Institut für Informatik der
Friedrich-Schiller-Universität Jena

Automatic Generation of Semantic Mashups in Web Portals

Diplomarbeit
zur Erlangung des akademischen Grades
Diplom-Wirtschaftsinformatiker

vorgelegt von
Thomas Fischer

betreut von

Prof. Dr. Birgitta König-Ries	Friedrich-Schiller-Universität Jena
Prof. Dr. Martin Welsch	IBM Deutschland Research & Development GmbH
Fedor Bakalov	Friedrich-Schiller-Universität Jena
Andreas Nauerz	IBM Deutschland Research & Development GmbH

30. Oktober 2008

Department of Computer Science at
Friedrich-Schiller-University Jena

Automatic Generation of Semantic Mashups in Web Portals

Diploma Thesis
submitted for the degree of
Diplom-Wirtschaftsinformatiker

submitted by
Thomas Fischer

supervised by

Prof. Dr. Birgitta König-Ries	Friedrich-Schiller-University Jena
Prof. Dr. Martin Welsch	IBM Deutschland Research & Development GmbH
Fedor Bakalov	Friedrich-Schiller-University Jena
Andreas Nauerz	IBM Deutschland Research & Development GmbH

October 30, 2008

Abstract

The Web has become an important source for information, which are created by independent providers. Web portals provide an unified point of access to content, data, services and web applications located throughout the enterprise. However, Web users have often only an insufficient available amount of time, to effectively use the available information resources. This thesis proposes a mashup framework that automatically mashes-up web portal content with related background information. The background information are derived from information web services that are composed by an evolutionary algorithm.

Author's Statement

I hereby certify that I have prepared this diploma thesis independently, and that only those sources, aids and advisors that are duly noted herein have been used and / or consulted.

October 30, 2008 Thomas Fischer

Acknowledgements

I would like to thank all who have supported me in writing this thesis. Foremost, I would like to thank my supervisors Prof. Dr. Birgitta König-Ries, Prof. Dr. Martin Welsch, Fedor Bakalov and Andreas Nauerz, who enabled this thesis and shared with me a lot of their expertise. Furthermore, I would like to thank my supervisors that they were always available for me in order to help me and guide me in my research.
Special thanks go also to my family that have supported me throughout the entire process.

Contents

CHAPTER 1

Introduction

In this thesis, a framework for the automatic generation of mashups is proposed. This chapter describes the context of this proposal. First, the problem area of this thesis is characterized in Section 1.1. Section 1.2 covers the project context. In Section 1.3, the aims and objectives of this thesis are outlined, followed by a description of the organization of this document in Section 1.4.

1.1 Characterization of the Problem Area

The World Wide Web has been designed to provide an open environment of information for humans and machines [BL]. Tim Berners-Lee has created the first web site at the European Organization for Nuclear Research (CERN)[1]. The popularity of the Web has been increasing continuously, but the growing amount of distributed data leads to a dilemma: "*...the more distributed and independently managed that resources on the Web become, the greater is their potential value, but the harder it is to extract value*" [SH05, p.7]. Zilberstein and Lesser [ZL96] state that the increasing number of information sources as well as the different levels of accessibility, reliability, and associated costs of these information sources present a complex planning problem of information gathering. In this context, web users often have the subjective experience of an information overload. This means that they have only an insufficient available amount of time, to effectively use available information resources [Sav07]. A mitigation of this problem could be achieved through human high level filtering of information as well as active withdrawal of information sources [Sav07]. However, this has turned out to be not a final approach, because the human agents may unwittingly neglect important information. This could lead to a decreased quality of the decision making process or task execution.

[1] `http://info.cern.ch`

Moreover, the trade offs between time, quality, and cost as well as the dynamic nature of the internet lead to the insight that the human user is not an appropriate controller of the information gathering process [ZL96], and this motivates the utilization of machine based processing of data and information.
The extraction of useful information from the Web is addressed by research on (Web) Information Retrieval (IR). "*Information retrieval (IR) deals with the representation, storage, organization of, and access to information items*" [BYRN99, p.1]. The present Web search types of information retrieval are mainly algorithmic search engines like Google[2] or Yahoo[3] as well as dictionaries [Lew05, p.24] [BYRN99, p.3]. However, search engines and dictionaries are not able to satisfy all information needs of a human user. Search engines are often queried multiple times by users to gather and aggregate all necessary information. Reasons are summarized in a recent thesis by Lewandowski [Lew05, p.32-37]. First, there is a lack of complex query language support as it is usual in information retrieval. Second, most users are untrained. While the first prevents potential complex investigations on a high level, the second leads to a marginal use of boolean query operators and advanced search forms, which would support more complex queries. Furthermore, while search engines are able to cover large amounts of web pages, they lack sufficient support for a huge mass of structured data sources that are not directly exposed on a web site [RTA07] [Hor07, p.120], the so called deep Web [MJC+07]. Even if a web site provides the required data, it is often not in a form that supports the current user needs. It seems to be clear that there is still a demand on approaches, technologies and tools [WH07] that consider personalized and efficient extraction as well as aggregation of distributed information.
This thesis is motivated by the research on so called mashups that have gained wide popularity in the last few years. In general, they address the problem specific aggregation of data from a wide variety of types of data sources. A famous example of a mashup is Housingmaps[4] by Georg Rademacher. It reuses housing data from Craigslist[5] and places it on a Google Map[6]. The combination of functionality and data sources from different providers leads to a new service that treats a specific problem domain. In fact, neither Google Maps nor the data from Craigslist was initially created for this purpose. Further examples can be found in the mashup directory ProgrammableWeb[7], which currently contains 3456 mashups. The creation of mashups

[2] `http://www.google.de`
[3] `http://www.yahoo.com`
[4] `http://www.housingmaps.com`
[5] `http://www.craigslist.org/`
[6] `http://maps.google.com/`
[7] `http://www.programmableweb.com/`

promises to reduce the information overflow, if these applications are directly personalized to the needs of the human user and address specific tasks or problems in a holistic manner. Even if there were a wide variety of existing mashups, the changing tasks, knowledge, and expertise of web users require a mass of mashups that can not be served by a relatively small set of developers. This motivates the creation of a framework that automatically generates mashups.

The research on mashups has turned out a variety of mashup definitions. Merrill defines mashups as web-applications that draw upon content retrieved from external data sources to create an entirely new innovative service [Mer] (similarly [TSK08], [BN08]). In addition, Nan Zang et al. define mashups as a coalescence of different data sources and application programming interfaces (APIs) into an integrated experience [ZRN08, p.3172]. While the first definition outlines the importance of content aggregation, the second also considers aggregation of functionality. Wong et al. state that mashups combine existing web based content and services to create new applications [WH07, p.1435]. The importance of services is also outlined by Ankolekar et al. that describe mashups as services from different sites that are pulled together in order to experience data in a novel and enhanced way [AKTV07, p. 825]. This is similarly stated by Thor et al. [RTA07] and Lathem et al. [LGS07]. In addition, Frederik De Keukelaere et al. define mashups as applications that mix and merge content coming from different content providers in the browser [KBS+08].

In general, the definitions provide a consensus over the integration and aggregation of different resources in mashups, while differences arise mostly from the types of considered resources. This is in accordance to Hoyer and Fischer [HF08] who provide an overview about different definitions as well as different mashup frameworks. Functionality and presentation integration is also a topic of some definitions, but there is no clear mashup reference model. Unlike many other web technologies, mashups are not standardised. Instead, mashup development has been furthered by thousands of different developers.

Services are seen as important technology that serve the creation of mashups. This thesis distinguishes them, due to their different meanings in different research fields. A service is defined as a "*product of human, organizational, or computational activity meant to satisfy a need, but not constituting an item of goods*" [SH05, p.520]. A web service is a special service that is defined as "*functionality that could be engaged over the Web*" [SH05, p.520].

The current development of mashups can be done manual or semi-automatic with the aid of tools. This thesis proposes a mashup framework for automatic generation of mashups. Therefore, Chapter 3 characterizes and discusses different patterns and lim-

itations of the existing mashup frameworks to describe the need for an automatic generation of mashups.
The next sections provide an overview about different research fields that are important for this thesis. Section 1.1.1 describes the principles of the Web 2.0 as well as the Semantic Web and outlines their relations to mashups. Section 1.1.2 describes the importance of knowledge representation and ontologies for this thesis. In Section 1.1.3, the principles of software agents are characterized. Section 1.1.4 describes the fundamentals of adaptive systems.

1.1.1 Web 2.0 and Semantic Web

Mashups are a principle of the Web 2.0 vision. The Web 2.0 is seen as a platform that utilizes collective intelligence through collaboration, focuses on the importance of data, drives the development process of applications towards the end-user, exerts light weight programming models, and provides rich visualizations to the user [O'R]. Ankolekar et al. state that the Web 2.0 should adopt the rich technical infrastructure of the so called Semantic Web, to exchange information across independent applications [AKTV07].
The Semantic Web, described by Berners-Lee et al. [BLHL01], is a vision that focuses on the extension of the current Web by machine readable and "understandable" meta data. This is important, because the current web sites are designed for human consumption. However, machines need access to structured collections of information and inference rules that they could utilize to perform an automated reasoning. The meta data refers to simple statements like "a car is a special transport vehicle" and should be formalized in a machine processable way. The vocabulary terms of the statements are typically derived from one or more ontologies, which are a conceptualization of the domain of discourse [Gru93]. In general, this leads to a shared understanding of the contents of the statements. Therefore, meta data is important to achieve a Web that enables an easier cooperation between machines and humans [BLHL01]. In fact, since mashups retrieve data from independent data sources, the Semantic Web seems to be important to achieve a uniform understanding and formal description of data across application boundaries and data providers.
The Semantic Web vision adopts important concepts and research topics such as knowledge representation, ontologies as well as agents [BLHL01], which are considered below in more detail.

1.1.2 Knowledge Representation and Ontologies

Knowledge representation and ontologies are important for mashups, because mashups combine data from disparate sources that can be only combined by a shared understanding of the meaning of the data. The semantic description (meaningful to a machine) of Web data has been driven by the research community through the creation of different standards, for instance, the Resource Description Framework (RDF) [KC], the Resource Description Framework Schema RDF(S) [BR] and the Web Ontology Language (OWL) [BvHH$^+$]. These approaches provide a formal way to specify shared vocabularies that can be used in statements about resources. Furthermore, they utilize a syntax based on the Extensible Markup Language (XML) [BPSM$^+$] and thus can be effectively processed by machines. The approaches are explained in more detail in Chapter 4.

The opportunities of combining the potentials of the Web 2.0 and the Semantic Web drive increasingly the interest in development and utilization of ontologies [AKTV07]. The research community has defined the term ontology in different ways. The most prominent definition was given by Gruber [Gru93], who specified an ontology as "*an explicit specification of a conceptualization*". Conceptualizations can be shared among agents. Therefore, the definition has been extended [CFLGP03, p.43] by Borst [Bor97, p.12]: "*Ontologies are defined as a formal specification of a shared conceptualization*". Later, Stuber et al. explained this definition in detail [RS98, p.25][8]. The research community has turned out further definitions of the term ontology (e.g. Guarino [Gua95], Uschold and Jasper[UJ99], Bernaras et al. [AB96]). This thesis refers to the definition of Borst[Bor97, p.12], which has been mentioned above.

It is important to note that an ontology distinguishes a priori the entities of the world (material or immaterial objects, events, quantities etc.) which are modeled through meta elements such as concepts and properties. Agents have to commit to this shared conceptualization ("view" of the world) to achieve a shared model of the world.

Conceptualizations can be denoted as taxonomy or ontology. Each has a different conceptualization quality. A taxonomy provides simply a hierarchy of concepts. This means that it typically provides relationships like generalization and specialization.

[8] "*A 'conceptualisation' refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. 'Explicit' means that the type of concepts used, and the constraints on their use are explicitly defined. For example, in medical domains, the concepts are diseases and symptoms, the relations between them are causal and a constraint is that a disease cannot cause itself. 'Formal' refers to the fact that the ontology should be machine readable, which excludes natural language. 'Shared' reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group.*".[RS98, p.25]

This is also part of an ontology, which itself can be more expressive through the definition of additional specifications, such as additional relationships between concepts or restrictions on properties.
In accordance to the above explanations, a shared conceptualization seems to be important to achieve a shared understanding of the world within a mashup application. Therefore, they are described in detail in Chapter 4.

1.1.3 Agents

Tim Berners-Lee figured out the importance of agents in the Semantic Web. Such agents collect content from different data sources, process the information and share the results with other ones [BLHL01]. In accordance to Russel and Norvig an agent is defined as "*anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through actuators*" [RN03, p.32]. Thus, an agent can be a human as well as a software agent. However, if not stated otherwise, this thesis refers to agents as software agents.
In the context of this thesis, a mashup agent would receive percepts from a sensor that monitors user interests, expertise as well as other environmental data to adapt the information gathering process and to provide personalized information and functionality. For instance, a changing user interest could result in the invocation of different web services as well as a change of the user interface of the mashup application. The outside

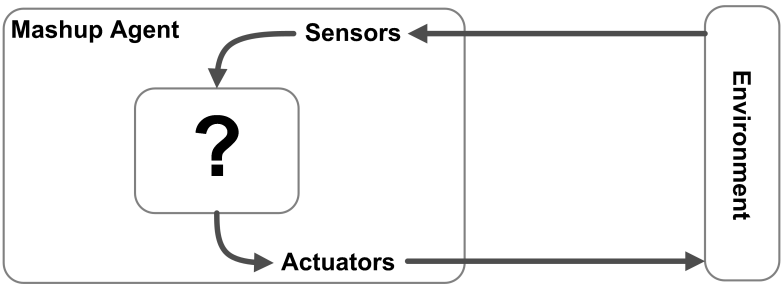

Figure 1.1: Mashup Agents interact with environments through sensors and actuators (adapted from [RN03, p.33])

visible behaviour of an agent is described by an agent function, which is implemented by an agent program[RN03, p.32]. The agent program processes the inputs of the sensors and returns the results.
Important to any agent is the concept of rationality. "*A rational agent is one that does the right thing*" [RN03, p.34]. This means not that the agent is omniscient or perfect. Rather, it means that the agent maximizes the expected performance of its actions. In

contrast, perfection would mean to maximize the actual performance [RN03, p.37]. This is clarified in the following simple example. The question is whether or not to take an umbrella along today, if there is an 90% raining probability reported by the wheather station. The software agent recommends to take the umbrella, which seems to be straightforward. However, even if no rain falls, the decision of the agent is not irrational. Instead, a perfect agent would have known that it will not rain today and thus would not have recommended an umbrella. Unfortunately, it is not possible to create such a perfect agent.

According to Russel and Norvig, a rational agent can be defined as follows:"*For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has*" [RN03, p.36]. This means that a mashup agent displays those information and functionality that is expected to maximize the support of the human agent and that is expected to have the highest utility for the human agent in dependence on the modeled formal goals of the software agent. Agents often operate in an informed way and therefore evaluate its present state by an evaluation function, which considers the preferences of the human agent. Such decision theoretic aspects are considered in Chapter 4.

The complexity of the agent program depends on the environment the agent is acting in. For instance, the agent of the umbrella example has to adhere to the stochastic nature. The environment can be specified along different dimensions (in accordance with Russel and Norvig [RN03, p.41-42]). Mashup agents are acting in a partially observable, stochastic, dynamic and distributed environment. The environment is based on the resources of the entire available internet as well as the unity of users which interact with the agent. The state of the internet as well as the user interests and preferences can be only partially observed. In addition, the behaviour is not determinable in advance and thus stochastic. Furthermore, the environment changes dynamically during the execution of the software agent program (e.g. web services become available or unavailable). In addition, the data sources as well as functionality that should be aggregated are distributed over the internet. This complexity of the environment has to be taken into consideration in the development of a mashup framework, because it influences the approaches and implementation of the agent program.

