce the staff members to add a source information for each data piece. Just as mentioned in resource trustworthiness, the problem of omitted copyright information can be inherited by mashups that are based on services and other mashups that actually do not mark the content with their sources or remove copyright information.

In enterprise mashup environments that could be accomplished by forbidding all building modules that remove copyright and when data are fetched from a module that the source is automatically added. But it is also inevitable that gadgets use basic operations like remove, append, select and filter can sidestep these measures. That is, in reality, more difficult than it sounds. For instance, let us assume that RSS data is processed. RSS data is basically in XML format and the information therein is structured in a tabular form with copyright as an own column. The user has now a variety of possibilities to remove the copyright, e.g. by selecting columns except for the copyright, build a new table and append columns except for the copyright, and replace the copyright column or the single data cells with their own name. Without security mechanisms the enterprise is compelled to trust the developers and key users that they comply with the copyright policies of the company.

Information leakage and creation of sensitive information through aggregation

As mentioned before, data is one of the most valuable assets of an enterprise. Based on the data, enterprises make decisions on their strategical and tactical direction, where they can cut costs, etc. Some of the enterprise's data and services is also used for marketing. Information leakage means that not only data and services that are intended for the general public are released, but private information is leaking beyond organisational boundaries. That information can be used by competition to infer sensitive information like the tactical direction of the enterprise in question. Even if the data has no direct value, with a lot of small pieces of information and Business Intelligence sensitive information can be created. For instance, a competitor can record the amount of the enterprise's new announced products of the last years and then infer knowledge about the strategical direction of the enterprise.

The simplicity of mashups allows almost everyone to build a mashup and together with the nature of the mashup that it could be easily shared, it facilitates the information flow of an enterprise that uses mashup solutions. Which data and services may be shared with the general public is becoming blurry and the data have to be categorized into private and public data. Thus, enterprises are forced to reconsider their data sharing policies and have to think about the enforcement of the established rules by appropriate business rules to protect their private data.

Distribution of data/sensitive data to unknown or unauthorized users

As mentioned in the chapter information leakage and creation of sensitive information through aggregation, the simplicity of mashups allows almost everyone to build a mashup. However, not everyone has granted access rights for all kinds of data or services. As mentioned in chapter 2.5 the security mechanisms of commercial mashup environment vendors are moving towards implemented access control for the fundamental data and services of the mashup, whereas it also has to be considered that the mashup can be shared and executed by other users. Let us assume that there is a sophisticated authentication system implemented for mashup creation and execution, due to the fact the mashups are developed as Web 2.0 technology and thus for non commercial application, the development of security mechanism like a secure and trustworthy transfer of data had been omitted for mashups. In enterprise mashup environments that has to be considered, otherwise by executing of mashups on the clients host or in an insecure network where an attacker can easily sniff the data transfer, valuable data could be exploited. In enterprise environments that is mostly assured through a platform, where the users have to be authenticated for mashup creation and execution as well as the centralized execution of the mashup.

However, it is not always enough to restrict the access for specific user roles. For instance, let us assume that in a mashup based health care system, a research organisation must have access to the patients data so that statistics can be calculated that are used for publications and the improvement of the health care system. The research organisation are now responsible that evaluations according to the weight of the patients are not allowed due to the patients' lacking declaration of consent or that the name of the patients is removed, etc. Another example is an enterprise where the staff group is responsible to create an annual report of the enterprise. For that it needs to calculate specific statistics that could only be calculated from the exact numbers of the year. Thus, the group is only authorised to create and execute mashups that generate specific outcomes from the evaluation of the data, but is not allowed to see the whole data. In summary, it is not sufficient to control what data the mashups are processing but also necessary to control what the user is doing with the data.

The fact that without special background knowledge in programming, data structures, security and privacy, mashups could be easily designed and executed, as well as the fact that mashup solutions are mostly intended for ad-hoc or situational applications, leads to careless applications and data handling. Most end users do not know which data they might access or which data may the person access for whom the mashup is designed. For instance, in enterprises a department may build a mashup solution for other departments. Each department should only see the relevant and most necessary data for accomplishing the task. That could be the monthly aggregated internal costs that each department puts to the one department to expense. It is important that other data should not be shared, like the revenue of the department.

Concluding, not only the prevention of data leakage is an important security mechanism that has to be considered in the usage of enterprise mashups but also the internal distribution of data brings along several risks and has to be secured with an appropriate security model.

CHAPTER 4

Semantic Web Technologies

4.1 The need for semantics

Computers are designed to process data in a short time frame. However, the evolution of the Web and inventions of technologies like mashups, blogs, wikis, etc. lead to huge amounts of knowledge in various media forms. Humans could understand and interpret data by filling the gap of information with their own knowledge and linking them to the original data, whereas a computer cannot without further instructions. (Breslin et al., 2009) For instance, a list of names which has no particular description which of the terms are the first name and surname. A human could interpret that by finding familiar first names and concluding them with the knowledge of the other names. The computer needs further instructions, e.g. that a name normally consists of two terms, sometimes a name includes a second first name, etc. For this reason there is a tendency of information processing toward knowledge representation mechanisms in various disciplines which include, for example, the Semantic Web.

Additionally, all forms of organisations depend heavily on data that are accumulated from the Web, partners and internal data from the last years. Due to increasingly growing storage capacity that also leads to huge amounts of data that is used for the organisations' strategical and tactical decisions, marketing, controlling, etc. In order to interpret this data only with computational facilities, the trend goes into the direction of labelling the data with meta data and providing the general meaning with ontologies (Breslin et al., 2009). Thus, we have to distinguish between the syntax and semantics of data. Syntax denotes the general structure of the data, type, length, etc. Semantics denotes the real *meaning* of the data. For instance, let us assume that a table with several numbers is given. The numbers only denote the type of data as integer, long or something similar. However, the numbers are completely useless due to the fact that it is not possible to infer what the numbers really mean. Let us assume the numbers denote the identification (ID) number of an enterprise's staff, or in other words the meaning or semantics of these numbers. Then the IDs have the additional meaning that each of the enterprise's staff members has a unique ID, so that we can infer which ID belongs to which staff member. In the example, the meta data of the data would be for instance the column label. With the meta data alone a machine would not understand what the word 'Staff ID Number' means and that is why we need an ontology. With ontologies the relations that humans could infer from their own knowledge could be modelled and used by a machine to do reasoning processes such as that the ID belongs to exactly one staff member of the enterprise in question. The ontology concept belongs to the sophisticated Semantic Web Technologies and will be introduced in this chapter.

The Semantic Web Stack

Semantic Web Technologies denotes the fundamental technologies and architecture of the Semantic Web. The *Semantic Web* is an attempt of the W3C that the resources of the Web are enriched with meta data and provided with a holistic framework that allows the annotated data to be processed according to their meaning. (W3C) For instance, this effort can improve the results of search engines because at the moment only key words are used to identify pages that fit to the users search. But also for the usage in enterprises the concepts of Semantic Web are used to improve their ability to share data with different structure with other enterprises, between departments or use it for intelligent systems like improved DSSs. Figure 4.1 shows a simplified and schematic overview about the inevitable technologies that build the basis of the Semantic Web and find their usage in several new applications, for example for new intelligent systems that are adapted for organisations, enterprises or communities.

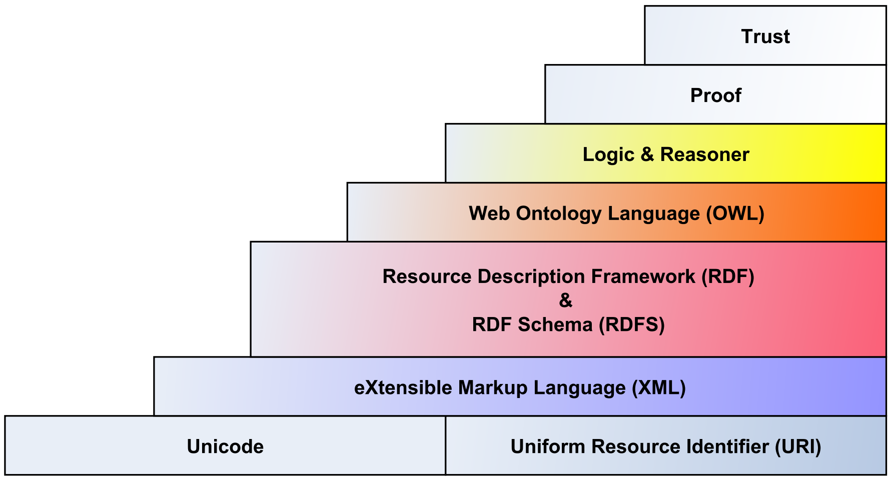

Trust
Proof
Logic & Reasoner
Web Ontology Language (OWL)
Resource Description Framework (RDF)
&
RDF Schema (RDFS)
eXtensible Markup Language (XML)
Unicode
Uniform Resource Identifier (URI)

Figure 4.1: A simplified illustration of the architecture of the Semantic Web. (Largely borrowed from (Obitko, 2007))

In the following the single layers of the Semantic Web Stack will be explained shortly. For reference and further reading the reader is referred to the W3C documentation.

Unified Resource Identifier (URI): URIs are intended to provide a comprehensive name system that allows data exchange and identification of the exact instances. For example, enterprise A has an employee with the name Mark Twain. If enterprise B has also a member with the same name and enterprise A takes over enterprise B, then enterprise A could not longer distinguish these two persons. All the same with the identification number of the enterprises, suddenly there are two people with the same ID. That is why URIs adds to the name a context or location part respectively. For instance, the name of the type of information is 'mark_twain' with the context 'http://wwww.enterprise-a.at/' so that the full URI would be 'http://wwww.enterprise-a.at/mark_twain'.

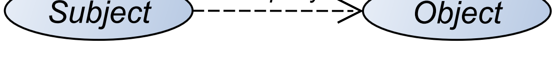
Unicode: The character set of the Semantic Web is standardised from W3C in Unicode so that resources are consistently encoded.

eXtensible Markup Language (XML): XML is standardised as general syntax of the Semantic Web. Thus, XML structures the data in the Web in the form of HTML and serves as syntax for RDF, RDFS and OWL. The fact that XML is compatible with Namespaces (cf. chapter 4.3) and Schemas enables well-formed and easily readable specifications, even for people that are not used to long names that came along with URIs and confusing/unstructured documents.

Resource Description Format (RDF): RDF serves as the core representational format of semantics, structuring the meaning in form of a graph. The nodes and directed edges (arrows) of the graph can be translated as a triple with subject, predicate and object:

$$< \textbf{Subject Predicate Object} >$$

Subject = Resource *(Node)*, Predicate = Property *(Arrow)*, Object = Resource *(Node)*



Each of these triples is called a statement, determining something that is true and predefined knowledge that can be used by a machine for inferring. For instance, we state that an *enterprise A has as CEO Mark Twain.* Then the subject would be enterprise_A, the predicate has_as_CEO and the object Mark_Twain. RDF is meant to specify such relations, saving it at a central place or carrying it as meta data in the text.

Resource Description Format Schema (RDFS): RDFS extends RDF by providing the fundamental elements for building taxonomies or type systems respectively. Taxonomies are basically classifications that order elements into classes. A type system is meant as a set of classes, where each class comprises a defined subset of the resources or other classes and thus the classes provide a type for the assigned classes. For instance, let us assume that we have the instances 'Tim Bauer', 'Mark Maurer' and 'Tom Schmidt'. Now, with RDF Schema we can define categories as well as a hierarchy of these categories. Figure 4.2 shows a possible class relationship of the above mentioned example. Thus, the defined instances are of the type humans as well as beings.

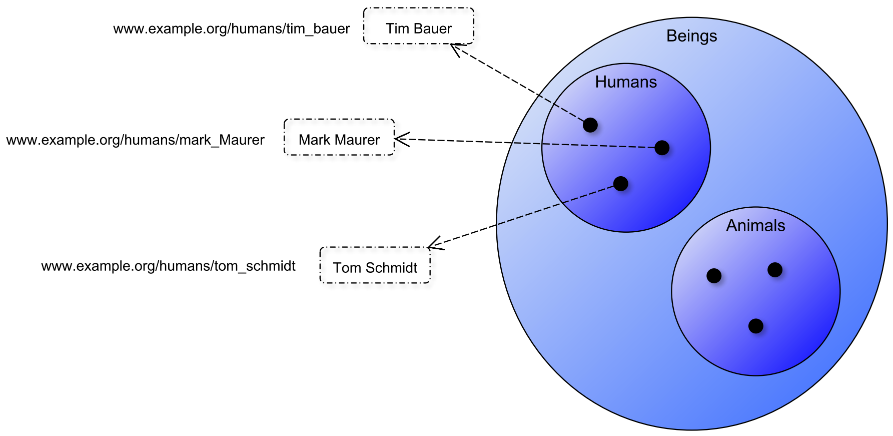

www.example.org/humans/tim_bauer
Tim Bauer
www.example.org/humans/mark_Maurer
Mark Maurer
www.example.org/humans/tom_schmidt
Tom Schmidt
Beings
Humans
Animals

Figure 4.2: A graphical description of a type system in the terms of an ontology and set theory. The instances shown are assigned to classes whereas a class could also be assigned to another class. The concept of URI for identification of resources is inherited from the general RDF Syntax.

RDFS also allows to classify properties or relations respectively and provides some other possibilities to provide meaning for resources. The RDFS taxonomies are stored at a central place and together with RDF that forms a lightweight ontology.

Web Ontology Language (OWL), Reasoner, Proof & Trust: OWL provides more functionality for ontology building and is derived from description logic. It is directly embedded in the syntax of RDFS and enhancing the whole concept to a full ontology by extending the taxonomy with rules. For providing the right solution for each ontology problem such as performance, expressiveness, etc., three kinds of OWL exists. *OWL Lite* provides the language for simple taxonomies and constraint; *OWL DL* is computational complete and decidable but although the language supports a high level of expressiveness there are some constraints; whereas *OWL Full* has no constraints on expressiveness but the language is not sound and complete. Reasoner can use the defined semantics for inferring and deduction. However, the layer proof and trust are merely omitted from the stack, but the whole concept is geared to enable computational proofs that lead to Trust in Web 2.0 and Enterprise 2.0 (cf. chapters 2.1 and 2.2).

Meta data & Context-awareness

Although the common definition of metadata seems meaningless at first, the term *"data about data"* describes the concept precisely. Metadata refers to descriptive structured data about different data itself, in different data formats for the purpose of describing other data, structure the basic data and provide meaning. (Breslin et al., 2009)

Meta data is a way to bring data into a context. For instance, random numbers without a specific assignment to a purpose are completely useless. Meta data is then added to the existing information in different forms. For example, the description of a table, an invisible but processable annotation in a HTML site or a document is centrally stored and indicates how to read received data. The purpose of meta data could be practically anything, important is only that the data gets meaning. For instance, Stock et al. (2010) discusses use cases for meta data in enterprise application. The authors classified seven distinct categories of meta data: definitional meta data, data quality, navigational meta data, process meta data, audit metadata, usage metadata, annotations (semi-structured comments). Meta data is also important for an increasing trend of context-aware applications. In Web applications the trend runs towards recognizing the user's location, identitying and timing them is becoming more and more popular with interactive applications in the area of ubiquitous computing (Abowd et al., 1999). For instance, the search machine Google recognises from your IP address your location and eventually, if your are logged in your Google account, it can also recognise the user's identity. All that is included in the search process and has an impact on the retrieved results. However, context information could be much more complex and have a greater impact on the processing. For instance, in Artificial Intelligence agents perceive with their sensors their environment (Russell and Norvig, 2003) and infer from that their actual context. That context then has together with defined rules or behavioural models effects on the agent's next actions. As mentioned in chapter 4.1 for a lot of data and automated processing, there is a great need for semantics and comprehensive specifications that are understandable for humans as well as machines. The Semantic Web Stack (cf. chapter 4.1) provides sophisticated technologies for the specification of the *agent's whole system*, *rules* and *ramifications of the agent's actions*. Basically, that is the application area of ontologies, building a system for proof and trust where machines can accomplish high-level tasks. But not only in Artificial Intelligence such models are necessary these days. All kinds of systems use the power of Semantic Web Technologies and build context-aware services or applications, where the *system's whole environment and internal behaviour rules* is described in an ontology so that these systems can include that in their decision-making procedures. For instance, for the authentication on a server or something similar, only a simple password check is not longer good enough for security-aware services. Instead, the trustworthiness of each login attempt will be checked according the defined conditions (or context information), such as the amount of logins, the time between each login attempt, etc. Thus, one of a variety of starting points for context-awareness and Semantic Web Technologies is being used.