Agents can act in single-agent or multi-agent environments. In multi-agent environments they may interact in competitive or cooperative manner. Due to this, mashups could in principle proactively reuse mashed-up data from other mashup agents to support the users cooperatively.

Proactive behavior of agents is achieved by the definition of goals that have to be

achieved by an agent through the execution of a corresponding plan [vRDW08, p.713]. The research mainly distinguishes between declarative and procedural goals [vRDW08, p.713]. Declarative goals represent an intended state, which would correspond to the goals of a mashup agent that gathers information to achieve some defined information state. In contrast, procedural goals represent the aim of execution of some desired action. Such an action could be buying a book or selling an item. The research community has developed several approaches for agent based planning, which are considered in Chapter 4.
The previous explanations on agents have outlined some important concepts. They are clarified in subsequent chapters of this thesis that consider special aspects of agents in the context of mashups.

1.1.4 Adaptive Hypermedia

The information needs and tasks of users are different and change dynamically. Hence, personalization is important to mashups. This is currently not reflected in most mashup definitions, because mashups per se are related to a specific domain of discourse. Nevertheless, this thesis considers mashups as parts of complex adaptive systems that enable an automated personalization of the contents and functionality that are provided to the user.
One important feature of any adaptive system is the user model, which is important to achieve the adaption effect among the different users [BKN07]. Adaption in Web-based systems includes adaptive navigation, adaptive search, adaptive recommendation as well as adaptive contents and presentation [BKN07, p.409]. In fact, adaptive contents, functionality, and presentation, which address the preferences of a human user, are important to mashups. In this context, the user model could contain information about the knowledge or expertise of a user, his long term and short term interests, goals, and tasks as well as data about the current environment or background of the user [BKN07, p.1-12]. This promises to allow the adaptive mashup to model the information gathering process as well as provided functionality differently among the users to reduce the information overflow.

1.2 Project Context

This thesis is part of the Minerva Portals[9] project, which is a joint effort of the Institute of Computer Science[10] at the University of Jena[11] and the IBM Deutschland Research & Development GmbH[12] in Boeblingen. The goal of the project is to create context-adaptive web portals, which are more intuitive, dynamic and flexible than the current web portals.
An enterprise portal provides an unified point of access to content, data, services and web applications located throughout the enterprise. In addition, it enables collaboration between human users as well as support for business processes. Unfortunately, the mass of enterprise wide information makes it increasingly difficult to find information and pages quickly. The research on context-adaptive portals tries to overcome this problem.
The project mainly focusses on context-awareness and adaptivity. Context-awareness means that the portal can be efficiently adjusted to fit the user's current situation. Adaptivity means that the portal learns the behavior of the user and his information needs to provide a fast accessible personalized subset of information. This shall reduce the information overflow and therefore increase the efficiency in information gathering.
Adaptive mashups can be one part of such a context-adaptive portal. They could support the user with relevant and related background information as well as functionality while he is reading business or scientific texts. This promises to reduce the time for search on relevant background information. Furthermore, the decision quality may be increases through a better understanding and evaluation of specialised texts. In addition, the adaptive mashup could support recurrent tasks (e.g. travel planning) by integrating relevant functionality from disparate providers.

[9] http://www.minerva-portals.de/
[10] http://www.informatik.uni-jena.de/
[11] http://www.uni-jena.de/
[12] http://www-05.ibm.com/de/entwicklung/

1.3 Aims and Objectives

This thesis proposes a mashup framework architecture that fully automatically creates personalized mashups based on a variety of different information web services. The information web services are automatically selected and composed to retrieve background information. The resulting mashup provides background information in relation to a specific provided content of a web portal. Thus, the focus lies on automatic content integration from different independent web service providers, which could be located throughout the internet or the enterprise.
It was aimed to create a prototype that automatically creates semantic mashups, which are adapted to tasks, interests, expertise and information needs of users.
This thesis defines semantic mashups as applications that are created by a software agent that retrieves and merges data, functionality, or presentation from distributed, disparate, and independent providers and transforms it into a representation meaningful to the agent.
Furthermore, it was aimed to investigate present patterns of mashups and mashup frameworks as well as economical incentives for their creation. This was important to outline the limitations of the present approaches as well as the advantages of an automatic generation of semantic mashups.
Most of the present mashups are based on information web services that were selected by human agents. Furthermore, the aggregation of data from different sources is mostly achieved through a manual wired data flow. However, the increasing number of available web services requires an efficient automatic selection and composition of such ones, which is not addressed by the current mashup development frameworks. It was aimed to overcome this problem by the utilization of semantic web service descriptions for an agent based selection and composition of information web services.
It was intended to provide an approach that completely describes the available information by formal statements, which refer to terms of a shared vocabulary, such that the mashup agent is able to automatically merge the information.
In the current development approaches, the user or developer of a mashup directly incorporates his preferences, interests, and tasks into the mashup. However, interests, experience and tasks are changing over time. Therefore, the mashups should adapt automatically to avoid unnecessary manual reengineering. This thesis states that personalization through automatic adaptation is one of the key elements of a mashup framework to avoid information overflow. This promises to be important to support the changing tasks and information needs of human agents appropriately. Therefore, the system aims to plan the information gathering based on the present goals of the user to

achieve personalized compositions of information web services.

1.4 Document Organization

This thesis is organized as follows. Chapter 2 describes requirements for this proposal. Chapter 3 considers related work on mashups and mashup frameworks and investigates their patterns and characteristics. The limitations of these approaches will lead to the insight that the stated requirements could not be fulfilled by the existing systems. Therefore, this thesis proposes the automatic generation of mashups. Chapter 4 describes in detail the theoretical foundations of the proposed approach. In Chapter 5, the architecture of the systems is described, which is based on the stated requirements as well as the theoretical foundations.
The key part of this thesis is an evolutionary algorithm that creates a plan for the gathering of background information from available web services, which can be invoked to derive a comprehensive set of data. The algorithm addresses the needs for a scalable and multi-objective preference based planning of web service composition and is described in detail in Chapter 6.
Chapter 7 describes the implementation of the proposal. Chapter 8 investigates the feasibility and scalability of the proposed framework in a quantitative evaluation before Chapter 9 concludes the thesis with a summary and an outlook to possible extensions.

CHAPTER 2

Requirements Analysis

This chapter describes and formalizes the requirements for an automatic generation of mashups. Section 2.1 gives an overview about the considered domain of this thesis. Section 2.2 describes the general requirements on the mashup framework. It investigates different functional prerequisites and describes use cases and examples in the proposed domain. Furthermore, it describes also common non-functional requirements that should be considered. In Section 2.3, the high level requirements are formalized in a set of related framework requirements, which are important to achieve an appropriate architecture for the automatic generation of mashups. Section 2.4 concludes this chapter with a short summary.

2.1 Domain Description

This thesis considers the domain of the financial sector. The financial sector represents an environment which requires to have relevant information timely as a foundation of decision making. Financial information are increasingly available over the internet. However, it is often difficult to utilize all available information in the decision making, because they are distributed over the Internet as well as the intranet of the financial enterprises. The gathering, merging, and visualization of all required data is therefore difficult to achieve. Especially in a competitive market like banking, poor information could result in poor decisions that affect the business performance significantly [KMP98].
The above stated problem relates to the problem of information overload that has been described in Chapter 1. Furthermore, the required aggregation and visualization of data from disparate providers is the central objective of mashups. Therefore, the financial domain should be suitable to show the automatic aggregation and provision of

background information through the automatic generation of mashups.
The proposed approach will be also transferable to other domains, which is explained as follows. In general, decision making requires the incorporation of background information to increase the quality of the decision making process. This is important for strategic and operational decision making. Especially for operational decisions, the available amount of time for the gathering and aggregation is limited. Therefore, the framework could in general provide background information for the support of operational decisions. For instance in the health-care domain, the framework can support the mash-up of medical patient data with background information to different pharmaceuticals, treatment plans etc.. Furthermore, the framework could also support the mash-up of domain specific technical and scientific texts to support the understanding of complex contents.
This section explained the domain of the proposal. The next section describes the general requirements on an architecture for automatic generation of mashups.

2.2 General Requirements

This section describes general requirements that should be considered in the architecture of the system. The first part describes different business use cases of the system. Based on these use cases, the common functional requirements of the framework are stated in the second part. The third part considers non-functional requirements and states important information criteria that have to be considered to create mashups that are compliant with local regulations, laws and intra-organizational guidelines. The fourth part of this section investigates requirements that originate from the research on Data Warehouses and Online Analytical Processing. Similar to mashups, these systems integrate and aggregate information of enterprises to support decisions. The requirements on such systems can therefore be also important for the development of mashups.

2.2.1 Business Use Case

This thesis considers companies of the financial sector to create a business use case for the mashup framework prototype. An existing approach that utilizes Semantic Web technologies in the financial domain could be found in Castells et al. [CFL+04]. The financial sector represents an environment which requires to have relevant information timely as a foundation of decision making. It is often difficult to utilize all available in-

formation in an investment decision, because the information are distributed over the Internet. The gathering, merging and visualization of all required data is therefore difficult to achieve. Especially in a competitive market like banking, poor information could result in poor decisions that affect the business performance significantly [KMP98]. It is important to note that the proposed approach should not replace the existing information infrastructure, but it should help to drive the information gathering process for background information more efficiently. Therefore, the resulting mashups do not contain any investment recommendations, as it is done in financial decision-support agent systems (see [TG02], [DSS08]).
Investment companies must anticipate market trends and evolutions to make competitive investment decisions. These decisions are based on detailed research activities of the investment company. Amongst others the research scope is related to different companies, markets or technologies based on various data sources, such as financial ratios, expert opinions, news releases etc.. These resources are increasingly available throughout the Internet and should be therefore utilized in automatic generated mashups.
Figure 2.1 reflects the business use cases of the mashup framework prototype. It contains two actors (a portfolio manager and a research analyst) that have corresponding tasks such as portfolio management or information research. The system should augment existing contents by background information to enable the actors to perform their tasks more efficiently. Tasks such as traveling are common to both actors, whilst investment research and portfolio management are only related to one actor.

2.2.1.1 Use Case: Portfolio Management

This thesis will not explain in detail strategies and concepts of investment theory, because it would exceed its capacity. Moreover, the focus relies on information gathering from distributed data sources with a special use case in the financial domain. The interested reader could refer to Radcliffe [Rad97], Fischer and Jordan [FJ95] or Fabozzi [Fab98] for an overview about investment management.
A portfolio investment process contains generally the plan, the implementation of the plan and the evaluation of the plan. According to Radcliffe, the general planning process of investment decisions could be described in the following way [Rad97].

Planning

1. Investor Conditions
2. Market Conditions
3. Investment / Speculative Policies

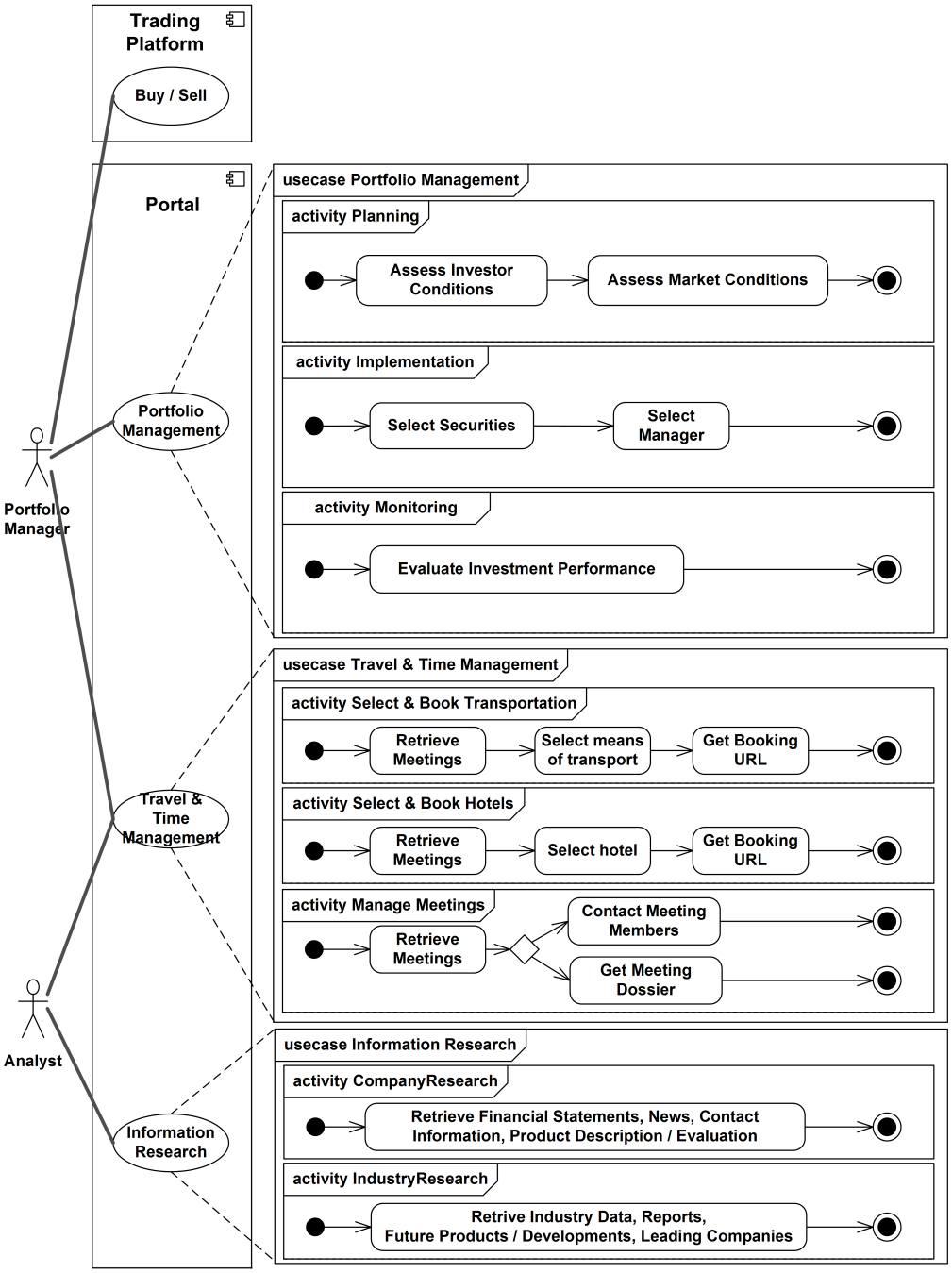

Figure 2.1: Business Use Case

4. Statement of Investment Policy

5. Strategic Asset Allocation

Implementation

1. Internal-External Management

2. Security-Manager Selection

3. Tactical Asset Allocation

Monitoring

1. Evaluate Statement of Investment Policy

2. Evaluate Investment Performance

The mashup framework could support different phases of the portfolio investment process. In the planning stage, the decision maker aims to achieve an clearly defined Strategic Asset Allocation (SAA) that represents the optimal combination and weighting of various asset classes [Rad97, p.742]. The human agent has to incorporate information about the investors financial situation as well as the present capital market conditions to achieve an optimal SAA. The typical required data includes a variety of business ratios, news, trends and time series.
Mashups of WebSphere Portal can serve as agents that automatically retrieve and merge corresponding data to avoid unnecessary search in the Web or intranet. For instance, a portfolio manager reads an assessment about a specific market (e.g. real-estate) on a web portal page. Assessments are often created by analysts. The mashup augments specific products and terms of the text with background data to enable the portfolio manager to quickly read and understand the content. This means that risk measures, performance indicators as well as charts are displayed on the mashup in relation to the provided content and the managed portfolios. Based on the expertise of the decision maker in the specific market, the system filters out irrelevant information to avoid an information overflow.
The data should be based on real-time information such as time series, news, etc., if this is important for the specific decision. The resulting analytic mashup enables the portfolio manager to discover impacts and cross-relations in the market more efficiently.
In addition, the mashup can also contain references to other portfolio managers or experts of this field to enable collaboration.

2.2.1.2 Use Case: Travel Management

Employees often work at different clients during the week, and thus spend much time for travel planning. This means that they have to search for hotels near the client, railroad, or airplane connections in accordance to the meetings and personal preferences. Furthermore, meetings could change in time and location. Therefore, people often spend much time to refine the plans.
Mashups can support the user with mappings that incorporate hotel, travel and client information. Furthermore, the mashups can automatically examine the relevant meetings from various sources (e.g. calendar) and suggest possible hotels near the meeting locations as well as trips by plane, car or railroad. Due to the automatic generation of the mashup, the presented mashup is adapted if the situation changes (e.g meetings are declined)[1].

2.2.1.3 Use Case: Information Research

Information research is very important to investment banking. The daily political, technological and company news have to be analyzed to assess the influence on controlled securities and industries. Mashups can automatically extract relevant news for a given watch list or portfolio. Furthermore, the news can be mashed-up with detailed information about technologies, products etc. concerning the expertise of the specific user.

2.2.2 Functional Requirements

The use cases clarify the following functional requirements for the framework. In general, the given contents of a web portal have to be augmented with background data $[Req.\ F1]$.
Information overflow has to be avoided. Therefore, the system has to consider the interests, expertise and tasks of users $[Req.\ F2]$. This means that the mashups are different among the users and therefore the systems has to create the mashups in an adaptive way.
Financial data is increasingly available through information web services (e.g. stock market time series and business news). Furthermore, also intra-organizational infor-

[1] It seems to be important that the user should also be able to look at the mashups on mobile phones to avoid unnecessary and time consuming starts of a laptop. This is not considered by the this thesis. However, it seems to be promising to generate the mashup presentation for different technical environments, while retaining on the same semantic data.

mation web services should be considered. In addition, web service offer the provision of real time information, which are very important for financial decisions. As explained in Chapter 1, web services are seen as important building blocks of mashups. Therefore, the system should incorporate the background information from available information web services [*Req. F*3].
The requirement on an adaptive behaviour of the system leads to insight that the underlying set of available information web services has to be selected dynamically [*Req. F*4], because the predefinition of the selection in dependence to the different contents of the portal as well as interests, expertise and tasks of users is not suitable. In addition, the systems has to dynamically plan the composition of the web services, because data from one web service has often to be used as an input of another web service. For instance, one web service retrieves the stock symbols of a company, which are required in a subsequent web service that returns the time series for this symbol. This planning process has to be done also in an automated manner [*Req.F*5] to let the users concentrate on the real problem domain. Furthermore, the system has to consider the invocation of the web services [*Req. F*6].
In Chapter 1 the importance of a shared understanding of the meaning of the data by the system has been figured out. Therefore, the data of the mashup (background data) has to be represented in a formal way to achieve automatic processing of the information [*Req. F*7]. Moreover, as stated in Chapter 1, machines need access to structured collections of information and inference rules that they could utilize to perform an automated reasoning and processing. Therefore, the system has to utilize semantic technologies to enable an dynamic and automatic processing of the data [*Req. F*7].

2.2.3 Non-Functional Requirements

Non-functional requirements are based on service quality oriented aspects as well as intra-organizational guidelines. The "Control Objectives for Information and Related Technology" framework (CobiT)[2] relates company objectives to the information technology architecture. It is important to adhere, because it is used by many of the top companies all over the world. The basic principle of CobiT is that "*IT resources are managed by IT processes to achieve IT goals that respond to the business requirements*" [ITG08] (Figure 2.2). CobiT defines workflows that help to ensure the processing of information, the handling of information resources and the execution of business services in accordance to the business objectives and regulatory compliance demands. This is nec-

[2] `http://www.isaca.org/cobit`

essary to setup an effective IT risk management. The CobiT framework was developed by the "Information Systems Audit and Control Association" (ISACA)[3] and is currently maintained by the IT Governance Institute (ITGI). It is integrated with other control frameworks for enterprise governance and risk management (e.g COSO) to manage the "*associated risks, such as increasing regulatory compliance and critical dependence of many business processes on information technology (IT)*" [ITG08] as well as decision processes.
Mashup frameworks have to adhere to the controlling aspects of information technology. In fact, controlling is important to understand and manage risks and to exploit benefits. Since there is a trend towards end-user development, the traditional IT management will be affected, because there is a mass of independent created mashup applications that have to be compliant with regulatory standards, laws and intra-organizational guidelines. Furthermore, these applications are used immediately after their creation. This complicates the controlling in a way that is not part of traditional applications that are audited, documented and secured in advance to their productive usage. In fact, if mashups become a significant part of the enterprise wide business processes, the security and blackout risks of theses systems have to be controlled $[Req.\ NF6]$. Otherwise, in an uncontrolled environment the neglect of risk assessment could lead to unwanted impacts that affect the performance or strategical position of the enterprise. Therefore, it is very important that mashups are based on a sounding framework that allows also the evaluation of the decisions and the quality of the decision related data.
CobiT defines effectiveness[4], efficiency[5], confidentially[6], integrity[7], availability[8], compliance[9] and reliability[10] as important information criteria[ITG08]. As a consequence, the enterprise mashup framework architecture should also consider these criteria and requirements to control decisions based on enterprise mashups while considering the overall business $[Req.\ NF1]$.
For instance in the financial domain, the management should be able to consider the

[3] `http://www.isaca.org`
[4] "*Effectiveness deals with information being relevant and pertinent to the business process as well as being delivered in a timely, correct, consistent and usable manner*" [ITG08].
[5] "*Efficiency concerns the provision of information through the optimal (most productive and economical) use of resources*" [ITG08].
[6] "*Confidentiality concerns the protection of sensitive information from unauthorized disclosure*" [ITG08].
[7] "*Integrity relates to the accuracy and completeness of information as well as to its validity in accordance with business values and expectations*" [ITG08].
[8] "*Availability relates to information being available when required by the business process now and in the future. It also concerns the safeguarding of necessary resources and associated capabilities*" [ITG08].
[9] "*Compliance deals with complying with the laws, regulations and contractual arrangements to which the business process is subject, i.e., externally imposed business criteria as well as internal policies*" [ITG08].
[10] "*Reliability relates to the provision of appropriate information for management to operate the entity and exercise its fiduciary and governance responsibilities. "* [ITG08].

reliability of mashup data in an investment decision, to evaluate the quality of decisions. This is also in accordance with Lippner [Lip07, p.142], who stated that the online market research in investment decisions has no laws, no quality control, no quality regulations as well as no quality standards.
Web services are a technology that should enable the creation of "contracts" between service provider and service requester. Based on these, the compliance could be controlled by the responsible management on the level of the web services. In general, this assumes that utilization of compliant web services leads to compliant mashup applications. Thus, the architecture should consider multiple information criteria for web services and compositions of web services for mashups [*Req*. *NF*1].

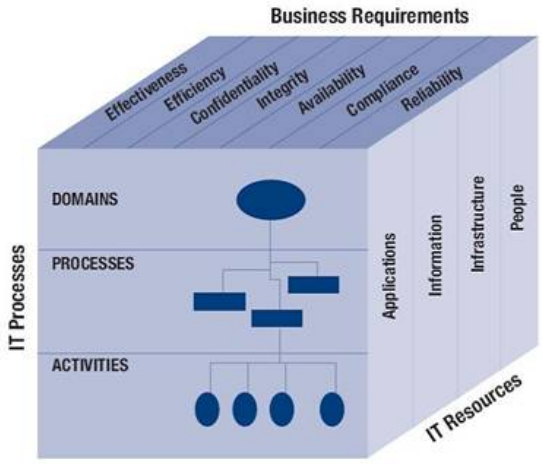

Figure 2.2: The CobiT Cube

As explained in the previous chapter, mashups remix and merge data and functionality retrieved from disparate providers. In the context of enterprises, the integration and aggregation of contents has been already addressed by the research on Data Warehouses. The next part investigates, if requirements on theses systems could be transfered to the domain of mashups.

2.2.4 Mashups versus Data Warehouses and Online Analytical Processing

In the context of enterprises the relevance of information gathering from different enterprise data sources is addressed by the research on Data Warehouses (DW) and Online Analytical Processing (OLAP). Devlin and Murphy introduced the concept of a data warehouse in 1988 [DM88]. A data warehouse integrates and aggregates data from different sources of the enterprise to support tactical and strategic decisions [Lus01, p.131]. In fact, this is indeed very similar to some definitions of mashups that has been

described in Chapter 1. Apatar[11] is a prominent provider of a combined data warehouse mashup solution.
A data warehouse increases the provision of information for decision makers and shows off relevant alternatives through the usage of WHAT-IF analyses, scenarios, simulations, time series and business ratios [MB00, p.37]. A key characteristic of a DW is the query intensity of the processing, instead of transactional intensity in traditional databases. In contrast to DWs, production databases are operative and provide complete details of all transactions. They are without redundancy, change intensive and have a complex data model [Lus01, p.131].

Often central DWs are combined with Data Marts. The DW is responsible to gather

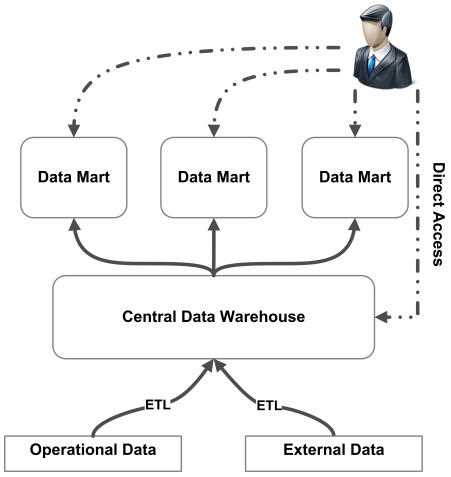

Figure 2.3: Data warehouses and Data Marts

and integrate the data. The aim of the Data Mart is to query and analyse a part of the whole data [Lus01, p.138]. For instance, a Data Mart could be related to a specific branch of the company. Data Marts achieve higher response times for a special subset of the DW data [MB00, p.411].
The extract-to-load (ETL) process of the data is very complex and thus DWs need significant upfront efforts to achieve a data integration of different data sources [FHM05] [RTA07]. As stated above, mashups (and may be also DWs) should leverage the technical infrastructure of the Semantic Web, which promises to make the data retrieval and integration more flexible than current approaches.
While data warehouses are more concerned about the back-end (e.g. handling and storing of data, Online Analytical Processing (OLAP) is more front-end oriented and addresses the human data access [MB00, p.403]. OLAP provides dynamic and multidimensional analyses on historical and consolidated data, to enable fast and interactive

[11] `http://www.apatar.com`

access to consistent and decision relevant data [MB00, p.402] [Lus01, p.130-131]. Thus, OLAP systems focus on the analysis of the data and are independent from the transaction of the business process as it is the case in Online Transactional Processing Systems (OLTP).
The previous explanations show that DWs, OLAP and mashups have to some extent similar objectives. Mashups and DWs are created to aggregate data from different sources to support users with relevant information in the decision making process. Of course, mashups has emerged from the consumer side, whereas data warehouses are mostly part of the enterprise. However, enterprise mashups penetrate the enterprise market on a level that enables "all" employees of a company to efficiently merge and aggregate problem relevant data. Thus, mashups are in competition to the data warehouses. DWs define an extract-to-load process that specifies from which data sources the data is gathered and how the data should be integrated. The present mashup frameworks, which are considered in detail in subsequent sections, also specify such a process, which is often supported by a graphical user interface that allows an wiring of different data sources.
However, the amount of aggregated data of DWs is much higher than in mashups, because mashups only combine problem relevant and problem specific data, whereas DWs tend to integrate huge sets of enterprise data over a long time. This is a important difference of both approaches.
Mashups integrate different APIs to provide different functionality and views for the data, for instance, in the form of mappings. However, if mashups should be part of the enterprise wide decision processes, the means for analyzing data should manifold. Since, OLAP provides fast access to problem specific data and enables multidimensional analyses, it is interesting to investigate requirements on OLAP systems, which could be a guideline for additional requirements for decision oriented enterprise mashups.
In the following, this thesis investigates some requirements on OLAP system, based on a investigation of Muksch and Behme [MB00, p.404-408]. An OLAP system should provide [MB00, p.404-408]:

- multidimensional perspectives.
- transparency (user has not to concern about technical things).
- access on variety of different data sources.
- stable response times and report views
- a client/server architecture.

- dynamic maintenance of sparse data matrices.
- multiuser capabilities.
- cross dimensional operations over dimensions.
- intuitive data manipulation such as slice and dice, drill down, roll up, filtering as well as absolute and relative visualization.
- flexible reports.
- an possible infinite number of dimensions and aggregation steps of data.

Multidimensionality means that the system provides different business ratios (such as sales, costs, earnings etc.) related to different business relevant dimensions (e.g. debitor, article, region, company location etc.). This promise is also important to mashups that support decision processes.
Transparency of the system seems to be also important to mashups, because human agents should be able to concentrate on their real problem.
Access to a variety of different data sources and aggregation of data is a key characteristic of OLAP systems as well as mashups and thus has to be considered.
Important to any OLAP system is that the response times are low to achieve an appropriate human machine dialog that not interrupts the flow of thoughts of the human. This seems to be also relevant for mashups.
The requirement of a client/server architecture is naturally addressed by many mashups, which are often exposed on a web site and thus support also multiuser capabilities.
Cross dimensional operations as well as slice and dice, drill down, roll up and filtering of data are key elements of OLAP systems. This should be adhered, to provide as much analysis flexibility as possible for the user. The reason is straightforward, aggregated data seems to be not valuable, if it could not be manipulated and transformed for specific decision purposes supported by a flexible reporting.

The above explanations outlined that DWs, OLAP and mashups have to some extent similar objectives. The ETL process of DWs needs significant upfront efforts. Therefore, mashup frameworks should provide simple and efficient means for the selection of data sources as well as the aggregation of the data [*Req*. *F*4][*Req*. *F*5]. Furthermore, many requirements on OLAP systems seem to be also important to mashups. Table 2.1 outlines important requirements for mashups, which result from the above investigation.

Table 2.1: Additional Requirements

Requirement	Description
[*Req. F9*]	multidimensional perspectives
[*Req. F10*]	rich analyse capabilities and intuitive data manipulation
[*Req. NF2*]	transparency to the user (no technical details)
[*Req. NF4*]	stable response times and report views

2.3 Framework Requirements

A framework for automatic generation of mashups has to consider versatile requirements, to achieve an appropriate system architecture [VAC+05, p.32,p.92]. In the domain of information technology the software architecture of a system could be defined as a description of the **system structure** as well as its software components and substructures. On the other hand, architecture as a discipline considers tasks and decisions that conceptualize and realize the software architecture [VAC+05, p.46-47]. Vogel et al. distinguishes between **functional** and **non-functional** dimensions of requirements [VAC+05, p.94-98] [FB98]. This section describes the architectural requirements of the mashup framework. They are derived from the general requirements of the previous section.