4.2 Ontologies

The term ontology is derived from the Greek word 'onto' that means being and 'logia' that means discourse. One of the most popular definitions for an ontology is:

"An ontology is an explicit specification of a conceptualization." (Gruber, 1995)

Whereas a conceptualization is a formal representation of *"objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them."* (Genesereth & Nilsson, 1987) Although the definition is in fact very accurate, the definition is very complex and not easily understandable. Thus, the definition of Horrocks and Sattler provides a better explanation about the whole concept:

> "An ontology is an engineering artefact constituted by a specific vocabulary used to describe a certain reality. A set of explicit assumptions regarding the intended meaning of the vocabulary. Thus an ontology describes a formal specification of certain domain: shared understanding of a domain and an interest; formal and machine manipulation of domain of interest." (Bloehdorn et al., 2009)

In order to describe a general and shared understanding of a domain, an ontology works basically with classes, instances and relations (cf. RDF & RDFS). Those concepts are taken over from a taxonomy and according to the requirements are extended with sophisticated functions (cf. OWL) that can compute for further meaning. For instance, figure 4.3 shows a very simple ontology. Chapter 4.3 describes the concepts in more detail as well as how to work with ontologies.

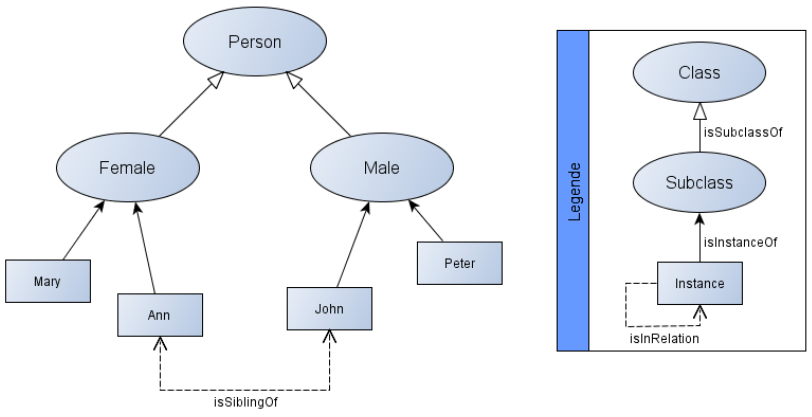


Figure 4.3: A simple example of an ontology with some classes, instances and relations. The general class is named person, which is divided into two disjunct subclasses, although that restriction is not visible in the graph. The concrete instances Ann and John are connected with the relationship 'isSiblingOf' that is in this case symmetric.

4.3 Ontology Languages

In the next sections, the ontology languages RDF, RDFS and OWL will be introduced and explained how to work with these concepts. We will restrict ourself to XML syntax that is beside N-Triples[1], Notation 3[2] and Turtle[3] a serialization syntax for describing a ontology graph.

Resource Description Framework (RDF)

The Resource Description Framework (RDF) is part of the formal languages, which are able to describe structured information. For this reason, RDF references entities to unique identifiers (Uniform Resource Identifiers (URIs)) and defines relationships between these entities (cf. chapter 4.1). For reference and further reading the reader is referred to the W3C (2004c) RDF Primer.

In the most general sense, RDF defines triples, each of them consisting of a subject, a predicate and an object. Listing 4.1 shows a simple example where an object and a subject is described, and linked with a predicate. The whole statement that is defined through the triple, can be translated to a sentence. For instance, 'Mashup processes information' with mashup as subject, process as predicate and information as object.

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ns="http://wwww.rdf-example.org/">
   <ns:Information rdf:about="http://www.rdf-example.org/example1#information"/>
   <ns:Mashup rdf:about="http://www.rdf-example.org/example1#subject">
      <ns:process rdf:resources="http://www.rdf-example.org/example1#process"/>
   </ns:Mashup>
</rdf:RDF>
```

Listing 4.1: Simple RDF/XML example.

The identification of objects RDF is realized through two general concepts: *URIs* and *literals*. The usage of URIs is quite understandable, due to the fact that names could be assigned to different objects, just as a lot of different people are called John Smith. Literals are constant values, represented by character strings, which are used to represent certain kinds of property values such as a date. The root node or header is important to declare the *namespace* of an ontology project. Without namespaces it would be necessary to write the full URI for each entity. Due to the hierarchical structure of URIs we can set the first part fixed and define an abbreviation (ns in the listing). The rdf tag denotes the namespace for rdf commands that is used from compiler.

[1]http://www.w3.org/2001/sw/RDFCore/ntriples/
[2]http://www.w3.org/DesignIssues/Notation3.html
[3]http://www.w3.org/TR/turtle/

Resource Description Framework Schema (RDFS)

RDF Schema (RDFS) is the language that extends the defined statements in RDF Syntax with additional meaning. The general goal is to express simple taxonomies (cf. chapter 4.1). The next paragraphs will introduce some concepts of RDFS that are directly embedded in the RDF syntax. For reference and further reading the reader is referred to the W3C (2004b) RDF Schema documentation and Hitzler et al. (2009).

Classes, subclasses and instances The RDF(S) predicate *rdf:type* is the basis of all class type assignments in this language. In set theory we would state for example $Attribute_1 \equiv Attribute$. With rdf:type we can define that a resource is an instance of the declared class. However, only with RDF it will not be clear if the resource Attribute refers to a single object or a class. Thus, for clarification we use the possibilities of RDFS and declare that Attribute belongs to the class of all classes by explicitly typing the it as classes:

```
<ns:Attribute rdf:about="http://www.rdfs-example.org/example1#attribute">
  <rdf:type rdfs:Class/>
</ns:Attribute>
```

With this statement, a search of Attribute would now return all the declared instances. An important fact is that every resource can belong to more than one class. Thus, the type relationship is not exclusively defined.

The subclass relationship provides further possibilities of typing. Thus, RDF(S) statement *rdfs:subClassOf* enables us to define that one class comprises another class. In set theory we would state that for instance $Attribute \subseteq Column$, or that the class Column comprises the class Attribute. With this premises we can now define

```
<ns:Attribute rdf:about="http://www.rdfs-example.org/example1#attribute">
  <rdfs:subClassOf rdf:resource=="http://www.rdfs-example.org/example1#column"/>
</ns:Attribute>
```

Subclass relationships are *transitive* and *reflexive* as defined. Transitive means that subclasses of subclasses are also subclasses or more formal according to our example $Attribute \subseteq Columns \wedge AttributeSubset \subseteq Attribute \implies AttributeSubset \subseteq Columns$. Reflexive means that every class is its own subclass or $Attribute \subseteq Attribute$. RDF(S) deduces these relationships automatically without further intervention of the user.

Properties, sub-properties and restrictions The purpose of *properties* or *predicates* is to describe relations between two resources, object and subject. For instance, we can define a relationship ’generateAttribute’

```
<ns:generateAttribute
  rdf: about="http://www.rdfs-example.org/example1#generateAttribute">
   <rdf:type rdf:Property/>
</ns:generateAttribute>
```

We can now use that relation to define one or more relations, thus we can use it for each sibling relationship. Therewith, we can say that the property isSiblingOf is the set of all siblings because rdf:Property is in general a class and not a property. Just like the subclass relationship we can also define subPropertyOf relationships that denotes that the one set of properties is a subset of another property set.

RDF(S) provides predefined properties to set restrictions with the goal of drawing further conclusions or enabling the calculation of inconsistencies. Thus, we have the following properties at our disposal: *rdfs:range* and *rdfs:domain* whereas rdfs:range allows us to classify subjects that mean the first part of the triple and rdfs:domain the object, also the last part of the triple, in combination with the defined predicate. For example we want to extend our previously defined property *generateAttribute* with the restriction that the subject has to be a mashup and the object has to be an attribute.

```
<ns:generateAttribute
  rdf:about="http://www.rdfs-example.org/example1#generateAttribute">
   <rdfs:range ns:Mashup/>
   <rdfs:domain ns:Attribute/>
</ns:generateAttribute>
```

OWL - Web Ontology Language

The Web Ontology Language (OWL) is the W3C recommendation for modelling of ontologies. In the last chapter RDF(S) was introduced but it will be shown that the OWL language is able to represent more knowledge and is more expressive. Thus, OWL is based on formal logic which allows also to deduce knowledge that is only implicitly represented. Although, OWL RDF syntax and OWL DL are not the only standards, in this thesis we will use only them because they conform to the already introduced RDF syntax. Because of the relatedness of RDF(S) and OWL RDF we will focus on the enhancing features of OWL. Thus, the concept of classes and properties are completely taken over but extend the relationships with new functionality. For reference and further reading the reader is referred to the W3C (2004a) OWL Reference, Hitzler et al. (2009), Rector et al. (2004) and Horrocks et al. (2007).

Header The OWL document description contains details about namespaces, versions and annotations. For instance, the next listing shows the example from the previous RDF chapter extended with the owl namespace as well as with a default namespace that is used for user defined resources.