2.3.1 Functional Requirements of the Framework

The following functional requirements should be considered in the mashup framework. The system should:

- [*Req. F1*] be able to augment data of existing portlets with user relevant background information.
- [*Req. F2*] incorporate user interests, tasks and experience to achieve an adaption effect among the mashups.
- [*Req. F3*] support information web services.
- [*Req. F4*] should select information web services automatically, based on the user needs.
- [*Req. F5*] should plan the information gathering without user interaction.
- [*Req. F6*] should invoke the plan dynamically to gather the background information.

- [*Req*. *F*7] use semantic technologies to enable processing and inference by the software agent.
- [*Req*. *F*8] support multidimensional perspectives
- [*Req*. *F*9] support rich analyse capabilities and intuitive data manipulation

Chapter 1 explained that a key part of any adaptive system is the user model. Requirement [*Req*. *F*2] describes the need for an user sensitive adaption among the mashups. However, the creation and update of a fully functional user model is not part of this thesis. Therefore, the thesis will only specify different tasks according to the business use case. The different tasks are responsible for the adaption effect. For a fully functional system, the current active task has to be specified based on the information from the user model.

2.3.2 Non-functional Requirements of the Framework

As explained in the previous section, the utilization of web services and compositions of web services has to consider different information criteria [*Req*. *NF*1]. The following list specifies the criteria that should be considered. This list of course not indisputable, but it should be appropriate to show the incorporation of multiple user preferences into web service compositions. This thesis concentrates on QoS aspects of web service compositions [SH05].
The considered criteria for compositions of web services are [*Req*. *NF*1]:

- *COMPLETENESS*: Completeness of a composition of web services denotes to which percentage the effects of the web services support the specified goal. In this context, the goal is a formalization of the required background information. It is aimed to maximize this objective to achieve maximum support for the user (maximization goal).
- *CORRECTNESS*: Correctness of a composition of web services is related to the capability to be invoked in a correct ordering, such that preconditions of all web services of the plan are fulfilled. The correctness is measured by the percentage of web services that could be invoked correctly. The percentage of correct invocable services should be maximized (maximization goal)
- *LENGTH*: The length of the web service composition is measured by the count of web services of a specific composition. It is aimed to have only necessary web services in the composition. Thus, the plan length should be minimized (minimization goal).

- *AVAILABILITY* : Availability denotes if the web services of the composition are ready for immediate usage. For one web service the availability is measured by its probability of immediate readiness. The availability of a web service composition is measured by the total probability $prob(X_1 = 1, \ldots, X_n = 1)$, whereby X_i denotes a web service of the composition that can be 0 or 1 (unavailable or available).
- *ACCESSIBILITY*: Accessibility denotes capability to serve a specific request. It is measured by the success rate derived from previous requests. The composition success rate is processes by the total success rate $prob(X_1 = 1, \ldots, X_n = 1)$, whereby X_i could be 0 or 1 (no success or success).
- *INTEGRITY*: Integrity is a quality aspect that denotes the correctness of the interaction in respect to the source. The integrity is measured on a scale of 0 until 5. A value of five represents a high integrity value. The composition integrity is measured as the minimum integrity of the web services of the composition.
- *PERFORMANCE*: Performance is measured by the throughput and latency. Throughput represents the number of Web service requests served at a given time period. The composition performance is measured as the average number of requests that could be served.
- *RELIABILITY*: Reliability represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service. The number of failures should be minimized. The composition reliability is measured as the maximum number of failures of all web services of the composition, because web services with a high failure rate has to be considered.
- *COMPLIANCE*: Compliance denotes how a web service conforms with local regulations, laws, and intra-enterprise standards. The compliance is measured by a compliance value $[0, \ldots, 5]$. The higher the value, the better is the compliance. The composition compliance denotes the minimum compliance value.
- *SECURITY*: The security is measured by a security value $[0, \ldots, 5]$. The higher the security value, the better the security. The composition security is measured as the minimum security value of all web services. This seems to be suitable, because one total insecure web service could compromise the whole system.
- *PRICE*: The price of a web service composition is naturally an important objective. It is processed as the total amount of all web services and is measured in Euro and per request. There could be also, basic agreements for web services that

are not deducted per request. In this case the costs per request could be estimated based on the total count of requests in a specific time period.

2.3.2.1 Runtime Requirements

Runtime requirements define qualities of the system while it is used, which are very important for its acceptance by users[VAC+05, p.102] and fundamental for interactive mashups[RTA07].

- [*Req.NF*2] **Transparency**: The system should be transparent (user has not to concern about technical things).
- [*Req.NF*3] **Users**: The system should provide multiuser capabilities.
- [*Req.NF*4] **Performance Behaviour**: The creation of the mashup has to be performed in an appropriate time regarding the user interaction. This means that the system is scalable for also huge sets of available web services. However, there is of course an trade-off between the amount of processed data and requested data sources as well as the time of mashup generation. Due to this, the time seems to be appropriate as long there is an significant time reduction for the information gathering task of the human agent without interruption of the flow of thoughts of the user.
- [*Req.NF*5] **Usability**: The user should be able to configure the modules by an graphical user interface, to affect the behavior of the system.
- [*Req.NF*6] **Security**: The security of a mashup framework is important especially to enterprise mashups, because the functionality and content could stem from various untrusted providers [KBS+08]. The main problem is that if it appears to a browser that content or code from different trust domains comes from the same origin, this can lead to uncontrolled interaction[KBS+08]. An unwanted behaviour of code and/or sniffing of the data can effect in the violation of local regulations, corporate governance and loss of company performance as well as strategic opportunities. Therefore, security has to be adhered by the mashup framework architecture as possible. Since the framework should be developed for a web portal, it could utilize its security features.

2.3.2.2 Development Requirements

The development requirements are common and depend in this case on the project context. The application should be developed as a web application that can be run in IBM WebSphere Portal[12]. Furthermore, the application components should be reusable, maintainable and extensible, which is important for later project phases. In addition to this thesis, the usage and program code have to be documented.

- [*Req.NF7*] **Platform Independence**: The mashup framework should be developed as web application that should be accessible throughout the WebSphere portal.
- [*Req.NF8*] **Reuseability**: The application should be structured in a way that allows an reuse of different modules without much effort.
- [*Req.NF9*] **Maintainability**: This means that changes of the web services as well as of the knowledge base should be possible without redeployment of the framework application.
- [*Req.NF10*] **Extensibility**: Extensibility is an important non-functional framework requirement [VAC+05, p.101]. Due to this, the software architecture should be structured loosely coupled way to enable the incorporation of new modules in later project phases.
- [*Req.NF11*] **Documentation**: The usage of the system should be documented as well as the program code.

2.4 Conclusion

This chapter has defined requirements for an automatic generation of mashups. It has defined use cases within the financial domain that outlined important functional requirements. In addition, non-functional requirements have been described that define important criteria for the composition of web services. These multiple objectives have to be considered in the mashup framework. In addition, this chapter analysed the adoption of requirements from Data Warehouses and Online Analytical Processing, which focus on the aggregation and analysis of data from different sources within the enterprise. The next chapter investigates related work on mashups and mashup frameworks, to evaluate the present limitations and to outline the need for an automatic generation of mashups.

[12] `http://www-01.ibm.com/software/de/websphere/portal/index.html`

CHAPTER 3

Related Work on Mashups and Mashup Frameworks

In the last few years a variety of mashups has been developed and various frameworks has been proposed. Section 3.1 provides an overview about the different characteristics of mashups. Section 3.2 describes frameworks for the creation of mashups, in order to identify limitations that are described in Section 3.2.4. Section 3.3 concludes this chapter with an overview about the requirements that are not addressed by the existing mashup frameworks.

3.1 Mashup Patterns and Characteristics

The structure and development of mashups is not standardized by an organization, as it is done by the World Wide Web Consortium (W3C)[1] or other organizations for many other technologies. The missing standard or reference model motivates an investigation of mashup patterns. So far, mashup patterns have been only limitedly an object of research. An initial survey about mashup patterns can be found in Wong et al. [WH08]. This characterization refers to existing mashups, registered at the mashup directory ProgrammableWeb[2], which contains more than 3400 existing mashups. Amongst other things, it considers **data**, **function** and **presentation** related properties as well as **client-side** and **server-side** processing. Furthermore, the **delivery types** of mashups are outlined. In addition, the characterization covers **enterprise mashups** and outlines the differences of **transactional** and **analytic** mashups. Furthermore, this section describes **economic incentives** for the provision of mashups. The final part of this section pro-

[1] `http://www.w3.org/`

[2] `http://www.programmableweb.com`

vides an overview about the identified mashup patterns.

3.1.1 Mashup Data

Mashups retrieve content from a set of different types of data sources to aggregate [WH08] different data. These could be information web services [AKTV07] [TSP+07] [Yee08, p.4] [TAR07], feeds, spreadsheets, CSV files, screen-scraped[3] content [TAR07] or even whole web pages [WH08]. In principle any structured or unstructured data source could be the grounding of a mashup. This heterogeneity of is one of the key characteristics of a mashup. Furthermore, it is important to denote that mashups combine various types of data sources, and thus also specify how the data is merged and aggregated.
In addition, Wong et al. [WH08] state that real-time behaviour is another important property of mashups in fast changing environments such as news (e.g. BBC News Map Mashup[4]) and financial information.

3.1.2 Mashup Functionality

Mashups do not necessarily combine only different types of data sources, they also could integrate functionality from disparate providers. The aggregation of functionality is often achieved through the incorporation of different Application Programming Interfaces (APIs) offered by various providers (e.g. GoogleMaps[5]). An API is a set of functions, procedures or classes that are provided by a library or web service and can be classified as **language-dependent** or **language-independent**.
A language-dependent API utilizes the syntax of a programming language in a particular context. An example of such an API is the JavaScript[6] library of Google Maps. The JavaScript code has to be inserted or referenced in the HTML code of the web page, to create a map as well as points on it (e.g. 5TVs uses the Google Maps JavaScript library to place TV channels on a map[7]).
Language-independent APIs are not bound to a particular programming language. Web services are prominent examples of such APIs. In fact, this merging of function-

[3] Screen scraping refers to the process of extracting data by software from the user interface output of another application. In context of the Web, the screen scraper extracts data directly from the web page.
[4] `http://dev.benedictoneill.com/bbc/`
[5] `http://code.google.com/apis/maps`
[6] `http://developer.mozilla.org/en/Core_JavaScript_1.5_Reference`
[7] `http://5tvs.com/internet-tv-maps/news`

ality refers to the composition of web services, such that outputs from one web service can be used for the invocation of other web services.
Plenty of the manual developed mashups are restricted to read and filter [WH08] capabilities on the aggregated data (e.g. Housingmaps[8], 2RealEstateAuctions[9]). However, also the update of mashup data from the mashup to the source data (bi-directional data transfer) can be realized in mashups.
Collaboration and collective intelligence are key elements of the Web 2.0 vision. This drives the creation of mashups that enable users to collaboratively read, reuse, add and edit content, created by them or other humans (e.g. beenhere.TV[10], ApartmentRatings[11], AnglingAtlas[12]). Thus, collaboration functionality can be an important part of a mashup. Furthermore, mashups can also reuse the collaboration and personalization data, which has been collected at different external providers [WH08]. This enables the creation of a comprehensive personalization model. Yet, comprehensive views on behavioural and interest related data of humans could lead to unwanted privacy concerns. An example mashup is RottenNeighbour.com[13]. People are enabled to create statements about good or bad neighbors. The information can be placed directly on a map and are accessible by every web user. For instance, an employer can look for good or bad neighbour information about an applicant. This personalized view on the behaviour is not under control of the pertaining people and thus has to be evaluated critical. Warner and Chun give an overview about privacy protection [WC08] in this context.
Mashups often reference to external business processes to support real world effects. AmzWish[14] is a good example of such a mashup. It provides the personal Amazon[15] wish list to other people. Furthermore, it enables them to click on an wish list item to buy it for the creator of the list (or themselves) at the Amazon web site. In this case, the user is not able to directly buy the item, but could click on a link that refers to a selling process of a book selling company.

[8] http://www.housingmaps.com/
[9] http://www.2realestateauctions.com/
[10] http://www.beenhere.tv/
[11] http://www.apartmentratings.com/
[12] http://anglingatlas.com/
[13] http://www.rottenneighbor.com/
[14] http://www.widgetbox.com/widget/AmzWish
[15] http://www.amazon.com

3.1.3 Mashup Presentation

The presentational styles of mashups are not limited. Mashups enable users to access remixed data in a new visual way (by alternative user interfaces) that has not been intended by the data provider [WH08]. Amongst others, data could be presented as a spreadsheet, map, tree, chart or combinations of them. Mapping mashups are the most prominent type of mashups (e.g. ProgrammableWeb contains more than 1700 mapping mashups). However, also other types of mashup presentation especially for not location based data are increasingly important (e.g. Daily Mashup[16] , mashfot[17]).
A mashup can also directly utilize the presentation of an independent provider. For instance, it can use inline frames (called iframes). The iframes are part of the HTML 4.0 [RHJ99] standard and allow the embedding of external HTML code into a mashup page. In this case the presentation of the external provider is directly utilized in the mashup.

3.1.4 Mashup Delivery Types

The delivery type is another characteristic of a software application. Mashups can be delivered as a web page, web service, web portal page, portlet[18], widget[19][WH08], or as an other type of web-application as well as a desktop application.
The delivery type determines how data, functionality, and presentation of the mashup is delivered. For instance, if a mashup is delivered as a web service, which aggregates a specific set of feeds, it provides only aggregated data. Such mashups can be called data mashups and can be utilized by other applications. Mashups delivered as widgets, web pages etc. also define the presentation and the functionality in addition to the aggregated data.

[16] `http://dailymashup.com/`

[17] `http://www.mashfot.com/`

[18] "*A portlet is an application that provides a specific piece of content (information or service) to be included as part of a portal page. It is managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to information systems.*"[Hep08]

[19] A widget is a little application that is not used as an separate program, instead it is incorporated into an existing user interface or web page. Widgets are also sometimes called Gadgets or Applets.

3.1.5 Enterprise Mashups

Mashups can also be classified as special enterprise mashups. Enterprise mashups differ mainly in the amount of available data sources and functionality. Naturally, an enterprise has significant more intra-enterprise web services, databases etc. than a private user. Besides this, the enterprise has to be compliant with local regulations and corporate governance. Furthermore, the decisions which are based on enterprise mashups may lead to key impacts on the company performance, in contrast to mashups for private users. Therefore, the development of an enterprise mashup framework has to adhere to a variety of different requirements. Hoyer and Fischer state that enterprise related mashups put a visual interface to the existing Service-Oriented Architecture (SOA) of companies [HF08]. Mashups can and should therefore utilize the existing investments in information technology.

3.1.6 Client-side vs. Server-side Mashups

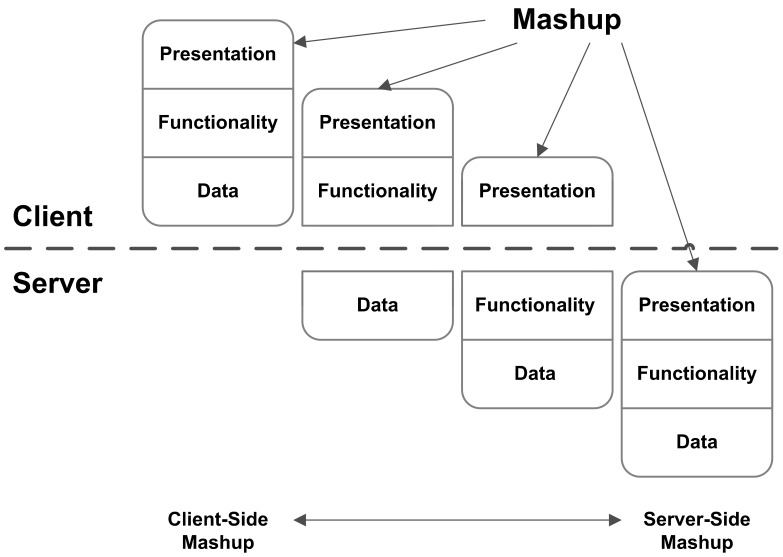

Figure 3.1: Client-side vs. Server-side Mashups

Mashup types can be also characterized by client- or server-side processing. A client-side mashup gathers and assembles data completely at the user client. Otherwise, it is also possible to create the mashup completely at a server. In that case, the gathering and aggregation of data as well as the presentation is created by the server and then transmitted to the client. In fact, there can be of course also various hybrid forms of server- and client-side mashups. For instance, the data of the mashup can be assembled on the server, whereas the functionality and the presentation is generated at the

different clients in dependence to their needs.

3.1.7 Transactional vs. Analytic Mashups

Transactional and analytic mashups should be distinguished (Figure 3.2), because the first are part of business processes (e.g. procurement), while the second are part of decision processes (e.g. tactical asset allocation in investment decisions).
Transactional mashups change the state of the environment and create real world effects. For instance, in a procurement mashup data retrieved from external suppliers as well as the internal data from SAP systems could be remixed. Furthermore, the mashup could directly provide functionality to buy products and to control the status of the process.
A mashup could be called analytic in the case of the utilization of information web services or other data sources as part of decision processes, which are not directly part of business transactions (e.g. Stockaholic[20]). However, decisions based on analytic mashups can lead to transactions. Figure 3.2 provides an overview about the differences of transactional and analytic mashups in the enterprise environment.

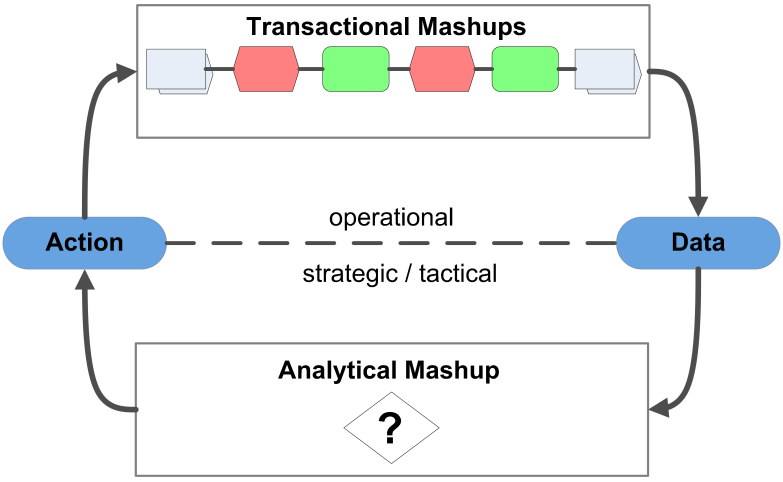

Figure 3.2: Mashup Categories

3.1.8 Economical Characteristics of Mashups

Mashups are a concept of the so called Web 2.0 [O'R]. The Web 2.0 is a set of concepts and principles that focus on software as a service (SaaS), importance of data

[20] http://www.editgrid.com/tnc/vincent/Stockaholic.new

sources, users as co-developers, collective intelligence as well as lightweight user interfaces [O'R]. However, present enterprise executives do not recognize mashups as a very important factor. In a recent global survey by McKinsey [McK] only 21 percent of 1046 executives stated that they are using mashups or plan to use them, while 54 percent do not consider mashups. Furthermore, 1801 of 2847 executives did not assess this point of the survey. This scepticism about mashups could have several reasons that should be investigated.

Mashups should be promising, if they create or fit into existing business models, enable the creation of new business opportunities or addresses local regulations. Due to this, an excerpt of business models of existing mashups is described below.

Plenty of mashups provide a unique entry point for special product categories (e.g. top54u.com[21]) and items of different providers and reference to their existing selling and buying business processes. The creator of the mashup is rewarded for sold or rent items, which is in fact the incentive for the development of the mashup.

Furthermore, it is also possible that a mashup is completely based on an implemented advertising business model [KBS+08]. This means that the provider of the mashup has to provide a special mashup service that attracts many users and is the basis for the reward of advertisements.

Mashups do not have to be necessarily consumer oriented, a service provider itself could also provide mashups to streamline its access to business process transactions.

In addition, mashup business models should also consider the analytical decision support of agents as described in the previous section. A better and faster decision making promises to directly result in reduced time and costs and therefore higher efficiency.

This short investigation shows that the scepticism of the executives of the McKinsey survey seems to be a general aversion or lack of knowledge about this technology, than the lack of business models. Instead, it seems that many of the executives do not perceive the potentials of mashups, which could be a reason for the 1801 executives that did not asses this point of the survey.

In a recent report of SERENA Software Inc., a provider of business mashups, the implementation of mashup technology for a client resulted in four million dollar annual cost savings through the streamlining of development processes [Inc07].

In general it seems to be clear that the incorporation of existing enterprise technologies as well as an easy enterprise-wide implementation should support the utilization of mashup frameworks.

In the last years, the main focus of companies relied on the creation of service oriented architectures (SOA). This is also confirmed by the McKinsey survey, where 80 percent of

[21] `http://shopping.top54u.com`

2615 executives said that they are using or plan to use Web Services in their enterprise [McK, p.5]. Service-oriented computing can lead to improved quality of applications through higher-level abstractions as well as standards-based interoperability [SH05, p.78]. The implementation of such a SOA is a strategic decision of enterprises, which has to be considered by the current scientific research. This means that new technologies should utilize the investments in SOA to make them attractive to the executives.

3.1.9 Conclusion

The previous explanations has described different mashup patterns, which are summarized in Table 3.1. The mashup patterns are important to achieve a shared understanding of the types of mashups.

Table 3.1: Summary of Mashup Patterns

	Description
P1	**Data, Functionality and Presentation**: Mashups provide the aggregation of various types of data sources and APIs from disparate providers. Furthermore, they visualize aggregated data in various ways.
P2	**Delivery Types**: Mashups could be delivered in a variety of different ways. Mashups that provide only data are called data mashups and can be utilized by other applications.
P3	**Client- vs. Server-side**: The creation of the mashup could be server-side or client-side.
P4	**Enterprise Mashups**: In context of the enterprise, mashups can utilize intra-organizational data sources and functionality. Furthermore, they have to be compliant with local regulations, laws and intra-organizational guidelines.
P5	**Transactional vs. Analytical Mashups**: Analytic mashups support decisions within the enterprise. Transactional mashups aggregate data and functionality for specific business processes and support the execution of transactions.
P6	**Economical Characteristics**: Mashups can support the streamlining of business and decision processes. Furthermore, they could be part of consumer oriented business models. Enterprise mashups should utilize the existing investments in SOA to make them attractive to the executives.

3.2 Mashup Frameworks

This thesis proposes a framework for automatic generation of mashups. This section therefore analyses related work on mashup frameworks. The first part describes existing approaches that are related to **automatic creation of mashups**. The second part gives an overview about the **mashup framework market**, which is described in detail in the third part of this section. The present **limitations of mashup frameworks** are provided in the final part of this section, which will lead to insight that the creation of mashups should be done in an automated manner.

3.2.1 Automatic Creation of Mashups

The automatic generation of mashup applications has been only limitedly part of research activities.
Carlson et al. [CNPZ08] propose a mashup framework for automatic composite mashup applications based on Lotus Expeditor. The framework focusses on mashups of non-web service components (e.g. widgets). In contrast to Carlson et al., this thesis focuses on the mash-up of background information provided by web services and not on the composition of different widgets. This means that in this thesis the focus lies on the aggregation of data, whereas in the approach of Carlson et al. different components (e.g widgets) are aggregated that contain data, functionality and presentation.
However, Carlson et al. describe the programmatic inputs and outputs of an application component by Web Service Description Language (WSDL) [CCMW] [CMRW].
The composite mashup applications are created by a match between a WSDL and a collection of target WSDLs. The score is based on the number of matching terms found. Furthermore, there is a second search performed that considers semantic matching on the semantic annotations defined by SAWSDL [FL07]. Based on the two scores the best matching component is selected.
The framework is mainly based on two paradigms: match and compose. Match refers to "*Using the output from a single web service and finding a second web service that can take that as input. The developer can continue this process and string together several web services with application specific control-flow pattern in order to complete a business process*"[CNPZ08].
Compose refers to: "*Starting with a known output and a known input, the developer uses search techniques that can allow them to find one or more services that will transform the output of the first web service into something that can be consumed by the final web service*"[CNPZ08].
In general, the compose approach of the framework could be denoted as a forward chaining approach, because the composite application is derived iteratively by adding

the best matching components to a starting component.
Forward chaining is often an inefficient approach [RN03, p.384] because it considers irrelevant actions (often called undirected search [KSKR05]). In this case, the framework may suggests highly matching components, but these components could be irrelevant, if they are not relevant to achieve the present goal. Therefore, the iterative selection of highly matching components is not completely sufficient to create mashup applications in an efficient way. As stated in Chapter 2 the composition of different components has to be part of a planning process.
This thesis focusses on a scalable approach for composition of information web services that efficiently walks through the search space. This is in accordance with Carlson et al.[CNPZ08], who outline outline the importance of a process based integration approach for their future work.

3.2.2 Market Overview

The increasing interest in mashups has led to a variety of different tools, editors, repositories and frameworks by business and research. The intention of this part is to give an representative (not entire) overview about the mashup framework market.
An existing market overview of mashup tools can be found in Hoyer and Fischer [HF08]. It classifies the mashup tools by their target group (consumer or enterprise) as well as functionality.
This thesis has a focus on the **skill requirements** of target users, to point out general limitations as well as the need for an automatic and adaptive generation of mashups. The skill requirements of users depend on the **development approach** that is used to create the mashup. The development approach of mashups could be based on **manual**, **semi-automatic** (tool-supported) or **automatic** creation. The following explanations describe these criteria in detail.

Manual creation means that data sources and functionality have to be integrated by programming. This is very complex, due to high requirements on detailed programming expertise [WH07] and knowledge about web technologies. The difficulty of programming (even for experienced developers) is to transfer the problem domain representation into an abstract, detailed, and technical language of the programming environment [SBD03, p.287].
The information needs, preferences, interests etc. are different among users. Thus, manual assembled mashups may have only a general scope that supports the preferences of a variety of different users. Therefore, it is likely that often an appropriate mashup is not available for a specific problem. Therefore, **semi-automatic** tools have been de-

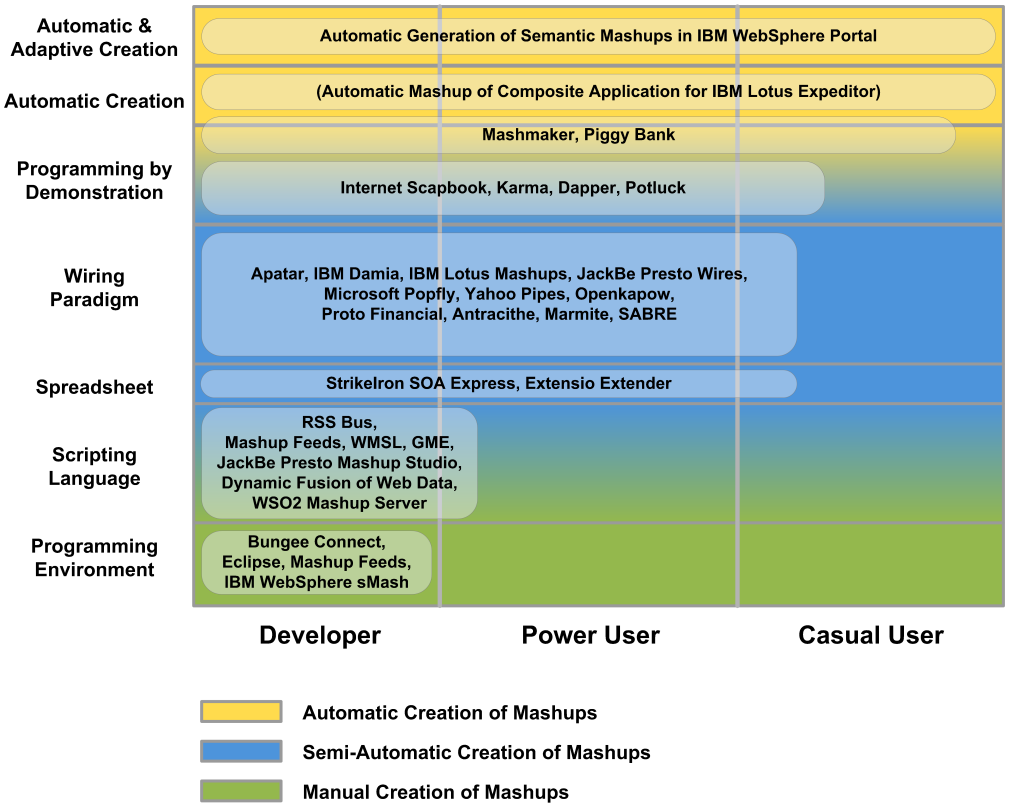

Figure 3.3: Mashup Framework Market Overview

veloped that support end-user programming (EUP), to enable users to create, adapt or extend applications on their own, in order to solve specific problems more efficiently [SBD03, p.287].

End-user programming is an important trend of information technology and the Web 2.0 [O'R], because it promises that the applications directly fit with the user needs and that the savings for development of applications in enterprises could be increased. This trend ends up the classic software release cycle, which means that the time for newly added functionality is decreased to weeks, days or hours. In this context, users as co-developers become increasingly important to react quickly on changed requirements [O'R]. End-user programmers has been defined as *"people who write programs, but not as their primary job function"*[MKB06]. In fact, the program is just written to achieve a main goal more efficiently.

The existing end-user oriented tools are based on a variety of approaches such as **wire based integration** of data and functionality, **scripting languages**, **spread-sheets** or **programming by demonstration**. In the following, these categories are shortly outlined.

The majority of end-user oriented tools simplify the creation process on the data layer through workflow based (piping) integration of data sources as well as through work-

flow based (wire) integration of functionality [HF08]. In the following, this is called the **wiring paradigm**.
Besides the wire based approaches, some tools utilize **spreadsheet** functionality or are integrated into spreadsheet standard software. Many people are familiar with spreadsheets and thus these mashup systems promise to have a good usability.
Furthermore, the research has turned out systems that focus on **programming by demonstration**, which enables users to "learn" the system by a set of examples.

Figure 3.3 gives an overview about the mashup framework landscape in relation to the development paradigm as well as the required skills of the user. The skill of a user is assumed to be divided into the categories **developer**, **power user**, and **casual user**.
A developer should be familiar with programming, web technologies, different APIs as well as the usage of development tools. A power user has no programming skills, by definition of this thesis, but does have detailed functional knowledge about a specific tool or set of tools and thus is able to create complex mashups. Casual users only have the skills to use the functions of a web browser and are able to navigate through the Web.