```
<rdf:RDF
  xmlns: ="http://www.example.org/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
</rdf:RDF>
```

Classes, subclasses and individuals Additionally to the borrowed class concepts from RDF(S), OWL introduces some new constructs that are summarized in table 4.1.

Table 4.1: Mapping of OWL Complex Class Descriptions to Description Logic (DL) and First Order Logic (FOL). (Largely borrowed and extended from Eiter et al. (2008))

OWL complex class descriptions	DL syntax	FOL short representation
$owl:Thing$	$\top$	$x = x$
$owl:Nothing$	$\bot$	$\neg x = x$
$owl:intersectionOf(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$	$C_1(x) \wedge \dots \wedge C_n(x)$
$owl:unionOf(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$	$C_1(x) \vee \dots \vee C_n(x)$
$owl:complementOf(C)$	$\neg C$	$\neg C(x)$
$owl:EquivalentClasses(C_1 \dots C_n)$	$C_1 \equiv \dots \equiv C_n$	$C_i(x) \leftrightarrow C_j(x)\, for\, 1 \leq i < j \leq n$
$owl:SubClassOf(C_1\ C_2)$	$C_1 \sqsubseteq C_2$	$C_1(x) \rightarrow C_2(x)$

OWL has two predefined classes: *owl:Thing* and *owl:Nothing*. Per definition every class must be a direct or indirect subclass or member of the Thing class. The Nothing class has no instances or subclasses. The reason to assign (directly or indirectly) to class Thing is to state that something is *true*. If an inconsistency would be inferred, the concerned class would be automatically assigned to class Nothing, thus something what may not be.

OWL also provides logical class constructors that are borrowed from set theory: *intersectionOf*, *unionOf*, *complementOf*. Also equivalence relations are introduced in addition to sub class relations.

Properties In OWL properties are distinguished into two groups: *DatatypeProperties* and *ObjectProperties*. DatatypeProperties connect individuals with data types. ObjectProperties connect individuals with individuals. Additionally, to the SubPropertyOf, domain and range concept, for properties different characteristica could be defined. ObjectProperties can be the inverse, disjoint or the equivalent to another ObjectProperty. Other characteristics are basically:

- *Functional*: The property can have only one value for each instance, whereby the value can be a data type or an instance. That is the only characteristic that could be defined for a DatatypeProperty.
- *InverseFunctional*: The inverse of the property can have only one value for each instance.
- *Symmetric*: The property and its inverse coincide, so the relation is also directed from object to subject.
- *Asymmetric*: The property and its inverse never coincide, so the relation could never be directed from object to subject.
- *Transitive*: If a transitive property links from A to B and from B to C then also A and C are linked with the property.
- *Reflexive*: The property relates the individual to itself.
- *Irreflexive*: Irreflexive means that no individual can be related by this property to itself.

Restrictions OWL enables that classes could be described based on the relations that individuals of the class participate in. These necessary or allowed relations are denoted as restrictions and can be distinguished into quantifier restrictions, cardinality restrictions and restrictions that denote which value the property has. In this thesis we restrict ourself only on the *for all* (∀) and *exists* (∃) quantifiers that are shown in table 4.2.

Table 4.2: Mapping of OWL Complex Class Restrictions to Description Logic (DL) and First Order Logic (FOL). (Largely borrowed from Eiter et al. (2008))

OWL complex class descriptions	DL syntax	FOL short representation
$owl{:}restriction(P\, owl{:}someValuesFrom(C))$	$\exists P.C$	$\exists y.P(x,y) \wedge C(y)$
$owl{:}restriction(P\, owl{:}allValuesFrom(C))$	$\forall P.C$	$\forall y.P(x,y) \supset C(y)$

An existential restriction ("some values from" or ∃) is the kind of restriction where each of the assigned individuals of the class must have at least one relationship, that is defined in the restriction, to a member of the specified class. For instance, each "mashup instance produces some values from attributes" would be translated according to set theory: each individual of the class *Mashup* must have at least one relation *produce* to an individual of the class *Attribute.*

A universal restriction ("all values from" or ∀) describes classes where each individual may have only a relation, that is defined in the restriction, to a member of the specified class. For example, each "mashup instance may only produce attributes" would be translated in: each individual of the class *Mashup* has only a relation *produce* to an individual of the class *Attribute.*

Describing and Defining a Class The concept of axioms, which basically are subclass relations and restrictions, is meant to provide semantics for classes as well as properties. In general in an ontology meaning is defined with subclass relations and restrictions, whereas a restriction also describes anonymous classes that are then super classes of the restricted classes. Thus, the subclasses of the restrictions may only contain all the individuals that fulfil the relationship requirements and the subclasses may not have conflicting axioms defined. A reasoner is able to understand the implicit logic of the whole ontology and is aimed to check the validity, that no inconsistencies between the relations, axioms and definitions occurred. Furthermore, a reasoner is also able to infer hidden relations that are not explicitly defined, by associating class and property identifiers with partial or complete formulations of their characteristics as shown in table 4.3.

Table 4.3: Mapping of OWL Complex Class Reasoning Concepts to Description Logic (DL) and First Order Logic (FOL). (Largely borrowed from Horrocks et al. (2007))

OWL complex class descriptions	**DL syntax**	**FOL short representation**
$owl{:}Class(C\,partial\,C_1 \ldots C_n)$	$C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$C(x) \rightarrow (C_1(x) \wedge \ldots \wedge C_n(x))$
$owl{:}Class(C\,complete\,C_1 \ldots C_n)$	$C \equiv C_1 \sqcap \ldots \sqcap C_n$	$C(x) \leftrightarrow (C_1(x) \wedge \ldots \wedge C_n(x))$

Partial or complete definitions are realized through subclass or equivalence relationships respectively and consist of a sequence of restrictions and (partial) other classes.

Partial definition means that a subclass (or the individuals of this subclass respectively) of that definition must fulfil one or more *necessary conditions.* Partially defined classes are also called described classes. Thus, by using a partial definition, it is not possible to infer the opposite direction, or in other words, from the conditions alone it does not entail that a class and a described class share a subclass relation if the class fulfils the conditions of a described class. This approach requires a manual assignment of all the described subclasses to the partial definition, by specifying the definition as super class of the restricted subclasses. The reasoner then will verify if the conditions hold for the subclasses.

Complete definition extends the necessary condition to a *necessary and sufficient condition.* That means that every class that fulfils the conditions that are specified for the defined class, thus basically a reference or template class, will be classified as subclass of the defined class. By complete definition, we talk about *defined classes* and compared with partial definitions, the entailment holds for each direction, thus both sides are equal and each side could be inferred from the other side. This approach is to be distinguished from partial definitions by the fact that every class will be checked if it fulfils the conditions of the defined class, it will automatically be classified by the reasoner as equivalent with the defined class.

Open World Assumption (OWA) OWL adhere to the OWA that is basically the typical applied assumption for monotonic logics. Databases are usually interpreted in the Closed World Assumption (CWA). The usage of OWA requires a change of thinking from closed to open world reasoning.

> The Closed World Assumption (CWA) "states that everything which is not explicitly true is considered to be false."
>
> The Open World Assumption (OWA) "leaves things undefined, i.e. something which is not explicitly stated to be the case - or not the case - is considered to be unknown." (Hitzler et al., 2009)

The whole concept in ontologies is to describe and define what is true. Databases follow the same idea but it is assumed that a database is considered to be complete, thus it comprises all the relevant knowledge. In an environment where the OWA holds, everything is true unless it is not proven as false. Hence, the descriptions and definitions of classes have to be carefully adapted for the OWA. Mostly, it is stated what exists or what must exist that a condition is fulfilled. However, due to the OWA it must also be stated that only these actual definitions and descriptions may exist. Thus, a description or definition looks like the following:

$$owl{:}restriction(P \; owl{:}someValuesFrom(C_1)) \tag{4.1a}$$

$$owl{:}restriction(P \; owl{:}someValuesFrom(C_2)) \tag{4.1b}$$

$$owl{:}restriction(P \; owl{:}allValuesFrom(unionOf(C_1, C_2)) \tag{4.1c}$$

Equations (4.1a) and (4.1b) state that what must exist when the necessary or necessary and sufficient condition is fulfilled. Equation (4.1c), that is realized with a for all ($\forall$) quantifier, denotes a closure axiom that states that the relation P may only build to the specified classes C_1 and C_2 to satisfy the condition. Thus, closure axioms can only restrict the defined relationship, each other relationships that may not occur have to be restricted in particular. If a relation may not occur in general for this class, the relation has to be restricted to $owl{:}Nothing$.

The same has to be considered for the classes that have to be determined per partial or complete definitions. That is basically due to the OWA that would without the closure axioms assume that the class or individuals respectively could have further relations defined. Especially, by automated classification it is a common failure that the closure axiom has been forgotten for the determining class and hence the reasoner will not classify the class as from designer desired.

Due to the complexity and uncommon concept, the interested reader is referred to the publications Hitzler et al. (2009) and Rector et al. (2004) for examples and more informations.

Part II

Prototype

CHAPTER 5

Security Model

5.1 The complete approach

Within the scope of this thesis, a security model for the usage of mashups in organisational context has been developed. The basic idea of this approach is to provide a secured and controlled platform where every staff member could design their own applications or mashups respectively that comply with the business rules of the company. This compliance will be checked by the platform system. In order to meet these requirements there are several issues that have to be considered. Figure 5.1 shows a graphical representation of the schematic approach.

The whole concept of mashups is aimed at gathering data, processing them according to the needs of the designer and generating the desired output that could then be represented by style sheets or by copying it in a template. Thus, in order to fulfill their actual tasks, mashups are able to collect the necessary data. However, the data of an organisation is one of the most valuable goods of that organisation. The whole problem is therefore a trade-off between the security of the data and the simple usage of mashups. When a staff member design a mashup and it could not access the necessary data the mashup would be useless due to fact that it cannot execute or perform the deliberated data processing. Otherwise, the data of the company could be exploited by careless handling or malicious purposes. Therefore we need a more finely grained protection mechanism.

However, how the data is actually protected is not in the scope of this thesis. It is only necessary that the *Secure Mashup Environment* could access the data whereas every direct access from a user is not permitted. One could argue that the data has to be protected by building an API where each user could only access the data for which the actual user has been granted access rights. For realizing this approach the whole data has to be transferred in a well-structured database, the API has to be build and

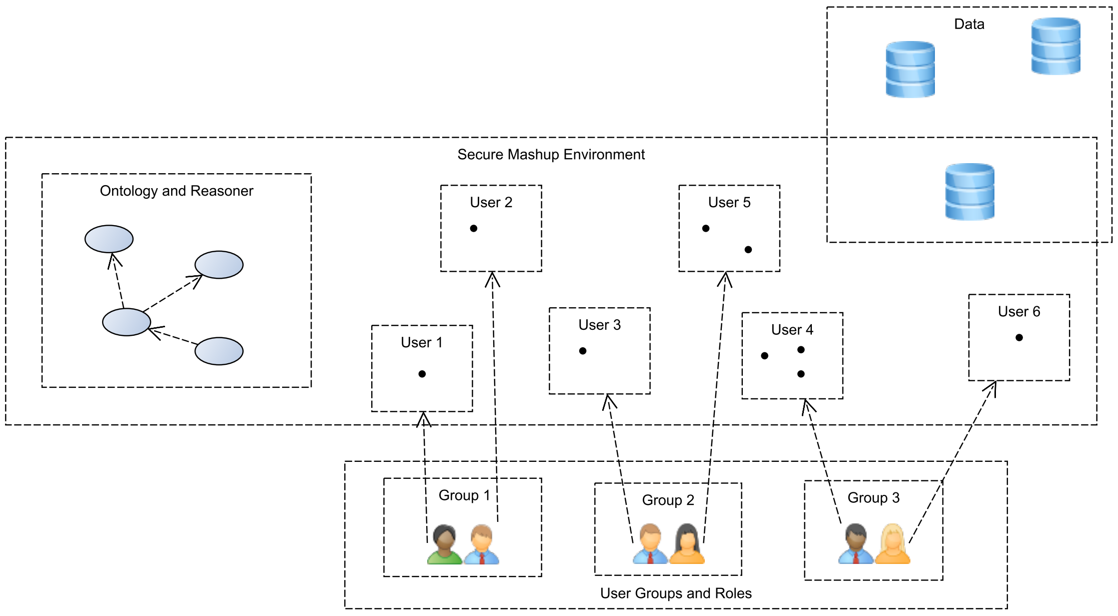


Figure 5.1: The simplified representation of the proposed security model for mashup usage in organisational context.

the whole system has to be maintained. Through a lot of data from different sources like CSV, XML and different databases as well as the fact that the amount of enterprise data is steadily increasing over time, that means an enormous effort which stands against the principle that the users build his/her own applications. Additionally, the needs of users usually change or additional needs emerge respectively. For every change request or requirement of a new method, the API must be adapted so that for example the average of the annual revenue for the last five years could be calculated and requested from the specific user. When another user then needs another data source the API must again be adapted. These simple data transformation tasks are in the scope of a mashup though. Thus, if the mashup could access the data, the data processing could be performed by the mashup and would be independent from the developer staff. However, each data processing operation can generate sensitive data, even for data for which the users have total granted access rights. It is therefore necessary to control the architecture of the mashup. Afterwards it is hardly possible to infer how the actual outcome has been processed. One option could be to parse the complete output and check for suspicious data tuples. However, that would mean that the semantics of the data must be taken into account, otherwise it would not be possible to check for all privacy/security related possibilities.

Another approach that is analysed in this thesis, is that already the semantics of the operations are checked and also conform to the established policies of an organisation. Hence, the *Secure Mashup Environment* has the task to secure the access rights on the whole data. It needs to consider the granted rights of user to actual data as well as which operations may be performed on the data. This information is inferred from the ontology which is embedded in the Secure Mashup Environment. In the ontology also complex rules could be formulated like the validation a set of data tuples that could be in combination lead to inference attacks, or in other words, someone could infer some hidden knowledge from the published data that is not allowed to become public. The fundamental idea is to provide an intelligent system that checks the mashups of the users on compliance with the established business rules and provides an environment where the organisation can trust its own staff by letting them write their own mashup solutions. How to describe mashups with ontologies and reason about them is being described in the next section.

5.2 Mashup reasoning

In the scope of this thesis, we define mashup reasoning as a process that allows computational decisions if the mashup conforms to the business policies of the organisation or locates the malicious operations based on the semantics of the control flow or architecture of the mashup respectively. The output of a mashup is basically defined as a set of attributes, which is formalized in equation (5.1).

$$mashup\_result := \{attr_0, attr_1, attr_2, \ldots, attr_{n-1}\} \tag{5.1}$$

Likewise, the input of a mashup is basically a set of attributes. The semantics of these attributes may at first be well-known and defined in an ontology, could be deduced from the names of the attributes, or defined as RDF meta data as subtext, etc. However, by uncontrolled processing of the data the semantics can get lost, for instance the average of certain data is calculated and from the name of the table or column it can not be deduced that it is an average value and from which data it was calculated. For example, the average of the last 5 years of annual revenue might be calculated and published but the average of profit may not. If the resulting attribute is named 'average' it can be deduced that the result is an average value but it can not any longer be inferred to which original values it belongs. Thus, in the proposed approach the architecture of the mashup is described in an ontology that comprises the meaning of the data operations as well as the architecture of the mashup. The whole concept how to describe a mashup is structured in the following sections:

- **Taxonomy:** delimitation of the mashup operations and building of a taxonomy;
- **Control Flow:** controlling the control flow of the operations;
- **Description of Operations:** describing the semantics of the operations;
- **Access Rights:** describing which operations a user may perform.

Taxonomy

Depending on the mashup script language, application area and actual use-case, a mashup is composed of several data processing operations and services. Thus, the first step has to be that a taxonomy of the allowed operations is designed. In figure 5.2 a simplified taxonomy is illustrated in order to simplify the explanation of the approach and put the proposed approach into practice. A short description of the concrete operations is given in table 5.1.

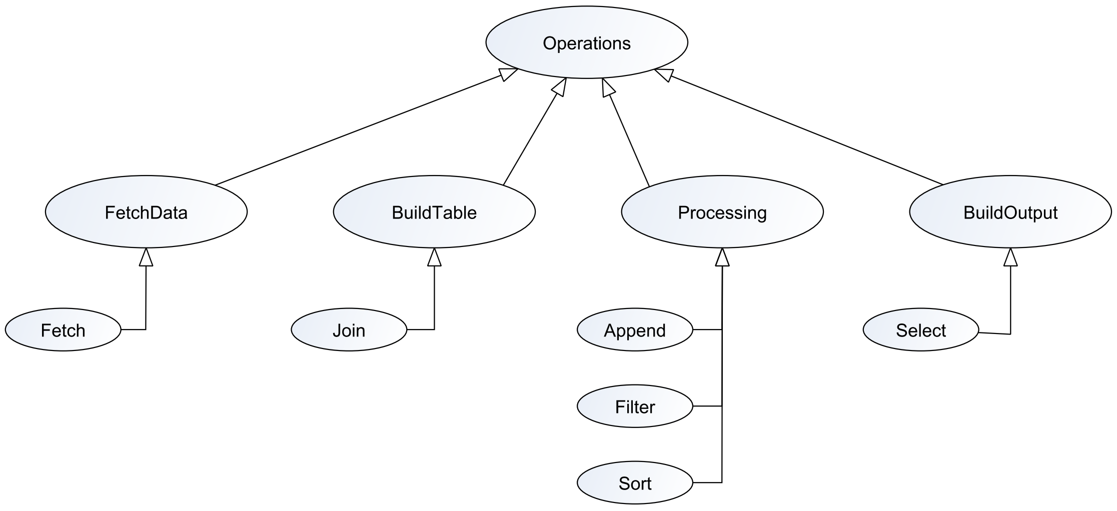


Figure 5.2: Taxonomy of a selection of (abstract) mashup operations that are used to describe the whole approach. The concrete operations like fetch, merge, join are *typed* with super types so that a hierarchy (or taxonomy respectively) of operation types is established.

Table 5.1: A short description of the (abstract) operations that are chosen for explanation of the approach.

Operation	Description
Fetch	Get a table of data from a declared data source like a Web service or file.
Join	Join two tables at the declared attribute.
Append	Append a (calculated) attribute to an existing table.
Filter	Filter a table according to a specified expression.
Sort	Sort an attribute or table respectively, according to an expression.
Select	Select only the attributes (columns) that are given as parameters.

Taxonomies already provide meaning or semantics, respectively. Thus, for each operation it can be entailed that it also has the type of the super type. Thus, all axioms that are defined for the super types must also hold for the actual subtype. One usage of that concept will be described in the next section.

Control Flow

A typical architecture of a mashup originates from the operation types and their corresponding sequence in the mashups. Hence, not only the operation types has an impact on the result, also the order of those operations. For instance, thinking of the operations *join* and *select*, if only non sensitive columns are chosen and another table is joined afterwards, it would imply that the joined table may still contain sensitive attributes which then are returned to the executer of the mashup. If the select operation is the last operation in the control flow before the output, then it will be possible to determine that only non sensitive columns are selected. In figure 5.3 a recommended control flow for operations that are defined in the simplified taxonomy is illustrated.

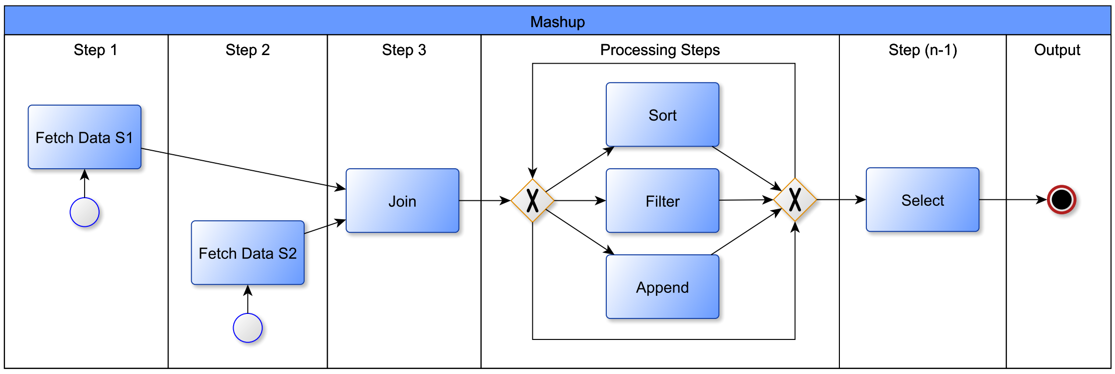


Figure 5.3: Proposed control flow of a simple mashup in BPMN syntax with two data sources (abbreviated with S). The fetch operations have to be performed at the beginning and the returned tables are joined afterwards. Then the joined table could be further processed and with select operation only allowed attributes are be chosen for the actual output based on the description of the select operation.

Thus, the sequence of the operations has to be described in the ontology. In an ontology an axiom or the truth respectively, is realized with statements in the form $< Subject\ predicate\ Object >$. The order of operations can be described as a successor system or more specifically, each $< Operation\ hasSuccessor\ Operation >$. The determined order, that means which concrete operation may follow after another operation, could be enforced as restrictions. Using the specified system type of the taxonomy, the necessary restrictions could be applied on the super classes of the concrete operations. If the real sequence of a mashup is then identified and transferred in the ontology, the order must conform to the described restrictions. Otherwise a reasoner would infer an inconsistency as explained in chapter 4.3 by the reference to partial definitions. Figure 5.4 illustrates how the proposed approach is converted in an ontology. Considering the characteristics of the properties or relations of the operations respectively, it will be necessary to determine that *hasSuccessor* is transitive. With this extension it can be deduced from *Operation_A hasSuccessor Operation_B* and *Operation_B hasSuccessor Operation_C* that also Operation_C is a successor of Operation_A.

The restrictions that are emphasized in blue in figure 5.4 are converted in DL Language in (5.2a) - (5.2d). In the following, universal restrictions are emphasized in blue and existential quantifier in red. As is shown, the restrictions are set as only (∀), so it is stated that there may or may not be a successor operation, but if there is, for example Operation_A has a successor Operation_B then Operation_B has to be in the specified set of possible successors of Operation_A. For instance, FetchData can *only* have a successor operation that is of type FetchData, BuildTable or Processing. The set of possible successors is build with *unionOf* (⊔) so that the successor must be in the class FetchData *or* (∨) BuildTable *or* (∨) Processing. The whole approach is to establish a well-ordering of a mashup's operations. Thus, it must be possible that after a fetch operation the extracted data can be returned (and all other operations are skipped), the data could be processed before the actual output is built or if two fetch data operations are executed then it must be possible to join the data. However, as stated before, the well-ordering should forbid that a join table operation is executed after the last select, otherwise it would be impossible to establish a rule that only the allowed attribute columns may return to the user.

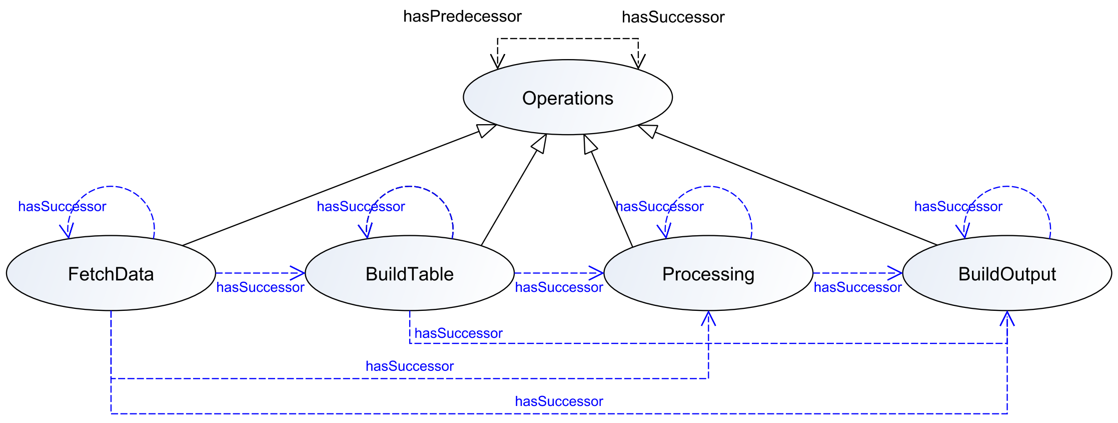


Figure 5.4: Proposed realization of the *description of the control flow* in an ontology. Please note that this control flow restrictions has to be adapted depending on the programming language, use-cases, etc. In the proposed realization, a fetch data as successor of build table is not possible but could be realized with two mashups.

$$FetchData \sqsubseteq \forall hasSuccessor.(FetchData \sqcup BuildTable \sqcup Processing \sqcup BuildOutput) \tag{5.2a}$$
$$BuildTable \sqsubseteq \forall hasSuccessor.(BuildTable \sqcup Processing \sqcup BuildOutput) \tag{5.2b}$$
$$Processing \sqsubseteq \forall hasSuccessor.(Processing \sqcup BuildOutput) \tag{5.2c}$$
$$BuildOutput \sqsubseteq \forall hasSuccessor.BuildOutput \tag{5.2d}$$

As is indicated in the figure, a successor relation also has the inverse relation *hasPredecessor*. If this characteristic is defined for the *hasSucessor* relation, a reasoner can infer for each operation which predecessor/predecessors it has. However, for the proposed taxonomy the predecessor relation has no specific use-cases but can be defined for some operations, that need another operation as precondition before execute on.

Description of Operations

As stated at page 51, to describe the architecture of a mashup the operation type has to be considered, or more specifically, what the operations really do. In figure 5.5 we just consider one operation in general. From the BPMN syntax we can deduce that an operation has at least one input and exactly one output. Although it is not visible in the figure we can also state that an operation might depend on one or more parameters that determine or affect the operation's result. However, since each operation performs another algorithm, its overall result can only be inferred from the operation's specific name, its parameters and input. Thus, each single operation has to be described with its details such as the restrictions and what might be done with these operations must be specified for all operations in detail. Considering that normally users have different granted rights, a specific operation must be more than once described, because, for instance, one user might only have the rights to perform the operation with different parameters than other users.

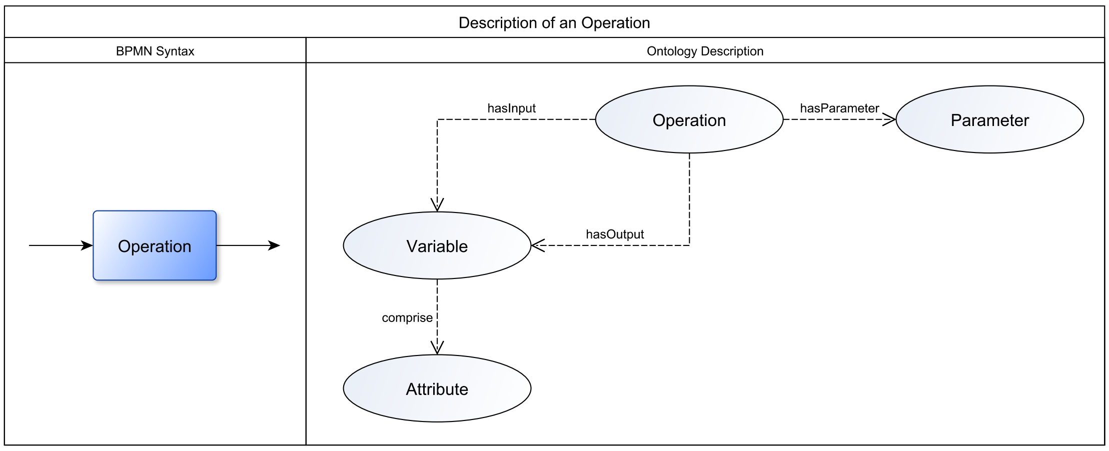


Figure 5.5: Generalized description of a mashup's operation transferred and extended from BPMN syntax.

Hence, in the following two examples or use-cases of operation descriptions will be presented, one for the filter operation and one for select operation.

Description of Operations - Example 1 A mashup filter operation returns rows of a table where one or more tuples match with a declared expression. Let us assume that in a health care institute the staff is able to write their own mashup solutions for daily tasks and the accessible data table would be of the form:

$$\{ID,\ Name,\ Date\_of\_Birth,\ Address,\ Sex,\ ZIP,\ Disease\} \tag{5.3}$$

A doctor has to be able to filter for all kinds of attributes so that he/she is able to deduce from the other patients characteristics, such as age, weight, sex, location, etc., on the actual patients disease. However, this task would not be necessary for a reception secretary but it would be rather unethical to allow such operations for that staff group. Thus, we have to describe two specific kinds of classes for filter instances, one unrestricted kind for the doctors and one for the rest of the staff, where the parameters for the filter operation are restricted on *ID* and *Name*. Figure 5.6 shows how those operations could be described with an ontology.

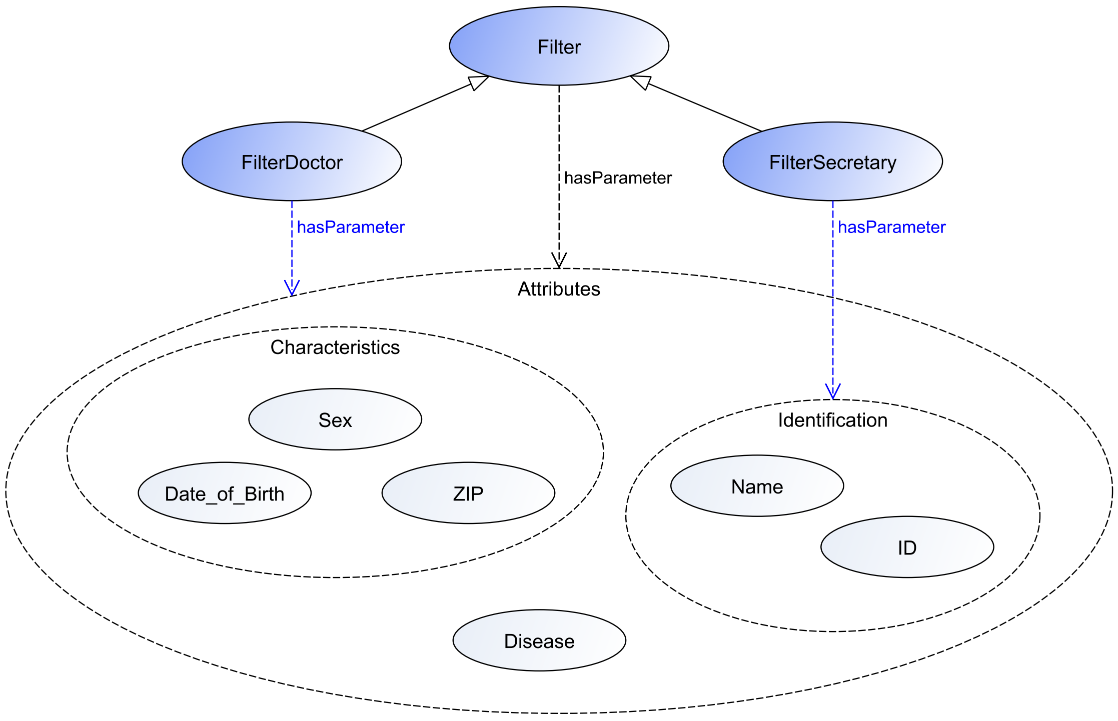


Figure 5.6: Description of a filter operation according to Example 1.

$$\begin{aligned} Filter &\sqsubseteq \exists hasParameter.Attribute \quad &(5.4a)\\ FilterDoctor &\sqsubseteq \forall hasParameter.Attribute \quad &(5.4b)\\ FilterSecretary &\sqsubseteq \forall hasParameter.Identification \quad &(5.4c)\end{aligned}$$

The equations (5.4a) - (5.4c) show the previously described property restrictions. Equation (5.4a) states that for each filter operation a *hasParameter* relation to the class Attribute must *exist* (∃). This restriction is inherited by the subclasses *FilterSecretary* and *FilterDoctor*. Equation (5.4b) states that, additionally to the inherited restrictions, each instance that is in the class of *FilterDoctor* *only* (∀) may have parameters that are from type attribute. Equation (5.4c) states that for FilterSecretary the hasParameter relations may *only* be built between the two specific attributes *ID* and *Name* that are subsets of class Identification.

Description of Operations - Example 2 As mentioned before with the select operation it is possible to state which columns (attributes) are returned to the user. Consider this operation based again on the health care example from above. A doctor needs unrestricted access to all attributes, otherwise he/she could overlook a specific detail that could lead to the correct prognosis of a disease. A secretary should not know from which disease the patient is suffering but he/she is responsible to fill in the personal information or check the previous things for correctness. Figure 5.7 shows the proposed relations and the necessary equations for these restrictions (similar to Example 1) are defined in equations (5.5a) - (5.5c).

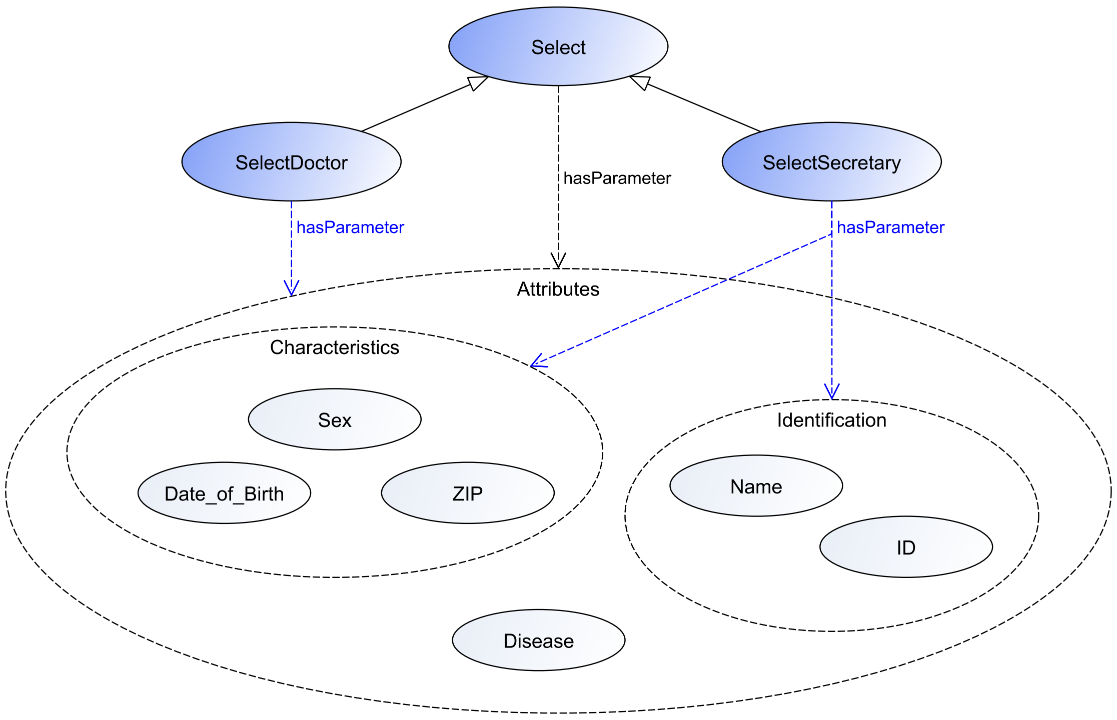


Figure 5.7: Description of select operations that have to be executed so that only allowed attribute columns are returned to the user according to Example 2.

$$Select \sqsubseteq \exists hasParameter.Attribute \tag{5.5a}$$
$$SelectDoctor \sqsubseteq \forall hasParameter.Attribute \tag{5.5b}$$
$$SelectSecretary \sqsubseteq \forall hasParameter.(Identification \sqcup Characteristics) \tag{5.5c}$$

Additionally, let us assume that sometimes publications are published to make a contribution to the overall health care system. Mostly, in these publications it is analysed which specific characteristics lead to a specific disease (Fung et al., 2010). It is clear that in these publications the *ID* and *Name* of the patients may not be mentioned, however even from a collection of attributes someone can guess the real identity of a person. A collection of general attributes that together build an identifier is called Quasi-Identifier (QID). For instance, a person could be potentially identified from the characteristic set: *Date_of_Birth*, *Sex* and *ZIP* (Fung et al., 2010). Hence, a new operation for publishing has to be described. Name and ID as concrete identifier must be forbidden and also that a QID is built from the data set. However, if only two of them are used as characteristics it has to be possible, otherwise evaluations on characteristics that lead to a disease would be impossible to publish. One way to describe such a select operation is shown in figure 5.8 and the formalisations of these restrictions are shown in the equations (5.6a) - (5.6b).

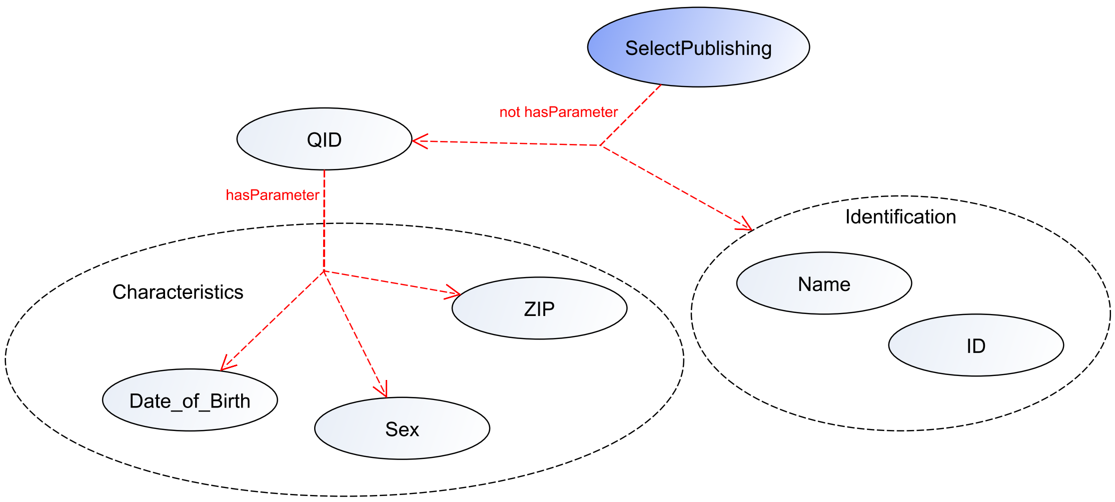


Figure 5.8: Ontology concept that aims to prevent QID according to Example 2.

$$QID \sqsubseteq \exists hasParameter.(Date\_of\_Birth \sqcup Sex \sqcup ZIP) \tag{5.6a}$$
$$SelectPublishing \sqsubseteq \neg(\exists hasParameter.(Identifier \sqcup QID)) \tag{5.6b}$$

Equation 5.6a describes a new class *QID* that comprises all classes that have at least one relation to the unified set $Date\_of\_Birth \sqcup Sex \sqcup ZIP$. Equation 5.6b describes that relations from the select operation to the classes ID, Name (comprised of class Identification) and a QID are forbidden. Please observe that this time the negation is used so that the specified relations to this classes are excluded and therefore not allowed. Thus, the attributes *Disease* and also *Date_of_Birth, Sex and ZIP* are allowed, as long as these last three attributes are not used together.

In modern age, an inference attack just like the identification of a person based on an set of attributes is a delicate subject in privacy. For more related examples and use-cases, the reader is referred to Fung et al. (2010), Liu and Yang (2012) and Minami and Borisov (2010).

Access Rights

As indicated before, each user group in an organisation normally has different granted rights for data access and how to process that data. That is typically realized by implementing an application or basically an API where all users are operating at the same database but can only access that data by the predefined logic layer. In the methods of the logic layer the queries for databases access and how the accessed data is processed is (statically) specified. In the proposed approach, those rights and procedures have to be described by the ontology. Therefore, together with the previously introduced concepts, we are considering basically three cases, that are illustrated in figure 5.9.

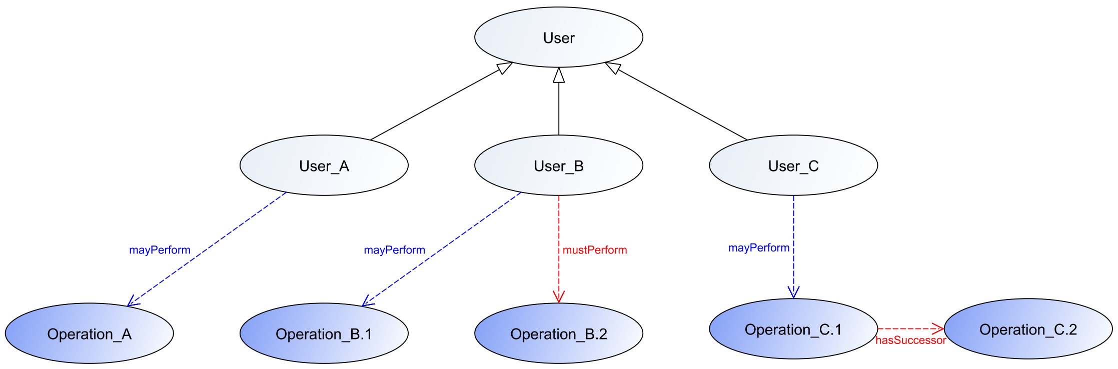


Figure 5.9: Abstract use case definitions of execution rights in an ontology.

Description of Access Rights - Example 1 Considering only User_A we build a closed system where it is stated which exact operations that user may perform. For instance, referring again to the health care example, a doctor has the rights to access the internal database and perform the operations FilterDoctor and SelectDoctor (cf. Equation (5.7a)) whereas a secretary may perform FilterSecretary and SelectSecretary (cf. Equation (5.7b)).

$$Doctor \sqsubseteq \forall mayPerform.(Fetch \sqcup FilterDoctor \sqcup SelectDoctor) \tag{5.7a}$$
$$Secretary \sqsubseteq \forall mayPerform.(Fetch \sqcup FilterSecretary \sqcup SelectSecretary) \tag{5.7b}$$

In open world reasoning for such sensitive restrictions it is required that it is described what is allowed whereas the option to forbid operations would involve the risk that operations could be forgotten so that data could be exploited, whereas the later release of operations may cause inconvenience due to the lack of functionality but it is typically saver in practical application.

Description of Access Rights - Example 2 The restrictions on User_B describe a scenario where it is helpful that additionally to the operations the user may perform it is also stated which operations must be performed. For instance, let us assume that in an organisation the accessible data sources are described with trust levels. An organisational policy could then be that the average trust level of all data sources in a mashup must be equal or over a specified value. The organisation could now provide a mashup operation that fulfils such a task but each user of the organisation has to perform that task anywhere in their implementation. Thus, a rule must be installed which states that such an operation must be performed, at least once (cf. Equation 5.8a).

$$User \sqsubseteq \exists mustPerform.Validate\_Trust\_Level \tag{5.8a}$$

Description of Access Rights - Example 3 Considering the restrictions on User_C we can observe that it could be defined that a user has the possibility to execute an operation but afterwards another operation has to be performed. For instance, considering the example with the QID again, where the set of characteristics *Date_of_Birth*, *Sex* and *ZIP* leads to possible identification of a person. Instead of forbidding the usage of the three characteristics altogether, the data could be anonymized. Fung et al. (2010) provides a survey for the interested reader, summarizing and evaluating different techniques and approaches for Privacy-Preserving Data Publishing (PPDP). For this example, let us assume that health care data should be published and the established policies stipulate that by usage of that a QID the data has to be k-anonymized.

> "k-anonymity: if one record in the table has some value qid, at least k-1 other records also have the value qid. In other words, the minimum group size on QID is at least k. A table satisfying this requirement is called k-anonymous. In a k-anonymous table, each record is indistinguishable from at least k-1 other records with respect to QID." (Fung et al., 2010)

This anonymization approach works by generalizing the data in comprising super classes. For example, a ZIP code '1010' could be generalized to 'Vienna' and the birth data '19.07.1987' could be substituted by the interval [1980 - 1990].

The simplest way to enforce that rule would be for the organisation to provide a service or predefined operation that performs the anonymization logic and enhances the select operation for publishing by the following restriction:

$$SelectPublishing \sqsubseteq \exists hasSuccessor.Anonymization \tag{5.9a}$$

Based on the previously defined well-ordering of operations and defined transitivity law of the property, the restrictions state that the anonymization operations have to be executed but must not interfere with the control flow. Thus, the operation could be inserted, in this case after the final select operation. Another use case could be the inverse hasPredecessor, so that for operations a necessary condition or predcessor could also be defined.

CHAPTER 6

Proof of Concept

6.1 General information

Within the scope of this thesis, a prototype for the security model has been developed, which should demonstrate in form of a testing environment for mashups.

In this context, a Web application demonstrates the concept of the central security environment where a user can submit mashups written in EMML. The background of the Web application is basically a *peer review system*, which is explained in the next subchapter in more detail.

The data is stored in XML format and, for simplicity, at the Web application's server. The Web application's logic has the responsibility to enforce the policies of the organisation so that the mashup will be executed and delivers the result to the user or if the mashup violates the organisationally defined rules the user will be informed. The role of the user is freely selectable for each test procedure. The logic is based on extracting operations, parameters and the operations' sequence by XML parsing, extended with the parsed information's embedded ontology and on evaluating if the instances adhere with the defined restrictions or policies respectively.
This chapter is structured in the following parts:

- Background information for the peer review system, the data structure and the business rules which have to be enforced by the system;
- A survey about EMML and how the embedded ontology had been built with Protégé;
- Information about the implementation Web application and the conducted experiments to show the proposed approach.

Peer review

According to Do (2003), in academic fields peer review is an important procedure, ensuring quality and valid knowledge and decreasing the rate of rejected publications. In general, a review process is started with the submission of the publication to an editor. The editor selects the reviewers in the same area of research. The task of the reviewers is to write a detailed feedback that should be worked on in the publication of the author in order to significantly improve and enhance the publications, within one or several submissions, to ensure fairness, avoidance of distraction and uniform standards. To achieve these goals as good as possible, the practice of single blind and double blind reviews is often applied:

- *Single blind* means that the authors may not see the identity of the reviewers.

- *Double blind* means that additionally to single blind the reviewers may not see the identity of the authors.

Data structure of the environment

The data of the peer reviewer system has to be distinguished from the Web application's data. The Web application has its own database which includes all the necessary information for temporary storage, etc. The environment data is generally described as all the information that could be mashed up by a user of the system. The following data model (cf. figure 6.1) should emphasize the possibility that the staff member can design their own applications without the constant support of professional developers. Thus, the data is stored in two big XML files for the reviewers and reviews of the system. The structure of these files is shown in figure 6.1 and the listings 6.1 and 6.2. Let us assume that these files are constantly changing and that the effort to maintain the files in a database would not cover the expenses. However, the files could contain important information for the staff but without security measures the risk of a user generating illicit data is unacceptable.

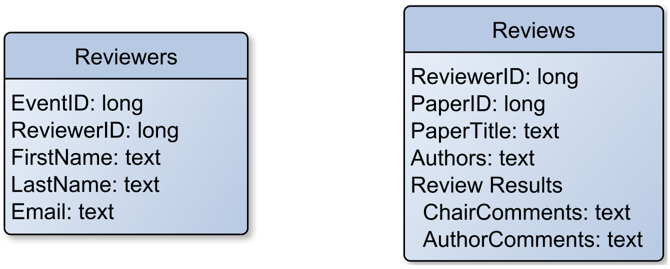


Figure 6.1: Data model of the Secure Mashup Environment. The tables illustrate well structured XML files that contain the variables as attributes.