3.2.3 Detailed Analysis

3.2.3.1 Tools based on the Wiring Paradigm

Wire-oriented tools such as Apatar[22] [Khi08], IBM Damia [SAM+08], JackBe Presto Wires[23], Microsoft Popfly[24], Yahoo Pipes[25], Openkapow[26], Proto Financial[27], Anthracite[28], SABRE [MLA08] remix and merge data, functionality or presentation through a graphical wiring of basic building blocks. This manual connection is sometimes called wiring or piping of different modules, connectors, components or blocks. The available components provide different functionality (e.g. data retrieval, data transformation, data presentation etc.) and have to be connected to achieve the desired coordination of the mashups. The tools often support different data source types such as RESTful and / or SOAPful (e.g. Openkapow, Proto Financial) web services, databases, spreadsheets and

[22] http://www.apatar.com/product.html
[23] http://www.jackbe.com
[24] http://www.popfly.com/
[25] http://pipes.yahoo.com/pipes/
[26] http://openkapow.com/
[27] http://www.protosw.com/
[28] http://www.metafy.com/products/anthracite

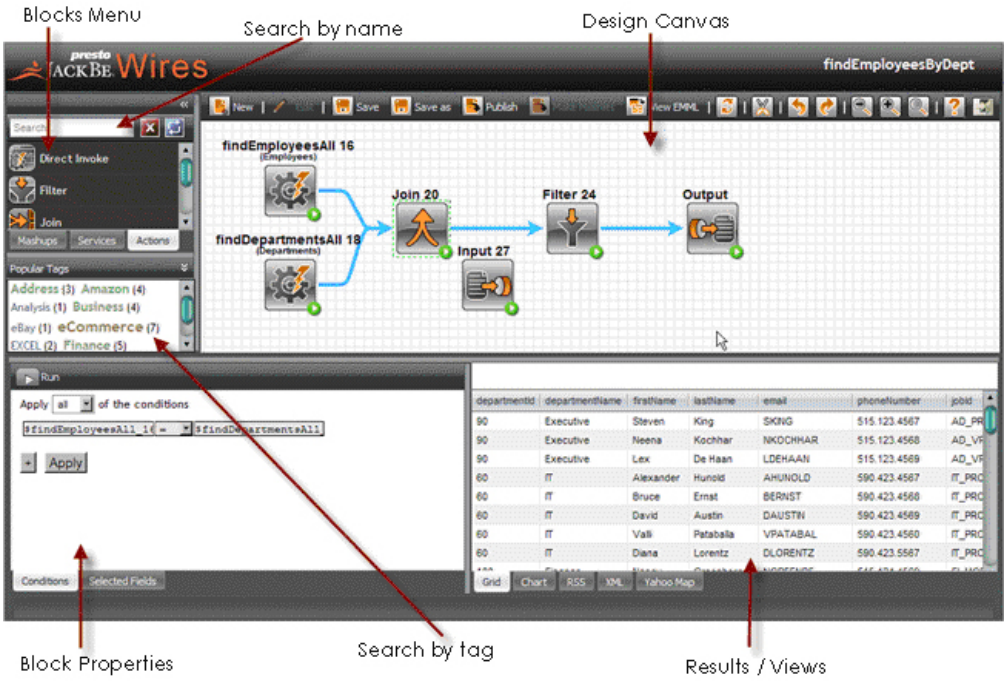

Figure 3.4: JackBe Presto Wires Mashup Composer

CSV files.

In addition some tools such as Apatar and JackBe Presto provide also bi-directional data transfer, in case of a change of the mashup data, the changes are published to the initial data source. This enables enterprise mashups to be actively involved in business processes transactions.

The data modules could be often connected to presentation blocks that provide different presentation styles such as map, table, tree etc..

Proto is specialized to the financial domain and provides the creation of comprehensive trading and research mashups as well as integrated functionality to analyse risks (e.g. processing of Value at Risk (VaR), Beta and correlation metrics).

The IBM QED (Quick and Easily Done) Wiki uses various data sources to create views on data. Furthermore it supports common Wiki functionalities like commenting, messaging etc.. It has been replaced by Lotus Mashups. Lotus Mashups [29] provides an user interface for assembling of personal, enterprise and Web content and is part of the IBM Mashup Center[30]. Different widgets can be wired through a lightweight user interface. The resulting mashed-up pages can be shared between the users.

[29] http://www-01.ibm.com/software/lotus/products/mashups/

[30] http://www-01.ibm.com/software/info/mashup-center/

BEA AquaLogic Pages[31] is a web authoring and mashup framework. It provides Wiki page like functionality to edit and create web pages. Furthermore, it provides drag-and-drop components that can be wired (like a workflow) to other components directly on a web page. The pages itself reuse data that has been created in advance in specific data spaces. The data could be retrieved from web services as well as from local sources. The presentation offers the creation of tables, lists, images, maps and normal texts.
The SABRE framework [MLA08] tries to bridge the gap between classical SOC applications and mashups. It uses the Reo coordination language to specify web service based mashup logic. SABRE focuses exclusively on tools for data-driven service coordination. Reo coordination is specified by different connectors that enables, for instance, the specification of parallel executions, synchronizations, transformations well as as filters or FIFO buffers [MLA08]. The SABRE implementation is based on a graphical mashup design tool as well as a runtime execution environment.
In general, the tools require basic skills about data processing, workflow concepts as well as knowledge about the types of data sources such as databases, XML etc.. The graphical user interface design simplifies the creation process. Due to this, they can be used by power users or developers that have significant knowledge about the functionality of the framework.

3.2.3.2 Tools based on Spreadsheets

Spreadsheet-based tools such as StrikeIron SOA Express for Excel[32], Extensio Excel Extender[33] focus on the remix of data. Unlike the workflow-based tools, the data is directly interchanged at a spreadsheet. This means that the outputs of a data source are written to cells that have been previously selected by the user. The cell values then serve as inputs of subsequent data source queries. StrikeIron SOA Express and Extensio Excel Extender utilizes SOAPful web services to create mashups. In addition Extensio Excel Extender can provide access to SAP, several databases as well as flat files.
The tool usability promises to be good among all types of users, because many users are already familiar with spreadsheets, especially in the enterprise environment. In fact, the users are able to leverage the powerful processing capabilities of spreadsheets to analyse data and create charts.

[31] http://www.bea.com/framework.jsp?CNT=index.jsp&FP=/content/products/aqualogic/pages/
[32] www.strikeiron.com/tools/tools_soaexpress.aspx
[33] www.extensio.com

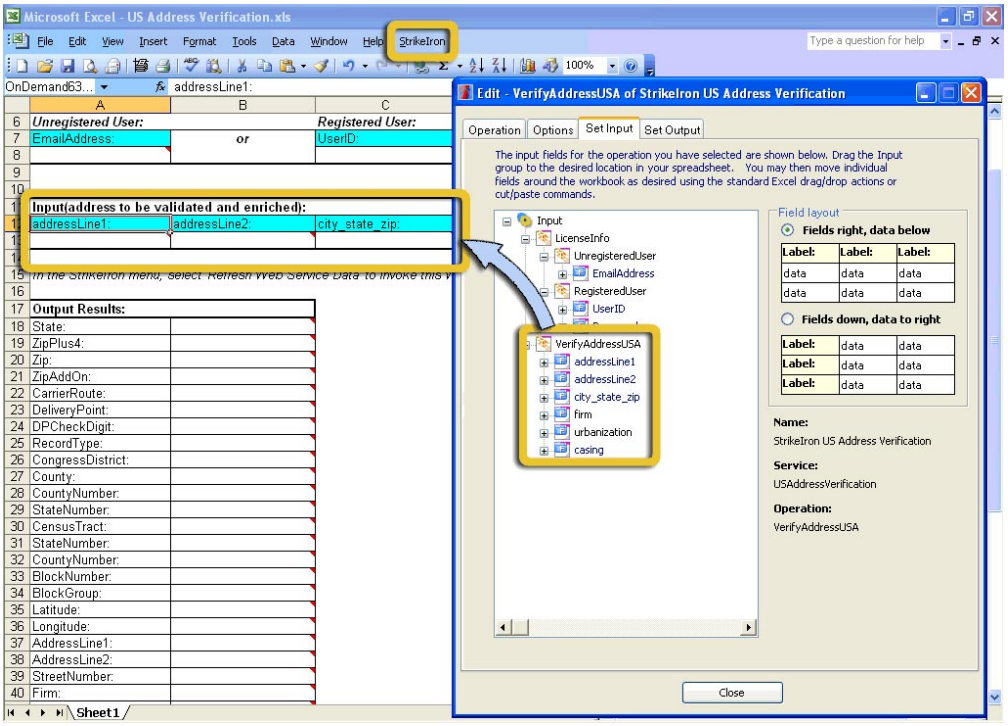

Figure 3.5: StrikeIron SOA Express for Excel

3.2.3.3 Scripting Languages

The development of mashups is supported by various tools that are based on domain specific scripting languages such as Google Mashup Editor[34] (GME), Web Mashup Scripting Language (WMSL) [SHSG07, p.1305], Dynamic Fusion of Web Data [RTA07] [TAR07], WSO2 Mashup Server[35].

GME allows developers to manually create mashups based on specific extensible markup language constructs. The incorporation of different mashup specific tags (e.g. ”gm:list“, ”gm:map“, ”gm:search“ etc.) as well HTML, CSS and JavaScript leads to specific mashup pages.

The Web Mashup Scripting Language (WMSL) [SHSG07, p.1305] enables the mashup of different web services. Different sections enable the import of Web Service Description Language (WSDL) files, XML-Schema definitions (XSD), OWL ontologies, and other WMSL scripts directly into the HTML page.

RSSBus is a data feed generator that uses the RSS protocol as interchange mechanism. The RSS 2.0 [Boa] extension allows to interchange any type of data, not only news items or blog postings. The feeds itself are retrieved through special connectors. Amongst

[34] `http://code.google.com/gme/`

[35] `http://ws02.org/projects/mashup`

others, it is possible to access a variety of Amazon web services, CRM systems, office applications, emails as well as RESTful web services. However, there is no support for SOAPful web services [RSS08].
Rahm et al. propose a framework architecture for the development of dynamic data integration mashups, which is based on a high-level scripting language [RTA07] [TAR07]. The scripting language defines a workflow by generic operators that can be applied to different structured or unstructured data sources and web services. The operators are set oriented and enable the classic set operations (e.g. union, intersection etc.) as well as data transformation (e.g. fuse, aggregate) [RTA07].

```
<gm:page title="Xmap 1" authenticate="false">
...
    <gm:map id="Xmap" style="border:solid black 2px" control="large"
        maptypes="true" data="\${Xlist}" latref="geo:lat" lngref="geo:long"
        infotemplate="XmapTemplate">
         <gm:handleEvent src="Xlist" />
       </gm:map>
...
  <gm:template id="XmapTemplate">
    <div align="center" repeat="true">
      <h3><gm:text ref="atom:title" /></h3>
    </div>
  </gm:template>
...
</gm:page>
```

Listing 3.1: Google Mashup Editor Excerpt of a Mapping Example

In general, it seems to be too complicated for a non-developer, to create such scripts in an appropriate time, because complexer mashups will need much more script code. Furthermore, the composition of the components is a hand-coded solution. This means that the developer has to analyze the data sources manually to achieve an appropriate data flow (mediation).

3.2.3.4 Programming

Tools are available that create mashups based on an integrated development environment (IDE). IBM WebSphere sMash[36] is a development and execution environment for dynamic web applications. It enables an easy reuse of web services and enables an

[36] http://www-01.ibm.com/software/webservers/smash

rapid integration of different web services.
BungeeConnect[37] is another platform that is offered as an online service. It is possible to create comprehensive web applications. Bungee automates the import of publicly available web services as well as traditional databases and data warehouses. However this approach is nearly equal to a complete manual development of mashups, which is supported by a variety of other IDEs.

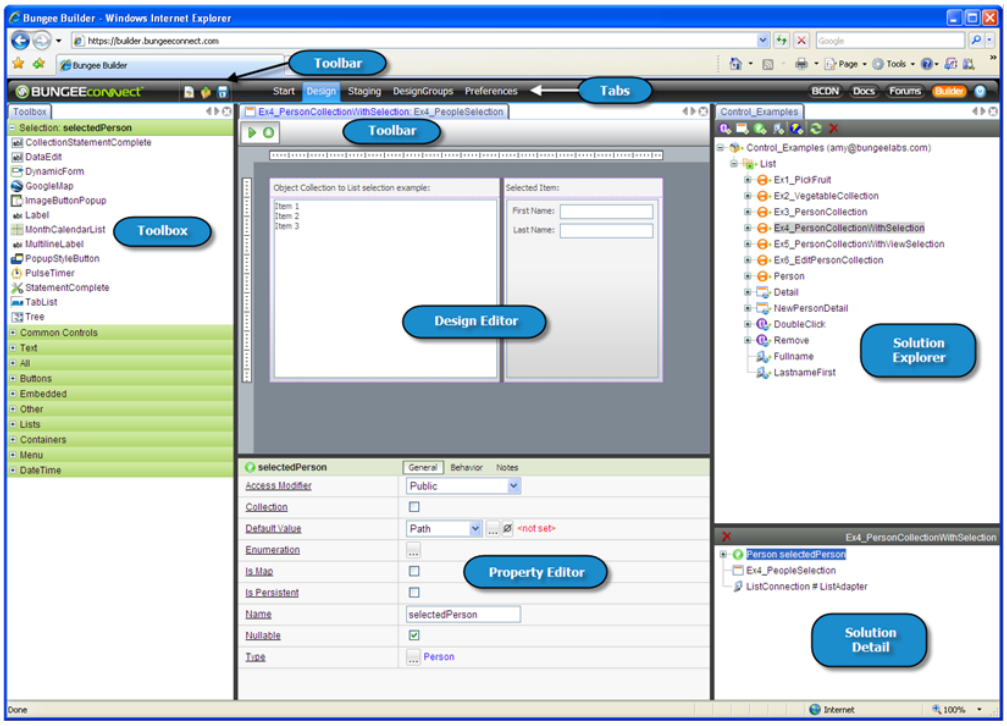

Figure 3.6: BungeeConnect Platform

3.2.3.5 Programming by Demonstration

Programming by demonstration enables users to learn a system by the provision of examples. The Internet Scrapbook [SK98] system allows users with little programming skills to automate recurring browsing tasks. The user is enabled to demonstrate which parts of a web page are interesting to him, by copying the relevant data from multiple pages to a personalized mashed-up page. The extraction of the data is based on the HTML structure of the specific web page.
Dapper[38] is an online service that is able to create an API for any web site. The source web site has to be initially specified, then the user can select graphically from some

[37] http://www.bungeelabs.com/
[38] http://www.dapper.net/

sample outputs the fields that should be extracted.
Karma [TSK08] utilizes **programming by demonstration** to extract lists of data from web pages through simple drag-and-drop of elements of a web page. The system leverages the DOM tree information of the browser and creates a table of the data. The data can be automatically joined with other tables derived from other pages, by a match of attribute name and value pairs.
Huynh et al. propose the Potluck[39] mashup tool, which enables users to create mashups without the need of programming knowledge [HMK07]. Potluck takes as input a set of URLs that contain the data that should be remixed. The datasets from the web pages are shown in a tabular environment. Fields that represent the same semantic attribute can be set as equal through a simple drag and drop of the field names. In this case, the presentation is automatically rearranged. Thus, the mediation between the different semantic heterogeneities is done implicitly by the user, while he is acting on the retrieved data. Furthermore, the system enables the user to clean up and homogenize the data in a faceted browsing environment[HMK07]. A faceted browsing environment, based on a faceted classification, allows a user to explorative search in huge data sets by hierarchical concepts[YSLH03]. A faceted classification allows the assignment of multiple classifications to an entity, thus the entities could be ordered in different ways.