```
<Reviewer>
  <ReviewerID> ... </ReviewerID>
  <Name> ... </Name>
  <Email> ... </Email>
</Reviewer>
...
```

Listing 6.1: Structure of Reviewers.xml

```
<Paper>
  <PaperID> ... </PaperID>
  <ReviewerIDs>
    <ReviewerID> ... </ReviewerID>
    <ReviewerID> ... </ReviewerID>
  </ReviewerIDs>
  <PaperTitle> ... </PaperTitle>
  <AuthorsOfPaper> ... </AuthorsOfPaper>
  <ReviewResults>
    <ChairComments> ... </ChairComments>
    <AuthorComments> ... </AuthorComments>
  </ReviewResults>
</Paper>
...
```

Listing 6.2: Structure of Papers.xml

Although the data structure of the proposed Proof of concept is not realized as a relational database the structure of the XML is similar to the attributes, records and tuples. Thus, it is possible to apply the definitions of the mashup concept that are introduced in chapter 5. In the following the data will be named as attributes according to the meaning of the data. For instance, the attribute PaperID means all the paper IDs of the file.

Business rules

As Proof of Concept the following polices had to be covered and enforced by the Ontology:

- The authors are not allowed to see reviewers' names
- Reviewers are not allowed to see authors' names
- For privacy reasons, the output of a mashup must not disclose email addresses

That means the essential context only comprises the *role* of the user. The role of the end-user should be freely selectable so that the implications of each role on the result of the reasoning procedure could be tested.

6.2 Mashup Language

Enterprise Mashup Markup Language (EMML)

The mashup samples are implemented in EMML which is an XML-based programming language. The XML scripts result in well-defined documents, which could be translated in an executable program. EMML was developed by the Open Mashup Alliance (OMA) and is an open language for the development of enterprise mashups (Open Mashup Alliance). This approach is based on XML and it describes the processing flow of a mashup. Thus, the user is able to define variables, parameters, data sources, etc. and statements, which serve as functions and execute actions. (Omelette, 2011)

EMML provides a lot of predefined features, which are illustrated in the table 6.1. The general concept of EMML is based on XPath expressions and XSLT stylesheets, to address data and apply functions on it. Additionally to the program statements, EMML documents could include meta data. On one hand, predefined meta data could be used to pass on processing statements to the Mashup Engine, whereas on the other hand, custom meta data could also be defined and could be used by applications connected with the mashup. (Omelette, 2011)

Table 6.1: The instruction library of EMML. (Largely borrowed from (Omelette, 2011))

Fetch	Mash	Enrich	Control	IO
<directinvoke>	<filter>	<append>	<if><else>	<input>
<invoke>	<sort>	<construct>	<while>	<output>
	<group>	<annotate>	<for>	<display>
	<join>	<assign>	<foreach>	<variable>
	<merge>		<parallel>	
	<select>		<sequence>	

In this chapter, the test samples of the Proof of Concept will be explained. At first, the general construction and concepts will be explained. Then, in the explanation of the conducted experiments, all the test mashups used to prove the security model will be introduced briefly.

The header of the sample mashup always follows the same procedure and could be embedded in a template. The first part is required for the indication of name spaces and general information such as the name of the mashup. Then the meta information about the user role is directly stored in the script. In the following template script it would be an author:

```
<mashup
  xmlns:
    xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openemml.org/2009-04-15/
      EMMLSchema ../schemas/EMMLSpec.xsd"
  xmlns="http://www.openemml.org/2009-04-15/EMMLSchema"
  name="Template">

<emml-meta name="author">Author</emml-meta>

...
</mashup>
```

Listing 6.3: Header of an EMML script which defines all necessary name spaces as well as the emml-meta operation for embedding the context information in the mashup script.

The operation for the used approach to gather data from a XML is not listed in table 6.1 and is realized in the PoC with the macro: XMLConstructorFromFile that loads the whole data from a XML file in a variable.