3.2.3.6 Framework Deployment

Mashup frameworks could be also characterised by its deployment approach. The frameworks are deployed in various ways such as browser plugins, desktop applications or online services. Extensio, Apatar, Proto, StrikeIron SOA Express are desktop applications. Microsoft Popfly, Yahoo Pipes, GME, BungeeConnect and Dapper are prominent online services.
Browser embedded tools are added as plugins to the browser. The browser based mashup of web pages often directly changes the appearance of the web page on the client. These tools focus mainly on easy extraction of data from web pages and the mashup of web pages while the user is surfing. This is in contrast to online services that often directly create a mashup page.
Examples of browser embedded Tools are Karma [TSK08], Hunterer Gatherer [SZM+02], Marmite [WH07] or Mashmaker [EG07b]. The Hunterer Gatherer system is a browser based tool that leverages the DOM tree information to extract user selected contents [SZM+02].
Marmite is a browser plugin that promises to be a mashup tool that enables end-users

[39] http://simile.mit.edu/potluck/

to create mashups without programming knowledge [WH07]. User can extract content from web-pages, transform data and aggregate it with other data sources (i.e. databases or web services) through an wire-oriented editor [WH07]. Users are immediately informed about the state of the data between the operations through partial runs of the data flow. The results are displayed as a spreadsheet.
MashMaker is a browser plugin that extends the normal process of browsing with mashup functionality. MashMaker does not require the specification of the mashup in advance by the users, as it is required by the workflow like tools [EG07b]. The software guesses a mashup that the user would find useful, based on the current browser content. At the end, the user even did not know the the page was mashed up [40]. Furthermore, MashMaker directly extracts entailed RDF data from a web site. If this is not possible, the tool extracts structured data from the raw HTML. The raw data is retrieved thru special "Extractors" that could be created for each web site. The extractors are stored on a special server and thus could be reused by all users of the framework [EG07a] [EBG+07].
PiggyBank[41] is able to process RDF data, which is embedded in web pages. Manual development of web page specific screen scrapers (e.g. for Flickr pages) allows the extraction of non-semantic data and their transformation to RDF. An advantage of PiggyPank is that the user has not to concern about retrieving and remixing of data, which is done automatically through the merging of RDF triples.
Mashmaker and Piggy Bank operate directly on RDF or use extractors to extract the data. However, RDF documents could refer to different vocabularies. It is unclear how the tools achieve ontology mediation or if the designers of the extractor components have to refer to one defined shared vocabulary.

3.2.3.7 Data Mashup Tools

Data mashup tools like Apatar, IBM Damia, Denodo Enterprise Data Mashup[42], Dynamic Fusion of Web Data [RTA07], Mashup Feeds [TSP+07, p.1128], Openkapow[43], RSS Bus [RSS07] [RSS08] retrieve, transform, aggregate and provide data to other applications. The tools virtually integrate content from any type of information source such as web web service, database, xml, file, data warehouse, SAP, Siebel etc.. Furthermore some tools such as Openkapow provide screen-scraping and thus offer the

[40] This behaviour is also an objective of this thesis, in regards to mashed up portlets in web portals.
[41] http://simile.mit.edu/piggy-bank/
[42] http://www.denodo.com/english/products.html
[43] http://openkapow.com/

possibility to grab data directly from web-pages. The results are often published as real-time data web services to other applications.
"*Mashup Feeds is a system that enables mashup developers to describe new integrated web service feeds as continuous queries over existing feeds and web services*"[TSP+07, p.1128]. Continuous means that the system automatically saves requested data in relation to a specific timestamp. Queries over the different timestamps are then called continuous queries [TSP+07, p.1128]. In other words, the results of the same service (with different timestamps) could be remixed. The system uses a scheduler to store the time series. The query syntax is similar to a programming language and could only be created by developers in a special developer module[TSP+07, p.1130].
PAYGO [MJC+07] proposes a new data integration architecture oriented to the Web. It is not based on a single data integration schema (like in traditional databases), instead it maintains a set of schemas that are clustered in relation to specific domains. Since an exact matching of those schemas is not available, the mappings of PAYGO are approximate, which means that there could be simple statements about related schemas as well as explicit user defined mappings. Thus the system provides mechanisms that enable the improvement of semantic relationships over time. The improvement is based on automated as well as suggestion related techniques that could also involve the feedback of the user.
Feed creators (or adapters [HF08]) provide special functionality to create feeds from a variety of different resources. They are often the basis of other mashup tools that utilize this functionality. IBM Mashup Hub is a server that stores and maintains information about feeds and widgets. Feeds could be created from various data sources such as CSV files, databases or XML files. Thus IBM Mashup Hub is a feed creator as well as a repository. Dapper[44] is an online service that is able to create an API that responds XML data for any web site.

[44] `http://www.dapper.net/`

3.2.4 Present Limitations of Mashup Frameworks

The following limitations of present mashup frameworks could be pointed out and are described in detail below.

1. Users need knowledge about web technologies and / or programming
2. Users are overstrained to select appropriate building blocks of wire-based modules
3. Efficient selection of data sources (e.g., selection of information web services) is not addressed by mashup frameworks and has to be done time consumingly by the user
4. Users need knowledge about the structure and semantics of the data sources to merge them manually
5. Mashups developed manually or by tools have to be maintained for further reuse
6. Mashup selection for a particular task or information need is a complex problem that could not be solved appropriately in an appropriate time by a user and current mashup descriptions do not support machine enabled reasoning
7. Current frameworks do not consider automatic adaption of mashups

3.2.4.1 Users need knowledge about web technologies and / or programming

Programming or script based frameworks like RSSBus, Google Mashup or BungeeConnect require knowledge about web technologies (e.g. WSDL, SOAP) and programming to create mashups (see also [TSK08, p.140]). Therefore, it is difficult for casual users as well as power users to create personalized mashup applications in an appropriate time. It is likely that they could not achieve any productivity gains through the creation of a mashup in relation to a manual aggregation of data and / or functionality.
Wire-oriented tools provide data retrieval modules that are related to specific technologies such as WSDL, HTTP, database etc., which requires knowledge about these technologies to configure and select modules appropriately. This is in accordance with Wong et al., who argue that most tools still require too much background knowledge on web technologies and programming and focus mainly on lightweight user interfaces [WH07].
Automatic generation of semantic mashups promises to overcome this problem, because the required interaction with the user is decreased significantly. Furthermore, the knowledge requirements are relatively small, since the user may not even know that the underlying data, functionality and presentation is mashed-up.

3.2.4.2 Users are overstrained to select appropriate building blocks of wire-based modules

The selection of appropriate blocks (modules, connectors, components) of wire oriented tools could be difficult. The end-users have to translate the problem and domain specific, high level representation of a problem into an abstract, detailed, and technical workflow of components, blocks etc. In open systems like Popfly the count of blocks increases steadily, which makes it increasingly difficult for the user to select building blocks from the cloud. Otherwise, a limitation of the extension would restrict the functionality of the whole framework. Therefore, tools like Popfly and Marmite try to overcome this problem by a suggestion mechanism. However, often the suggestion is based on a comparison of complex data types, instead of using semantics. The quality of such a suggestion mechanism then depends on the assumption that developers describe equal concepts by equal complex data types, which is of course not mandatory.
The problem could be summarized as follows, the selection and extraction problem of information is shifted to a selection and combination problem of appropriate building blocks. The problem of appropriate block selection is similarly stated in [TSK08, p.140] and Carlson et al. [CNPZ08]. Furthermore, especially unexperienced users will not be able to use the tools out of the box. The time to study existing examples (learning by example) as well as the functionality of the software has to be noted. Thus there seems to be no significant time reduction for one time ad-hoc usage.
In a recent test of the Marmite system, Wong and Hong stated that many users without programming knowledge do not understand the data flow concepts behind such a wire-based editor[WH07] and could therefore not solve even simple problems. Another interesting result of this test was that, if users extracted some data from a source they would like to have other attributes of this data without adding extra operators for extraction and aggregation of related data[WH07].
Automatic generation of semantic mashups promises to overcome this problem, because the framework needs no user interaction to retrieve and merge relevant data. This should enable the user to concentrate on the real domain problem or task.

3.2.4.3 Efficient selection of data sources (i.e., selection of information web services) is not addressed by mashup frameworks and has to be done time consumingly by the user

The fundamental drawback of basically all existing platforms is a lack of **automatic data source selection**. This means that the user has to specify the web pages, web services, data bases, files etc. the mashup should rely on. Manual data source selection does not only require the location (e.g. URL) of the data sources, it requires also knowledge about the structure and semantics of the information that could be retrieved. This is important because otherwise the human agent could not evaluate if a specific data source is appropriate for the specific problem.
The internet and intranet could contain millions of data sources, which leads to the insight that the selection could be time consuming. Moreover, it is likely that the user also does not know many of the information sources and therefore is limited to a set of known ones, which may not satisfy requirements such as trust, reliability, security etc. Thus, the decision quality could be influenced negatively.
This thesis is fully based on information web services. The web services selection of an information web service is done automatically by utilization of semantic service descriptions. Furthermore, it is intended to incorporate the user preferences, interests, tasks and experience in the selection process.

3.2.4.4 Users need knowledge about the structure and semantics of the data sources to merge them manually

The aggregation of different data is handled in various ways. Spreadsheet based tools achieve the data flow through joint cells that are used as input and output fields. Similarly, wire-oriented tools are based on a manually defined data flow. In any case the alignment of different attributes has to be done manually or is suggested through complex type comparison, as described above.
Automatic generation of semantic mashups promises to overcome this problem, because the data sources as well as the retrieved data are described completely semantically, which enables an automatic assembling of the data.

3.2.4.5 Mashups developed manually or by tools have to be maintained for further reuse

The manual or tool-supported creation of mashups requires maintenance, if the mashups should be reused. This seems to be likely, because the user has invested time and other resources to create the mashup. Therefore, many frameworks provide repositories of mashups. However, this requires to define processes that determine when the mashup applications should be deleted as well as who is responsible for their maintenance. It has also to be specified, which actors (e.g. employees) get access to this mashup application or to specific populated data sources. Access rights seem to be important, because otherwise the mashup framework could not be used for sensitive data.
Automatic generation of semantic mashups promises to overcome this problem, because the mashups are created ad-hoc and discarded after their usage. Thus, the system requires no maintenance of the mashup applications. Moreover, the system maintains the information web services. This offers the possibility to create contracts between provider and requester that could be controlled by the management, which should lead to compliant mashup applications that consider local regulations and intra-organizational guidelines.

3.2.4.6 Mashup selection for a particular task or information need is a complex problem that could not be solved appropriately in an appropriately time by a user and current mashup descriptions do not support machine enabled reasoning

Productivity gains can be achieved through an efficient reuse of existing mashups. This means that the information selection and extraction problem is shifted towards a mashup selection problem. ProgrammableWeb[45] is an example of a mashup directory, which currently offers a list of 3046 mashups. It seems to be likely that the count of mashups increases rapidly as more users become familiar with lightweight mashup editors. In the context of ProgrammableWeb mashup selection is very difficult, because it relies only on human readable information, which itself only provide a short description about the mashup as well as a list of used APIs and tags. The absence of a semantic description of the capabilities of a mashup avoids an automated reasoning for mashup selection.
Furthermore, it is also very difficult for a human to evaluate if a mashup fits his particular user needs, because he has to filter out a mashup based on a short description and

[45] `http://www.programmableweb.com`

a list of used APIs. This requires also the knowledge about the APIs itself, to evaluate an mashup in relation to type, quality and reliability of the information provided by the mashup. Furthermore, a selection of mashups should also consider trust issues.
Tag clouds [46] or special mashup vs. API matrices [47] are also not appropriate to large scale repositories, because in the first case the user is forced to specify his information needs only by a tag, which seems to be not detailed enough. In the second case, the steadily increasing count of APIs and mashups [EG07b] leads to a huge matrix that is difficult to oversee. Regarding this, the user is also forced to have a deep knowledge about the APIs, to evaluate whether a combination of APIs meets his situational information needs or not.
Furthermore, Wong and Hong figured out that many mashups on ProgrammableWeb are poorly developed and only assembled for demonstration purposes to collect community feedback or to drive advertising revenue [WH08, p.36]. This means that even if appropriate existing mashups could be selected, it is still unclear if those mashups are working properly.
Automatic generation of semantic mashups promises to overcome these limitations, because mashups are generated situational for each user, which avoids an time consuming selection of existing ones.

3.2.4.7 Current frameworks do not consider automatic adaption of mashups

The available frameworks allow the creation of mashups to reduce the time of information gathering from disparate data sources. A high value could be achieved for those mashups that support recurring tasks. For instance, a manager looks every week for appropriate hotels and flights to his clients based on his meeting schedule. It would be ideal if the manager could reuse the mashup each time. However, the mashups adapt only in dependence of the data of the data sources. An example of an adaptive mobile mashup is the Telar system[BN08]. The data is retrieved automatically and provides data which is closely to the current location or point of interest (POI). The data is then automatically displayed on a map. However, besides the current location or POI the adaption should be also based on interests, tasks and experience of a user.
It has also to be considered that the available data sources are not static, this means that there is no guarantee that a data source is available, when the mashup is invoked the next time. Therefore, the mashup may has to be reengineered to avoid errors or crashes. An automatic generation of mashups, which involves an automatic selection of the data

[46] `http://www.programmableweb.com/search`
[47] `http://www.programmableweb.com/matrix`

source close to usage, avoids time consuming reengineering and automatically considers changed interests, tasks and experiences.

3.3 Conclusion

This chapter has explained the common patterns and characteristics of mashups. The characteristics and types of mashups are manifold, which was expectable, since there is no standardization of mashups.
The mashup framework market is dominated by semi-automatic tools that provide an graphical user interface to allow end-users to create mashups. However, as explained in the previous section, the existing frameworks have several limitations. They do not provide an efficient mean to select web services. Furthermore, the composition of the different components has to be done manually, which could be time consuming. Therefore, these systems do not address the requirements for an automatic generation of mashups (see Chapter 2). Especially, the automatic selection and composition of web services is not addressed by these frameworks. Therefore, a dynamic and automatic gathering of background information can be therefore not achieved with the existing frameworks.