```
<macro:XMLConstructorFromFile
  filename="reviewers.xml"
  outputvariable="$ReviewerSource"
/>
```

Listing 6.4: Fetch data from XML file operation in EMML.

The control flow of the EMML mashups can be calculated by taking the sequence or order, of the operations into account. However, it is possible that the mashup has been designed with several branches or with operations, which have no effect on the actual output. For building the mashup instruction pipe it is necessary to consider that each operation has an input and an output variable. An output variable could be reused by a successor operation or a new branch could be started with a new variable. Together with the sequence of operations it is now possible to construct the control flow tree with all its branches and loops. Another basis of comparison is a directed graph.

In the following, we will consider the relations of a mashup as a graph. The relations as well as sequences of the operations are designed with directed edges. The parameters of an operation are designed as undirected edges.

6.3 Ontology

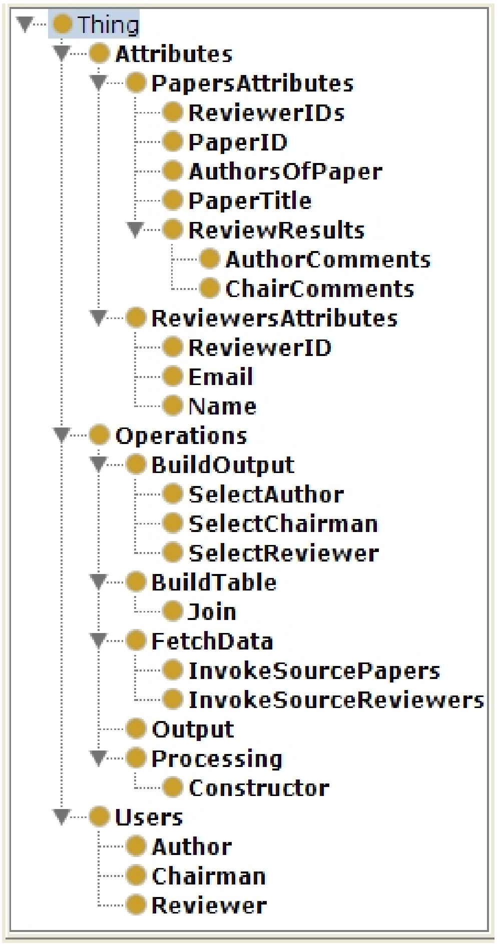
The ontology embedded in the application is designed with Protégé. Protégé is an ontology editor from the Stanford university (Stanford University). It is an open source project and written entirely in Java.

The implementation phase of the ontology has been structured in the following steps: First, the classes were defined and by giving them a hierarchical structure a taxonomy was built. Second, the object properties were prepared. At last, the restrictions were set that enabled the semantic reasoner to infer if the mashups conform with the business policies.

Building the Taxonomy

The taxonomy comprises all the necessary concepts for the applied reasoning, which basically divided into four top categories attributes, operations, data sources and user roles.

Protégé provides an intuitive interface for designing the taxonomy. The operations are defined similar to chapter 5 but adapted so that it conforms with the EMML operation names and peculiarities. The attributes basically serve as the operation's parameters, for the attributes of the files that could be loaded with the operation *invoke*. Thus, an invoke operation could be distinguished by the declared source. The user defined roles author, chairman and reviewer are providing the context for the operations. For instance, a selectAuthor may only be executed by an author. The hierarchical structure of the classes will later allow that restrictions could be set for the superclasses which are automatically inherited by the subclasses. In figure 6.2 the complete taxonomy is illustrated in form of a hierarchically structured tree.



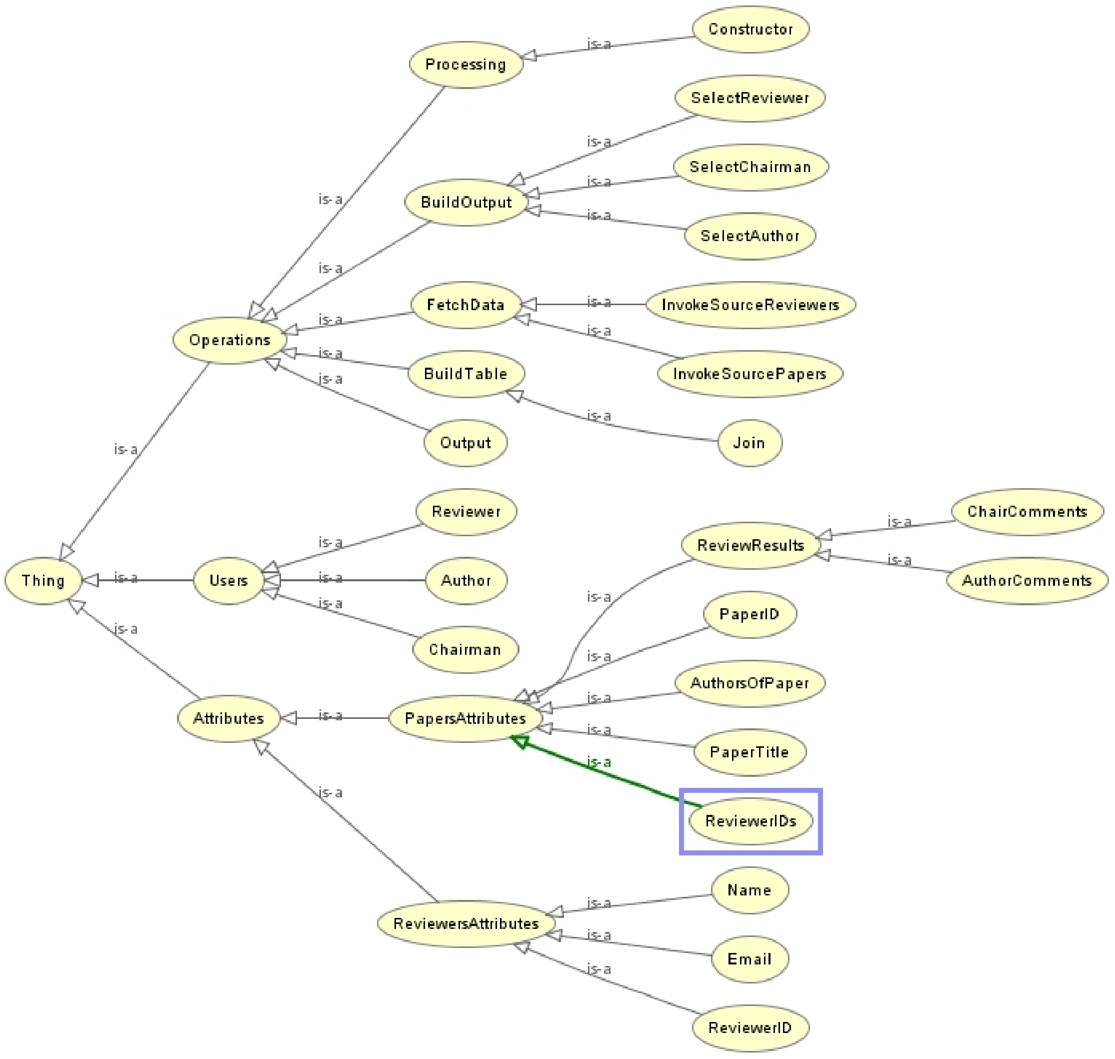
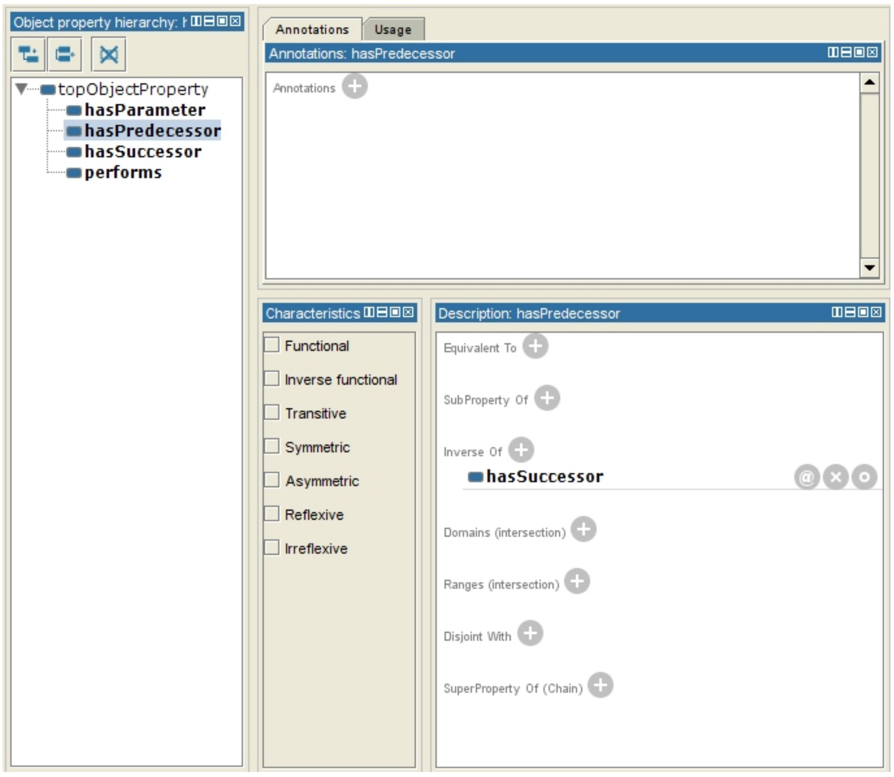


Figure 6.2: Visualization of the Taxonomy of the mashup driven peer review system. (Created with OWLViz plug-in)

Configuration of the Properties

Protégé provides a distinct tab for the definitions of properties (cf. figure 6.3). The properties describe the relationships between the different classes. For the peer review system, four different properties have been defined, *hasParameter, hasPredecessor, hasSuccessor and performs*. The property *hasSuccessor* and *hasPredecessor* are the inverse of each other and should realize the well-ordering of operations that have been proposed in chapter 5. The property *hasParameter* is a key factor in order to decide if a specific role may execute an operation with the restriction for this property. For instance, a select operation where each parameter is allowed, except the email address, can only be executed by a chairman. The property *performs* is necessary to restrict which specific operations may then be executed by each role.



topObjectProperty
- hasParameter
- hasPredecessor
- hasSuccessor
- performs

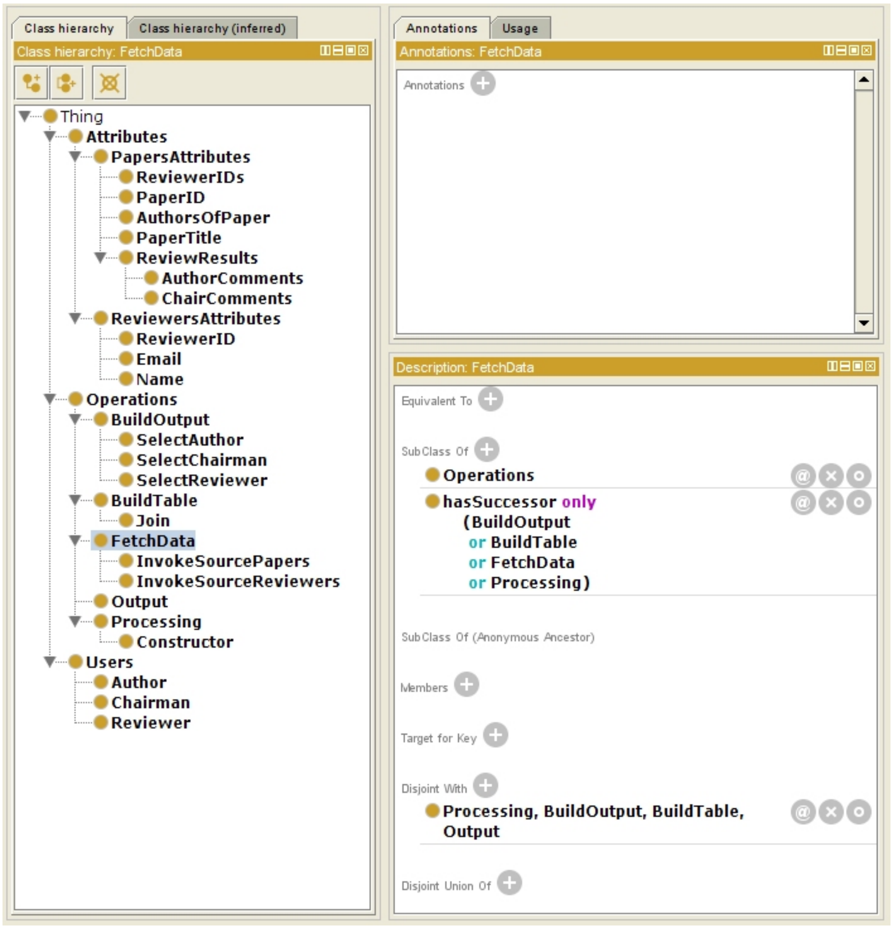
Inverse Of
hasSuccessor

Figure 6.3: The object property tab of Protégé and the defined properties for the peer review system.

The implemented Web application uses object properties to describe the relations between the instances. For example, used for instances created by the operations are the *hasSuccessor* to describe the actual control flow of the operations. The relations are built as triples as described in section 4.

Configuration of the restrictions

The restrictions could be set in the classes tab of Protégé. Figure 6.4 illustrates the interface for the restriction configuration. Thus, they could be set up as anonymous super classes in the section *SubClass Of*, as illustrated in the figure. The inherited restrictions are $FetchData \sqsubseteq \forall hasSuccessor.(FetchData \sqcup BuildTable \sqcup Processing \sqcup BuildOutput)$. These restrictions are also inherited from all subclasses of FetchData, in our case *InvokeSourcePapers* and *InvokeSourceReviewers*, as well as all individuals of the classes. Afterwards, the implemented system generates individuals that represent all characteristics of the mashup itself as well as the chosen user role. The whole idea is that non of the individuals, or more precisely the relations of the individuals, do not violate the restrictions of its superclass. Otherwise, a reasoner can compute the inconsistency in the resulting ontology.



```
Thing
  Attributes
    PapersAttributes
      ReviewerIDs
      PaperID
      AuthorsOfPaper
      PaperTitle
      ReviewResults
        AuthorComments
        ChairComments
    ReviewersAttributes
      ReviewerID
      Email
      Name
  Operations
    BuildOutput
      SelectAuthor
      SelectChairman
      SelectReviewer
    BuildTable
      Join
    FetchData
      InvokeSourcePapers
      InvokeSourceReviewers
    Output
    Processing
      Constructor
  Users
    Author
    Chairman
    Reviewer
```

SubClass Of

Operations

hasSuccessor only (BuildOutput or BuildTable or FetchData or Processing)

Disjoint With

Processing, BuildOutput, BuildTable, Output

Figure 6.4: Interface provided by Protégé for the configurations of the restrictions.

In the following, all restrictions that have been set up will be shortly listed and explained what they are intended for.

Operations

$$FetchData \sqsubseteq \forall hasSuccessor.(FetchData \sqcup BuildTable \sqcup Processing \sqcup BuildOutput) \tag{6.1a}$$
$$BuildTable \sqsubseteq \forall hasSuccessor.(BuildTable \sqcup Processing \sqcup BuildOutput) \tag{6.1b}$$
$$Processing \sqsubseteq \forall hasSuccessor.(Processing \sqcup BuildOutput) \tag{6.1c}$$
$$BuildOutput \sqsubseteq \forall hasSuccessor.(BuildOutput \sqcup Output) \tag{6.1d}$$

Restrictions (6.1a) - (6.1d) are similar to the proposed restrictions in chapter 5, only that they are extended with the *Output* class for EMML, and together they realize the proposed order of the operations.

$$SelectAuthors \sqsubseteq \forall hasParameter. (AuthorsOfPaper \sqcup PaperTitle \sqcup AuthorComments) \tag{6.2a}$$
$$SelectReviewer \sqsubseteq \forall hasParameter.(Name \sqcup PaperTitle) \tag{6.2b}$$
$$SelectChairman \sqsubseteq \neg(\exists hasParameter.Email) \tag{6.2c}$$

Restrictions (6.2a) - (6.2c) specify SelectAuthors, SelectReviewers and SelectChairman as *described classes* (cf. chapter 4). Based on the declared Attributes of the hasParameter relationship, the select operation, the reasoner can determine if the select operation of the chosen user role is valid or not. The particular name of the owl class can be concatenated with a String manipulation using the name of the select operation and the user role in question. Those restrictions should illustrate the described operations as explained in chapter 5.

User Roles

$$Author \sqsubseteq \exists performs.SelectAuthors \sqcap \neg\exists performs.InvokeSourceReviewers \tag{6.3a}$$
$$Reviewer \sqsubseteq \exists performs.SelectReviewer \tag{6.3b}$$
$$Chairman \sqsubseteq \exists performs.SelectChairman \tag{6.3c}$$

Restriction (6.3a) defines that an author has to perform a select author operation. Furthermore, it is defined that an author must not perform the InvokeSourceReviewers. Restriction (6.3b) and (6.3c) are used to enforce the SelectReviewer or SelectChairman operation respectively, at the end (due to the well-ordering) of the process for reviewers. We can compare these restrictions with the proposed access rights restrictions in chapter 5.

6.4 Automated Reasoning

Basically, for the implementation of the reasoning part the following languages and frameworks have been used:

- **JAVA**: Java (Oracle) is a popular and highly used object-oriented programming language.

- **Wicket**: Wicket (Apache, b) is a framework for building Web applications.
- **JUNG**: JUNG (O'Madadhain et al.) is a library for designing graphs and is used for the graphical representation of the evaluated mashup pipe.
- **Jena**: Jena (Apache, a) is a framework for semantic Web applications, providing all the ontology related functionality that is required in the proposed approach.
- **Pellet**: Pellet (Clark & Parsia) is a sophisticated reasoner used to evaluate if the restrictions hold for the tested mashups.

The architecture of the PoC is illustrated in figure 6.5.

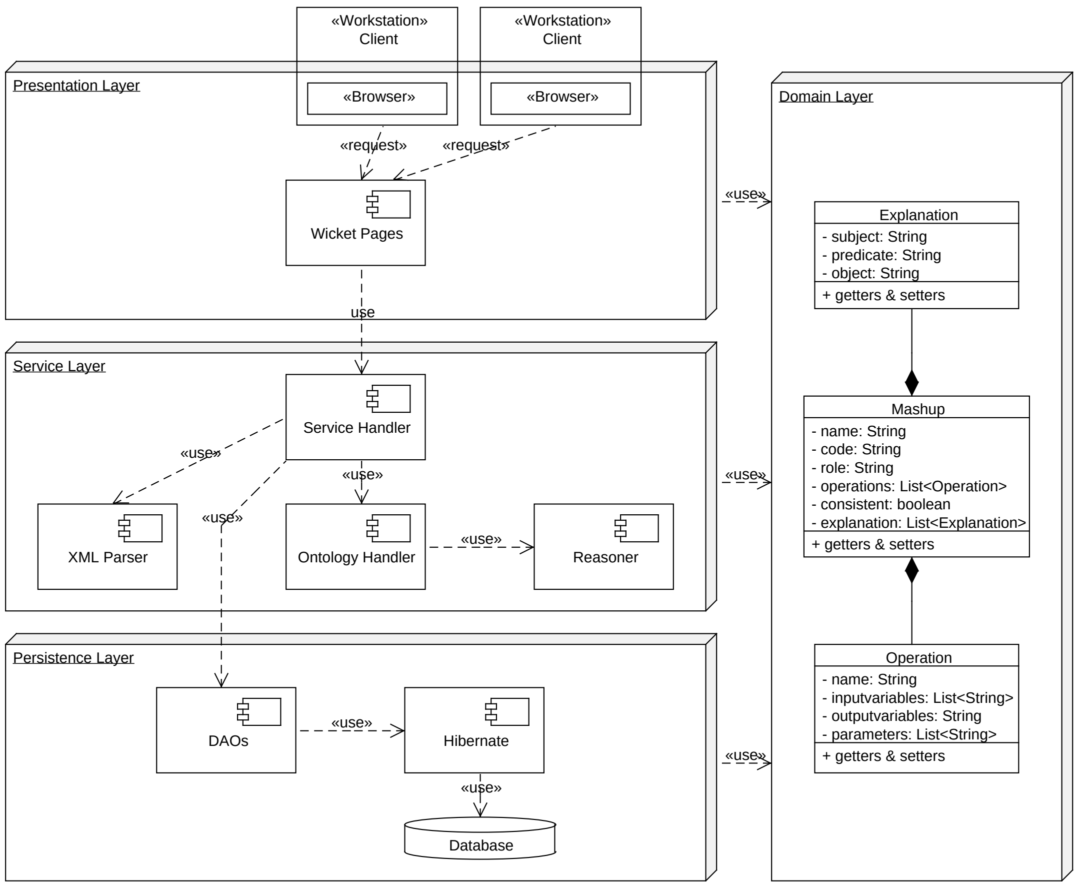


Figure 6.5: Proposed architecture of the proof of concept implementation. (Created with UMLet)

For the testing, two views have been implemented. In figure 6.6 the User Interface for the upload of mashups is shown. Figure 6.7 illustrates the view for the testing of

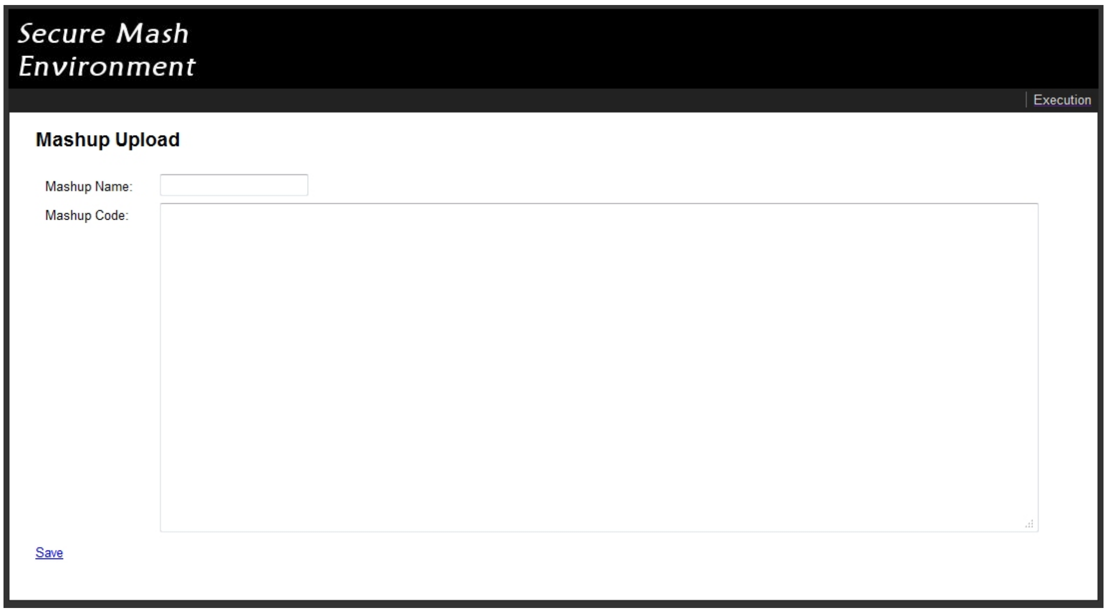
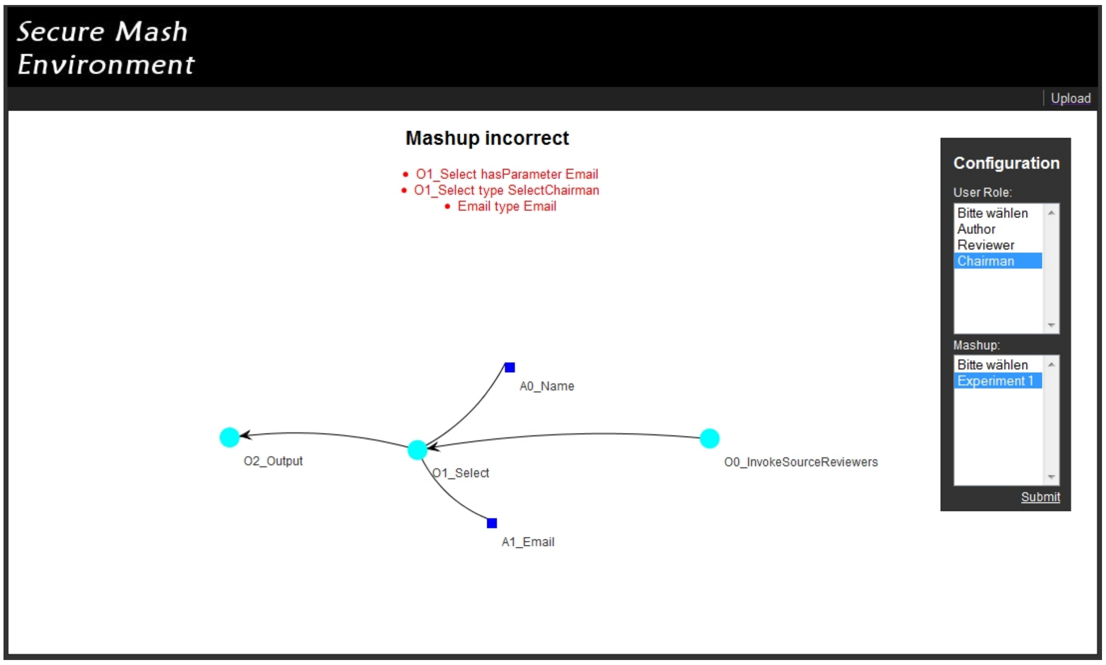


Figure 6.6: User Interface for the upload of mashups that could be afterwards in the execution view tested.



Mashup incorrect

- O1_Select hasParameter Email
- O1_Select type SelectChairman
- Email type Email

Figure 6.7: User Interface for testing the uploaded mashups with different user roles freely selectable.

the uploaded scripts. Therefore, in the configuration menu, the user role could be chosen as well as script that should be tested. The output of each test procedure includes if the mashup is valid and the parsed relations that are visualized in as graph. In the case that the script does not adhere to the defined restrictions, the computed statements of the reasoner are printed.

Every time the architecture of a mashup should be evaluated by the PoC implementation, through the implementation process and the steps that are illustrated in figure 6.8.

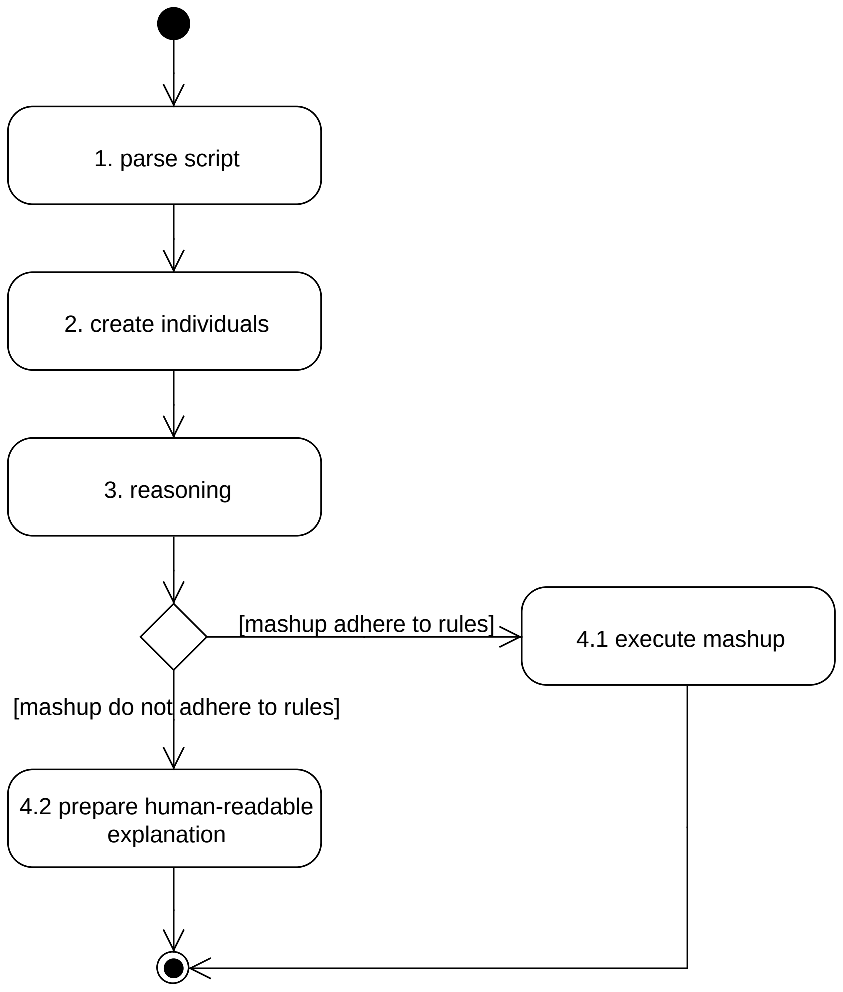


Figure 6.8: Proposed approach of the proof of concept implementation. (Created with UMLet)

1. Parse the EMML script and filter the operations, parameters and input/output variables. The information can be stored in simple domain objects. Afterwards, from the actual sequence of the operations and the input/output variables the graph of the operations and parameters are computed (cf. figure 6.9, 6.11 and 6.14).
2. Create concrete individuals from the previously parsed information that describe the context information, operations and parameters. These individuals have to be dedicated to the predefined classes in order to allow automated reasoning.
3. The reasoning procedure is performed by an embedded reasoner. The result of the reasoner will determine the next step.

4.1. If the mashup script fulfils the predefined rules, the mashup should be executed.

4.2. If the mashup does not adhere to the predefined rules, a human-readable explanation is computed by the embedded reasoner.

For the sake of simplicity, we will concentrate in the following on the logical reasoning. Listing 6.5 presents the code of the implementation of the reasoner. Due to the ability of Pellet to reason about inconsistent ontologies (Parsia et al., 2005), the pellet reasoner API has been used in order to detect individuals or relations that do not adhere to the defined policies or restrictions respectively. To ensure that Pellet can explain the inconsistencies after the reasoning procedure, it has to be configured by setting the Pellet option *USE_TRACING* on true.

The lines 11-13 propose how the inference model of the ontology is created. The inference model is the result of applying the reasoner on the fundamental ontology. It contains all entailed knowledge about the ontology, e.g., subclass relationships that are inferred from defined classes. Then, subsequently, we can compute if the inferred model of the ontology is consistent. Hence, if a predefined rule is violated, we can iterate over the axioms in order to find statements that do not comply with the predefined axioms. However, only the individuals that do not comply with the restrictions and their relations should be printed as explanation for the user due to the fact that we consider all axioms as correct. Certainly, also failures in the ontology design could be possible but would require manual examination of the rules. Finally, all statements that concern individuals have to be prepared in a human-readable form. As mentioned in section 4, ontologies work with URIs that are inconvenient for humans to read and understand. Thus, we decompose the statements into subject, predicate and object and are using only the local names.

The following subsections will illustrate the reasoning procedure based on three EMML mashup examples. The experiments have been conducted with the Web Application that had been implemented to prove the proposed approach. The appendix comprises the User Interface screenshots of the particular experiments.

```java
public class PelletReasoner implements OntologyReasoner {

  public boolean compute(OntModel model, Mashup mashup) throws
      OWLOntologyCreationException, Exception {

    final List<ExplanationStatement> explanationList = new ArrayList<
        ExplanationStatement>();

    PelletOptions.USE_TRACING = true;

    final Reasoner reasoner = PelletReasonerFactory.theInstance().create();

    // create an inferencing model using Pellet reasoner
    final InfModel infModel = ModelFactory.createInfModel(reasoner, model);
    final PelletInfGraph pellet = (PelletInfGraph) infModel.getGraph();

    // evaluate if the mashup adhere to the predefined rules
    if (!pellet.isConsistent()) {

      // create explanation
      final Model explanation = pellet.explainInconsistency();

      // iterate over the axioms in the explanation
      for (Statement stmt : explanation.listStatements().toList()) {

        // The predefined axioms of the rule set have to be considered as true.
        // Hence, the false individual triples will be printed as explanation.
        if (isIndividual(stmt)) {
          final ExplanationStatement expl = new ExplanationStatement(
            stmt.getSubject().getLocalName(),
            stmt.getPredicate().getLocalName(),
            stmt.getObject().asResource().getLocalName());
          explanationList.add(expl);
        }
      }
    }

    mashup.setExplanationList(explanationList);
    return pellet.isConsistent();
  }

  private boolean isIndividual(Statement statement) {
    Property property = statement.getPredicate();
    RDFNode node = statement.getResource();

    return (!(BuiltinTerm.find(property.asNode()) != null) ||
      !(BuiltinTerm.find(node.asNode()) != null) && property.equals(RDF.type));
  }
}
```

Listing 6.5: Adapted reasoner class of the PoC implemented in JAVA and by using the Pellet as reasoner inference engine.

6.5 Experiments

Experiment 1

The script of the tested mashup is illustrated in listing 6.6. The experiment should show that in the case of an author being the writer of the mashup, the mashup should be evaluated as invalid. That is according to restriction (6.3b) which states:
$\neg\exists performs.InvokeSourceReviewers$ or in other words the script is invalid due to the restriction that an author may not perform an operation that is classified as *InvokeSourceReviewers*. The *macro:XMLConstructorFromFile* is translated into the name of ontology class *InvokeSourceReviewers* with a simple name mapping and, in the same manner, for the other single operations and parameters as well. For each of the parsed operations and parameters, individuals are created and statements are built by connecting them with the object parameters *hasParameter* and *hasSuccessor*. The individuals and their relations are visualized as a graph, as is illustrated in figure 6.9. Also, for the user role an individual is created and related with the operations with the object property *performs*.

```
<mashup>
...
  <macro:XMLConstructorFromFile filename="reviewers.xml"
    outputvariable="$ReviewerSource"/>

  <select inputvariable="$ReviewerSource"
    outputvariable="$ReviewerSource" selectexpr="/data/reviewer">
    <columns>
      <column>Name</column>
    </columns>
  </select>

  <output name="ReviewerSource" type="document"/>

</mashup>
```

Listing 6.6: EMML mashup script 1: The mashup script contains three operations. First, it is instructed that the *reviewers.xml* has to be loaded. Afterwards, from the loaded table the name and the email address column should be selected and printed as output.

The implementation of the reasoner (cf. listing 6.5) computes an inconsistency, in the case that a author is stated as writer of the mashup script. The prepared result is in such a case (cf. figure 6.10):

1.) Author type Author

2.) Author performs O0_InvokeSourceReviewers

3.) O0_InvokeSourceReviewers type InvokeSourceReviewers

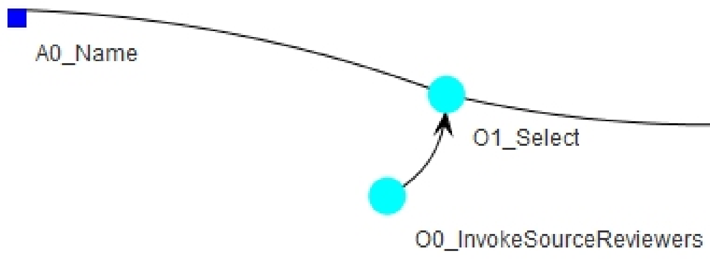
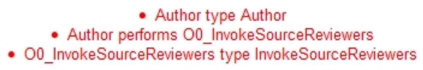


Figure 6.9: Automated visualization of the individuals and their relations in the first experiment. (Created with JUNG)



- Author type Author
- Author performs O0_InvokeSourceReviewers
- O0_InvokeSourceReviewers type InvokeSourceReviewers

Figure 6.10: Computed (human-readable) explanation for experiment 1.

The computed result of the reasoner agrees with the assumption that the script does not adhere with restriction (6.3b). Line 1.) and 3.) provide information about the comprising classes and 2.) explains that the printed statement does not comply with the defined mashup polices. However, the mashup is also invalid due to the *Name* parameter of the *Select* operation.

Experiment 2

This experiment is similar to Experiment 1 (cf. section 6.5), with the exception that the restricted operation sequence is violated with a second data fetching operation that overrides the output variable before the results are published. In this case, the operation does not adhere to the restriction (6.1a) that states that an operation classified as *FetchData*, must not have a *hasSuccessor* relation to operation that is classified as *Select*. The automatically created graph in figure 6.11 visualizes the relations of the described operations and parameters.

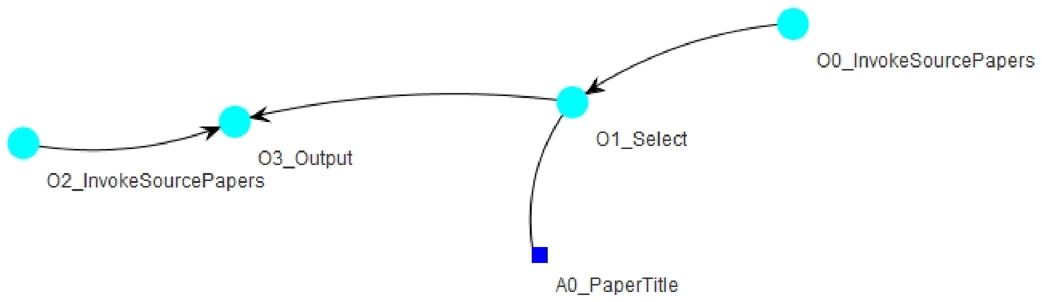


Figure 6.11: Automated visualization of the individuals and their relations in the second experiment. (Created with JUNG)

```
<mashup>
...
  <macro:XMLConstructorFromFile filename="papers.xml"
    outputvariable="$PaperSource"/>

  <select inputvariable="$PaperSource"
    outputvariable="$PaperSource" selectexpr="/data/papers">
    <columns>
      <column>PaperTitle</column>
    </columns>
  </select>

  <macro:XMLConstructorFromFile filename="papers.xml"
    outputvariable="$PaperSource"/>

  <output name="PaperSource" type="document"/>

</mashup>
```

Listing 6.7: EMML mashup script 2: The script is similar to listing 6.6 but a second fetch data operation instructs that the output variable should be overwritten with all the unselected data from the papers.xml file just before the result is published.

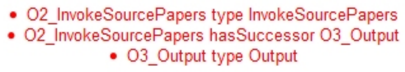
The implementation of the reasoner (cf. listing 6.5) computes an inconsistency without influence of the chosen user role. The prepared result is (cf. figure 6.12):

1.) O2_InvokeSourcePapers type InvokeSourcePapers

2.) O2_InvokeSourcePapers hasSuccessor O3_Output

3.) O3_Output type Output

The computed result of the reasoner agrees with the assumption that the script does not adhere to restriction (6.1a). Similar to the result from 6.5, line 1.) and 3.) provide information about the comprising classes of the individuals and 2.) explains that the printed statement does not comply with the defined mashup polices.

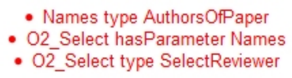


- O2_InvokeSourcePapers type InvokeSourcePapers
- O2_InvokeSourcePapers hasSuccessor O3_Output
- O3_Output type Output

Figure 6.12: Computed (human-readable) explanation for experiment 2.

Experiment 3

In the third experiment, we will examine what happens if we rename attributes, as shown in listing 6.6. Listing 6.8 is similar to 6.6 but with the exception that the variables are renamed with the constructor operation. In such a case, the direct super class of the renamed attribute is fetched and a new individual is created for that class. Hence, the new individual holds all inherited restriction of the super class. Restriction (6.2b) forbids a reviewer to select the *AuthorsOfPaper*, thus, the same has to apply to *Names* because it is a new individual that is classified as *AuthorsOfPaper*.



- Names type AuthorsOfPaper
- O2_Select hasParameter Names
- O2_Select type SelectReviewer

Figure 6.13: Computed (human-readable) explanation for experiment 3.

```
<mashup>
...
  <macro:XMLConstructorFromFile filename="papers.xml"
    outputvariable="$PaperSource"/>

  <constructor outputvariable="PaperSource">
    <Names>{$PaperSource//AuthorsOfPaper}</Names>
    <Title>{$PaperSource//PaperTitle}</Title>
  </constructor>

  <select inputvariable="$PaperSource" outputvariable="$PaperSource"
    selectexpr="/data/reviewer">
    <columns>
      <column>Names</column>
      <column>Title</column>
    </columns>
  </select>

  <output name="PaperSource" type="document"/>

</mashup>
```

Listing 6.8: EMML mashup script 3: The script is similar to listing 6.6 but the attributes are renamed with the constructor operation before the columns are selected.

The implementation of the reasoner (cf. listing 6.5) computes an inconsistency, if a reviewer is selected as writer of the mashup. The prepared result is (cf. figure 6.13):

1.) Names type AuthorsOfPaper

2.) O2_Select hasParameter Names

3.) O2_Select type SelectReviewer

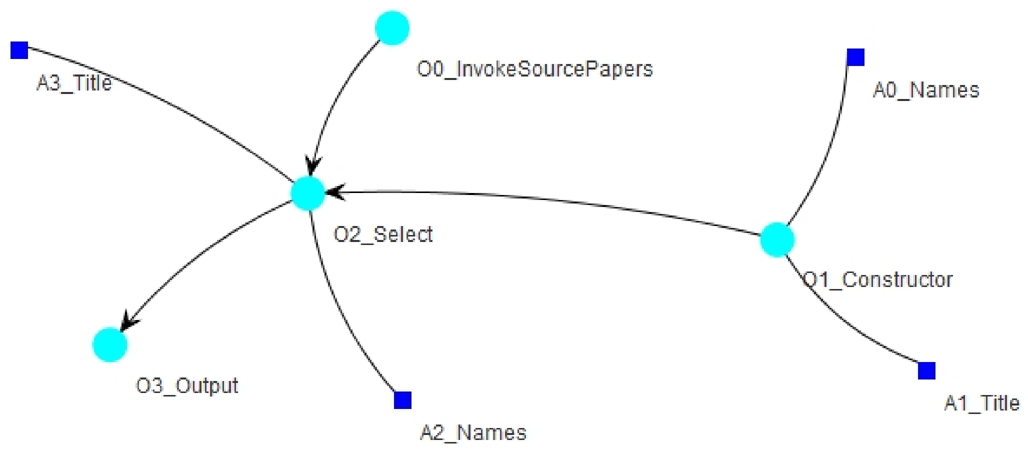


Figure 6.14: Automated visualization of the individuals and their relations in the third experiment. (Created with JUNG)

The computed result of the reasoner agrees with the assumption that the script does not adhere to restriction (6.2b). The output is the same as the case without the "rename" operation, with a simple difference that *Names* is printed instead of AuthorsOfPaper as label for the individual. However, the reasoner provides the type information of the attribute in the first statement.

CHAPTER 7

Summary & Conclusion

Mashups are simple and handy applications that allow sophisticated tasks to be easily automated. For the development of mashups, various editors and frameworks exist that enable even non-programming or inexperienced people to write their own applications. In organizational context, mashups are denoted as Enterprise Mashups. Enterprise Mashups could have a major impact on limited resources, the Long Tail of User Needs, the enterprises' agility & flexibility and the enterprises' knowledge management. However, with all the possibilities mashups can offer, the whole concept of mashups also has drawbacks to security, privacy and trust issues.

Semantic Web Technologies provide a way to describe the real world so that even machines could process and infer new knowledge from the described knowledge. Taking sophisticated ontology languages into account, it is also possible to restrict what may exist in the ontologically represented world. In this thesis, the mashup architecture, comprising the operation sequence, the individual operation types and the parameters of each operation, is described with ontologies. Hence, the following machine understandable statements were proposed[1]:

- $\langle Operation\rangle\ hasSuccessor\ \langle Operation\rangle$
- $\langle Operation\rangle\ hasParameter\ \langle Attribute\rangle$
- $\langle Role\rangle\ performs\ \langle Operation\rangle$

These statements describe semantically which relations exist between the operations of the mashup script and what parameters are used to configure a particular operation. The "performs predicate" describes all the operations a user has specified in the script. A taxonomy provides meaning for all types of roles, attributes and operations. The

[1]The angle notation indicates classes. All sub-classes of the specified class can be used instead of the super class, thus, it enables us to state what information we extracted from a mashup script.

ontology-based approach provides also possibilities to restrict these proposed relations, so that an enterprise is able to control how the end-users can design their mashups. Furthermore, some particular use cases are explained which emphasize the implications of this approach on security and privacy related issues, in order to prevent data leakage and semantic aggregation by using a reasoner which checks the extracted information with the established policies of the organization.

Within this thesis, a Proof of Concept implementation of the proposed approach has been developed. The implemented application illustrates how mashup scripts could be described with relations or statements respectively, and automatically be evaluated by reasoning about the consistency of the ontology. The PoC covers background knowledge about the underlying peer review system, the data structure and the used mashup language. Furthermore, the ontology building and implemented Web application are explained as well as the conducted experiments that prove the proposed approach.

From the experiments we can learn that the proposed approach could be used to prevent malicious mashup compositions that violate the defined policies or restrictions respectively. However, the ontology can only cover what is defined for our domain. That means, everything that is forgotten in the design of the ontology will not be enforced by the reasoning system.

With rising complexity of the whole system or the enterprise's data structures respectively, the scaling of the ontology, comprising all classes, restrictions as well as individuals, has to be evaluated in further work.

Bibliography

G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *Proceedings of the international conference on Web search and web data mining*, pages 183–194. ACM, 2008.

A. Anjomshoaa, G. Bader, and A. M. Tjoa. Exploiting Mashup Architecture in Business Use Cases. In *2009 International Conference on Network-Based Information Systems*, pages xx–xxvii. IEEE, 2009.

Apache. Apache Jena. Website, a. Available online at `http://jena.apache.org/`; last visited on August 14th 2012.

Apache. Apache Wicket. Website, b. Available online at `http://wicket.apache.org/`; last visited on August 14th 2012.

G. Bader. *Exploiting business impact of Enterprise Mashup solutions.* PhD thesis, TU Vienna, 2012. [under review].

G. Bader, A. Anjomshoaa, and A. M. Tjoa. Privacy Aspects of Mashup Architecture. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, pages 1141–1146. IEEE Computer Society, 2010.

M. Barhamgi, D. Benslimane, C. Ghedira, and A. Gancarski. Privacy-preserving data mashup. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, pages 467 –474, 2011.

B. Beckman. Why LINQ Matters: Cloud Composability Guaranteed. *Queue*, 10:20:20–20:31, 2012.

S. Bloehdorn, P. Haase, Z. Huang, Y. Sure, J. Völker, F. van Harmelen, and R. Studer. Ontology Management. In J. Davies, M. Grobelnik, and D. Mladenic, editors, *Semantic Knowledge Management*, pages 3–20. Springer, 2009.

J. Breslin, A. Passant, and S. Decker. *The Social Semantic Web.* 2009.

R. Burke. *Project Management: Planning and Control Techniques.* J. Wiley, 1999. ISBN 9780471987628.

L. Cherbakov, A. J. F. Bravery, and A. Pandya. SOA meets situational applications, Part 1: Changing computing in the enterprise. Website. Available online at `http://www.aiim.org/community/wiki/view/Index-E`; last visited on April 14th 2012.

Clark & Parsia. Pellet: OWL 2 Reasoner for Java. Website. Available online at `http://clarkparsia.com/pellet/`; last visited on August 25th 2012.

F. De Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama. SMash: secure component model for cross-domain mashups on unmodified browsers. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 535–544, New York, NY, USA, 2008. ACM.

E. Y.-L. Do. Why Peer Review Journals. *International Journal of Architectural Computing*, page 253 – 266., 2003.

T. Eiter, G. Ianni, T. Krennwallner, and A. Polleres. Rules and Ontologies for the Semantic Web. In *Reasoning Web*, pages 1–53, 2008.

Forrester Research. What's The Social Technographics Profile Of Your Customers? Website. Available online at `http://empowered.forrester.com/tool_consumer.html`; last visited on April 24th 2012.

C. K. D. Frappaolo. Association for Information and Image Management: Enterprise 2.0. Website. Available online at `http://www.aiim.org/community/wiki/view/Index-E`; last visited on April 14th 2012.

B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, pages 14:1–14:53, June 2010.

T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, pages 907–928, 1995.

N. Guarino. Formal ontology, conceptual analysis and knowledge representation. *Int. J. Hum.-Comput. Stud.*, 43:625–640, 1995.

P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies.* 2009.

I. Horrocks, P. F. Patel-Schneider, D. L. McGuinness, and C. A. Welty. OWL: a Description Logic Based Ontology Language for the Semantic Web. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition)*, chapter 14. Cambridge University Press, 2007.

V. Hoyer. Enterprise Mashups. Website. Available online at https://sites.google.com/site/fastonlinecontest/what-are-enterprise-mashups; last visited on July 9nd 2012.

V. Hoyer and M. Fischer. Market Overview of Enterprise Mashup Tools. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 708–721, Berlin, Heidelberg, 2008. Springer-Verlag.

V. Hoyer and K. Stanoevska-Slabeva. Service-Oriented Computing — ICSOC 2008 Workshops. chapter The Changing Role of IT Departments in Enterprise Mashup Environments, pages 148–154. Springer-Verlag, Berlin, Heidelberg, 2009a.

V. Hoyer and K. Stanoevska-Slabeva. Towards a reference model for grassroots enterprise mashup environments. In *ECIS*, pages 2279–2290, 2009b.

V. Hoyer, K. Stanoesvka-Slabeva, T. Janner, and C. Schroth. Enterprise Mashups: Design Principles towards the Long Tail of User Needs. In *Services Computing, 2008. SCC '08. IEEE International Conference on*, pages 601 –602, 2008.

IBM. IBM Mashup Center Website. Website. Available online at http://www-01.ibm.com/software/info/mashup-center; last visited on April 7th 2012.

InetSoft. InetSoft - Mashup Driven Dashboards & Reporting. Website. Available online at http://www.inetsoft.com/; last visited on July 2nd 2012.

Intel. Intel Mash Maker. Website. Available online at http://software.intel.com/community/wiki/view/Index-E; last visited on June 29th 2012.

JackBe. Presto Mashup Composers. Website. Available online at http://www.jackbe.com/products/composers.php; last visited on June 12th 2012.

JackBe Corporation. A Business Guide to Enterprise Mashups. Technical report, 2008.

X. Liu and X. Yang. Protecting Sensitive Relationships against Inference Attacks in Social Networks. In S.-g. Lee, Z. Peng, X. Zhou, Y.-S. Moon, R. Unland, and J. Yoo, editors, *Database Systems for Advanced Applications*, Lecture Notes in Computer Science, pages 335–350. Springer Berlin Heidelberg, 2012.

A. P. McAfee. Enterprise 2.0: The Dawn of Emergent Collaboration. *MITSloan Management Review*, pages 21–28, 2006.

M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja - Safe active content in sanitized JavaScript. Technical report, Google Inc., June 2008.

K. Minami and N. Borisov. Protecting location privacy against inference attacks. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, WPES '10, pages 123–126, New York, NY, USA, 2010. ACM.

S. Murugesan. Understanding Web 2.0. *IT Professional*, pages 34 –41, 2007.

M. Obitko. Semantic Web. Website, 2007. Tutorial, Available online at `http://obitko.com/tutorials/ontologies-semantic-web/`; last visited on June 12th 2012.

M. Ogrinz. *Mashup Patterns: Designs and Examples for the Modern Enterprise.* Addison-Wesley Professional, 1 edition, 2009. ISBN 9780321579478.

J. O'Madadhain, D. Fisher, and T. Nelson. Java Universal Network/Graph Framework. Website. Available online at `http://jung.sourceforge.net/`; last visited on August 25th 2012.

Omelette. State-of-the-art in the field of Mashup concepts. Technical report, TUC, June 2011.

Open Mashup Alliance. OMA: EMML Documentation. Website. Available online at `http://www.openmashup.org`; last visited on April 5th 2012.

Oracle. Java. Website. Available online at `http://www.oracle.com/technetwork/java/index.html`; last visited on August 14th 2012.

T. O'Reilly. What Is Web 2.0? Website. Available online at `http://oreilly.com/web2/archive/what-is-web-20.html`; last visited on April 3rd 2012.

I. Pahlke, R. Beck, and M. Wolf. Enterprise Mashup Systems as Platform for Situational Applications. *Business Information Systems Engineering*, pages 305–315, 2010.

I. Panagiotopoulos, L. Seremeti, A. Kameas, and V. Zorkadis. PROACT: An Ontology-Based Model of Privacy Policies in Ambient Intelligence Environments. In *Informatics (PCI), 2010 14th Panhellenic Conference on*, pages 124 –129, Sept 2010.

B. Parsia, E. Sirin, and A. Kalyanpur. Debugging OWL ontologies. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 633–640. ACM Press, 2005.

A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *In Proc. of EKAW 2004*, pages 63–81. Springer, 2004.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* 2003.

A. Silva, F. Moreira, and J. Varajão. The Enterprise 2.0 Concept: Challenges on Data and Information Security. In *WSKS (1)*, pages 363–368. Springer, 2010.

Stanford University. Protege. Website. Available online at `http://protege.stanford.edu/`; last visited on August 9th 2012.

D. Stock, F. Wortmann, and J. H. Mayer. Use cases for business metadata: a viewpoint-based approach to structuring and prioritizing business needs. In *Proceedings of the 5th international conference on Global Perspectives on Design Science Research*. Springer-Verlag, 2010.

TheFreeDictionary. Mash-up. Website. Available online at `http://www.thefreedictionary.com/mashup/`; last visited on August 25th 2012.

T. Trojer, B. Fung, and P. Hung. Service-Oriented Architecture for Privacy-Preserving Data Mashup. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 767 –774, 2009.

E. Turban, J. E. Aronson, and T.-P. Liang. *Decision Support Systems and Intelligent Systems (7th Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004. ISBN 0130461067.

W3C. Semantic Web. Website. Technical Report, Available online at `http://www.w3.org/standards/semanticweb/`; last visited on June 12th 2012.

W3C. W3C OWL Web Ontology Language Reference. Website, 2004a. Technical Report, Available online at `http://www.w3.org/TR/owl-ref/`; last visited on May 30th 2012.

W3C. W3C RDF Schema. Website, 2004b. Technical Report, Available online at `http://www.w3.org/TR/rdf-schema/`; last visited on May 30th 2012.

W3C. W3C RDF Primer. Website, 2004c. Technical Report, Available online at `http://www.w3.org/TR/rdf-primer`; last visited on May 30th 2012.

H. J. Wang, X. Fan, J. Howell, and C. Jackson. Protection and communication abstractions for web browsers in MashupOS. *SIGOPS Oper. Syst. Rev.*, 41:1–16, Oct. 2007.

Yahoo! Yahoo! Pipes Website. Website. Available online at `http://pipes.yahoo.com/pipes`; last visited on April 7th 2012.

J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding Mashup Development. *IEEE Internet Computing*, pages 44 –52, 2008.

D. Zhiquan, Y. Nan, C. Bo, and C. Junliang. Data mashup in the internet of things. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, pages 948 –952, 2011.