CHAPTER 4

Theoretical Foundations

This thesis proposes the automatic generation of mashups. The mashups augment contents with background information that have been dynamically retrieved from different independent web services.
The previous chapter outlined the limitations of the existing frameworks as well as the need for such an automatic generation. This chapter points out important theoretical foundations of this thesis, which are required to achieve the automatic generation of mashups.
Figure 4.1 outlines the main questions that are answered in this chapter.
As stated in the requirements analysis (Chapter 2), the automatic generation of mashups

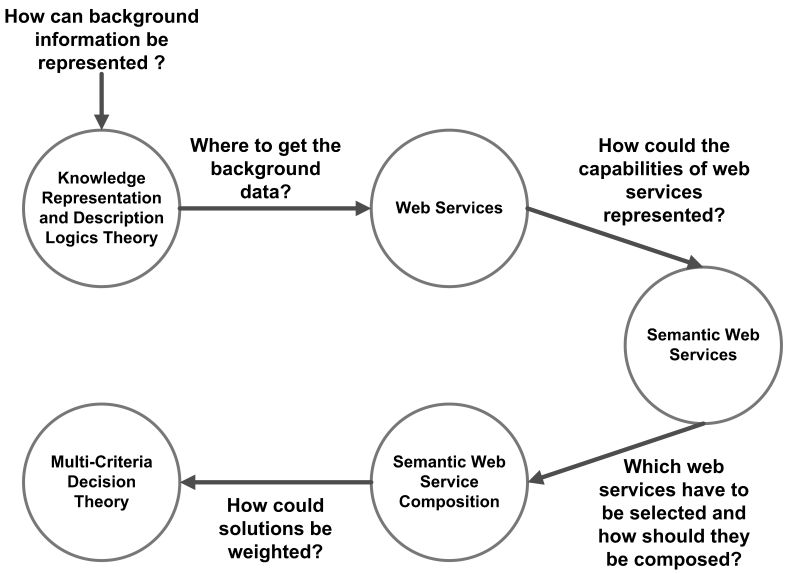

Figure 4.1: Chapter Overview

should utilize semantic technologies to enable automatic processing and inference by agents. Therefore, Section 4.1 and 4.2 give an overview about knowledge representa-

tion on the Web and theory of description logic.
The background information of the mashups are retrieved from web services. Therefore, Section 4.3 describes the central principles of web services and service oriented computing.
The existing mashup frameworks are based on a manual selection of web services. Furthermore, many tools allow the composition of different web services through a wire oriented approach. However, as stated in the previous chapter, the selection and composition of web services should be done automatically. This is addressed by the research on Semantic Web Services, described in Section 4.4. Furthermore, Section 4.5 describes the theoretical foundations of the automatic planning of web service composition.
Finally, there is the need for an weighing of found web service compositions in dependence to the information criteria, which have been stated in the requirements analysis (Chapter 2). This topic is addressed by Multi-Criteria Decision Theory which is described in Section 4.6.

4.1 Knowledge Representation on the Web

The Extensible Markup Language (XML) [BPSM$^+$] provides data formats for documents and structured data and is the preferred syntax of many higher level approaches. The structure of an XML document can be easily processed by a software agent. XML provides no means to express the semantics of the format used by a document [SH05, p.21]. However, this is important to share knowledge and information among different independent and distributed applications. Listing 4.1 and 4.2 provide a simple example of the problem.

```
<?xml version="1.0"?>
<name>
 <foreName>Thomas</foreName>
 <lastName>Fischer</lastName>
</name>
```

Listing 4.1: XML Example 1

The contents of Listing 4.1 and Listing 4.2 have equivalent information (not equivalent data). However, both XML documents do not specify any meanings to achieve a shared understanding of the data. Therefore, agents are not able to infer that ”foreName” could be seen as semantically equal to “preName‘ unless they commit to a shared vocabulary.

```
<?xml version="1.0"?>
<name>
 <preName>Thomas</preName>
 <surName>Fischer</surName>
</name>
```

Listing 4.2: XML Example 2

Furthermore, the agents are not able to infer that the string ”foreName“ represents a forename unless it is referenced to a shared vocabulary or it is directly defined in the application code. However, the programmatic definition of the meanings is not suitable, because there is a huge mass of other XML enitites that could refer to forename, for instance the German word ”Vorname“ or even the string ”fNm”. Thus, the programmatic definition would not be able to deal with other XML formats.
The research community has proposed frameworks such as RDF [BAB], RDF-S [BR] and OWL [BvHH+] that provide more expressive meanings. Nevertheless, XML provides a structure that could be traversed by machines and thus is also often utilized by higher level approaches.

4.1.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a framework for representing information on the Web, motivated by an increasing demand on agent driven automation of data processing outside the initial environment, as well as the combination of data from different applications [KC]. RDF is based on formal concepts. In general, RDF allows

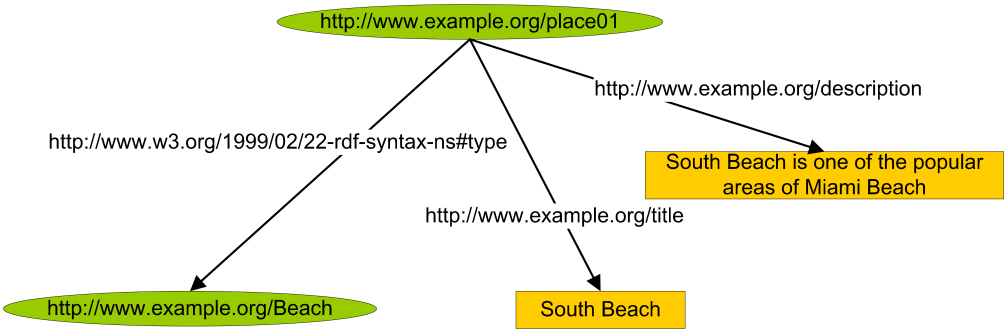

Figure 4.2: An RDF Graph Describing South Beach

anyone to make statements about any resource, which could be a material or immaterial thing. A **statement** is defined as a triple, consisting an **subject** s, **predicate** p and **object** o, written as $p(s, o)$. This means that an subject s has an predicate (or property) p with value o.

RDF is based on a graph data model. A **RDF graph** $G = (V, E)$ is a representation of the document triples. Furthermore, a subgraph of a RDF graph is a subset of the triples of the document. A node $n \in V$ could be a subject s or object o, which is connected through a directed arc $(s, o) \in E$ that represents the predicate p. Figure 4.2 shows a graph that provides different information about "South Beach" such as "title" and "description".
The vocabulary of RDF is based on URI references. A URI reference of a node (resource) identifies the thing the node represents (e.g. "http://www.example.org/place01" represents "South Beach"). Resources have to be not necessarily an existing network address, they can be abstract. URI references are also used in predicates to specify relationships between nodes (e.g. "http://www.example.org/description" represents the property "description"). Besides URI references, the graph could also contain different types of values such as integers, dates, strings etc.. Values are represented by plain (e.g."South Beach") or typed literals. The data types of typed literals are related to XML Schema data types (e.g. "23"^^<http://www.w3.org/2001/XMLSchema#integer>).
It is also possible to insert blank nodes into the graph. Blank nodes have no shared meaning or name. They are simply used to denote a unique node in the graph [KC], which is not specified in more detail.
The whole RDF graph is derived by a logical conjunction of the statements. Two different sets of statements can be merged by the union of the triples[1]. In fact, this enables in general also a simple merge of statements that were derived from disparate data sources and that utilize terms of a shared vocabulary.
RDF statements can be reified. Reification allows to insert statements as values of subjects or objects, to build nested or chained graphs that enable, for instance, the specification of doubt or support for statements [SH05, p.124].

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://www.example.org/">
  <rdf:Description rdf:about="http://www.example.org/place01">
    <rdf:type rdf:resource="http://www.example.org/Beach"/>
    <ex:title>South Beach</ex:title>
    <ex:description>South Beach is one of the popular areas of Miami Beach</
        ex:description>
  </rdf:Description>
</rdf:RDF>
```

Listing 4.3: RDF/XML Example

[1] If equal blank nodes are in the sets of triples, they have to be renamed, since they are only unique local to the specific set.

An RDF graph can be serialized in different XML based formats that enable an easy processing of the document by software agents.
Listing 4.3 provides an example of a serialization of the RDF graph of Figure 4.2. The URI references such as “rdf:Description“ are written as XML QNames (qualified names). This means that they have a prefix that denotes the namespace URI as well as an local name that refers to a qualified element or attribute of the namespace (e.g. ”http://www.w3.org/1999/02/22-rdf-syntax-ns#” and “Description”) .
An important element of RDF is ”rdf:Description“, which captures statements about specific resources. The ”rdf:about“ attribute specifies the URI reference of the resource of the statement. Properties (e.g. ”http://www.example.org/title”) of the specific vertex are added as sub elements of this resource element. It is important to denote that the entities “rdf:Description” and “ex:description” have completely different meanings. The first refers to a description about a specific resource, whereas the second denotes a property of this resource.
The abbreviated form of the XML serialization directly incorporates the resource type (e.g. “http://www.example.org/Beach”). Listing 4.4 provides an example of the abbreviated XML serialization.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex:="http://example.org/">
  <ex:Beach rdf:about="http://www.example.org/place01">
    <ex:description>South Beach is one of the popular areas of Miami Beach</
        dc:description>
    <ex:title>South Beach</dc:title>
  </ex:Beach>
</rdf:RDF>
```

Listing 4.4: RDF/XML Example - Abbreviated Notation

As described above, a RDF graph is based on a set of subject-predicate-object triples. The direct serialization of the triples is shown in Listing 4.5.

```
<http://www.example.org/places/01>
  <http://www.example.org/description>
    "South Beach is one of the popular areas of Miami Beach" .
<http://www.example.org/place01>
  <http://www.example.org/title>
    "South Beach" .
<http://www.example.org/place01>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.example.org/Beach>
```

Listing 4.5: RDF/XML Example - Triples Notation

The previous explanations have investigated how to formulate formal statements about information. Mashup data should be represented by such statements to allow an automatic processing and inferences by mashup agents.
However, RDF provides no means to define the terms of the vocabulary (e.g. “http://www.example.org/Beach“) that is used throughout the statements. This is addressed by RDF Schema (RDF-S) [BR] and the Web Ontology Language (OWL) [BvHH+].

4.1.2 RDF Schema (RDF-S)

RDF-S [BR] is a minimal ontological language. It has capabilities to define classes (e.g. “http://www.example.org/Beach“) and properties (e.g. ”http://www.example.org/title”), and enables the specification of how they should be used together. Instances of a class are referenced to its class through the “rdf:type” definition, which has already been used in Listing 4.3. Furthermore, classes could be arranged in a hierarchy through the subclass relationship “rdfs:subClassOf“, which is transitive. This means that A is also

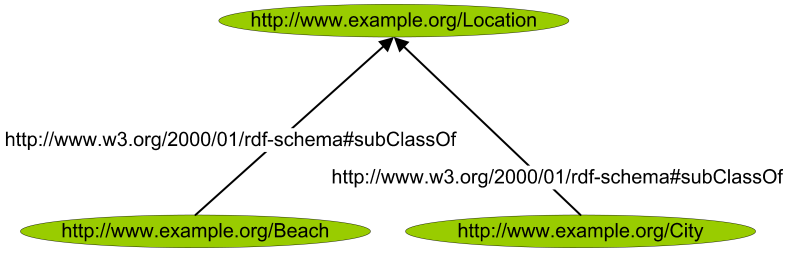

Figure 4.3: RDF-S Hierarchy Example

a subclass of C, if class A is a subclass of B and B is a subclass of C. This holds similar for the instances, which means that an instance i of class A is also an instance of type C. The reader should note that this is very similar to the object-oriented understanding of inheritance. However, RDF-S enables also the separate specification and inheritance of properties through the definition of sub properties and thus differs from the object-oriented approach.
A property has a specific domain and range. The range value specifies that instances of this property are values of a specific class or type (e.g. ”xsd:integer“). The domain relation determines that the property applies to a specific class [BR].
Listing 4.6 is based on the serialization of the graph of Figure 4.3 that specifies ”Location“, ”City“ and ”Beach“ as classes. ”City“ and ”Beach“ are subclasses of the class ”Location“. Furthermore, it specifies the properties for title and description of locations.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.org/">

<rdf:Description rdf:ID="Location">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="Beach">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Location"/>
</rdf:Description>

<rdf:Description rdf:ID="City">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Location"/>
</rdf:Description>

<rdf:Property rdf:ID="title">
  <rdfs:domain rdf:resource="#Location"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="description">
  <rdfs:domain rdf:resource="#Location"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
</rdf:RDF>
```

Listing 4.6: RDF-S Example

The previous investigation showed how to achieve a simple shared vocabulary. Nevertheless, the expressiveness of RDF-S is limited. Amongst others things, it provides no support for **cardinality constraints on properties**, **transitive properties** as well as **equivalence and disjointness** relationships of classes and individuals. The Web Ontology Language (OWL) [BvHH+] has more facilities to express an ontology and is considered below in more detail.

4.1.3 Web Ontology Language (OWL)

The Web Ontology Language (OWL) [BvHH+] [MvH04] is build on top of RDF and RDF-S. OWL provides the three sub languages **OWL-Lite**, **OWL-DL** and **OWL-Full**. The usage of a language depends on the needed expressiveness of the ontology.
OWL-Lite is a light-weight approach that is not as complex as the other ones and enables a quick migration of existing conceptualizations as well as a simple development of tools. OWL-Lite supports class and property equivalence relationships. Furthermore, it is possible to denote two individuals as equal. In addition, properties could be characterised as "ObjectProperty", "DatatypeProperty" as well as transitive, symmetric or functional property [BvHH+]. Object properties reference from an instance of a class to an instance of another class, whereas data type properties are related to an instance of a data type. Functional properties denote that a property could not have more than one value for a specific individual.
OWL-DL supports high expressiveness while retaining decidability. This means that in contrast to OWL-Full, the inference processing time is guaranteed to be finite. Nevertheless, OWL-DL contains all OWL language constructs. To achieve the decidability, the language constructs have to be used under specific restrictions. For instance, a class can not be an instance of another class. Furthermore, OWL-DL enables the specification of disjointness of classes and allows the creation of cardinality restrictions for non-negative integers (instead OWL-Lite allows only *O* or 1 for cardinality restrictions). Listing 4.7 provides an example of a shared vocabulary based on OWL-DL. It defines the three OWL classes: "Location","City" and "Beach". Cities and beaches are special locations. Furthermore, a city is denoted to be disjoint from a beach.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.example.org/ontology.owl#"
    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#">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="City">
    <owl:disjointWith>
      <owl:Class rdf:ID="Beach"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Location"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:Class rdf:about="#Beach">
  <rdfs:subClassOf rdf:resource="#Location"/>
  <owl:disjointWith rdf:resource="#City"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="locatedIn">
  <rdfs:range rdf:resource="#Location"/>
  <rdfs:domain rdf:resource="#Location"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="description">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Location"/>
</owl:DatatypeProperty>

<owl:FunctionalProperty rdf:ID="title">
  <rdfs:domain rdf:resource="#Location"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>

<Beach rdf:ID="place01">
  <title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    South Beach</title>
  <description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    South Beach is one of the popular areas of Miami Beach </description>
</Beach>
</rdf:RDF>
```

Listing 4.7: OWL-DL Example

The ontology defines data type properties for the title and description of locations. Furthermore, the title is a functional property that could have only one value per instance. In addition, the example defines an object property ”locatedIn“ for locations. The range and domain of this object property are instances of type ”Location“. Thus, locations could be located in other locations. For instance, a beach could be located in a city.
Besides the formal definition of classes and properties, the ontology defines an instance of the class ”Beach“ with the identifier ”place01“. The instance is described by the inherited properties ”title“ and ”description“.

In the previous parts of this section, the representation of knowledge on the Web has been described. Such a formal representation is needed to enable inference by software agents. The required semantical expressiveness determines if a simple vocabulary such

Table 4.1: SPARQL Result

x	http://www.example.org/ontology.owl#place01
desc	South Beach is one of the popular areas of Miami Beach

as RDF-S is sufficient or if more expressive approaches such as OWL-Lite and OWL-DL should be used to represent knowledge.

4.1.4 SPARQL Query Language for RDF

SPARQL [PS] is a query language for RDF that supports different query types such as *SELECT*, *CONSTRUCT*, *ASK* and *DESCRIBE*. A query language for RDF is important to get relevant subsets of triples or to create graphs programmatically . Different query forms are used throughout this thesis and are therefore described shortly.
Typically a SPARQL query defines a basic **graph pattern** that consists of a set of **triple patterns**. Triple patterns are similar to RDF triples. However, they can furthermore contain variables annotated by question marks (e.g. ?x). The SPARQL processor **matches** the triple patterns to a given RDF graph and substitutes the variables by the values of the matching triples. Thus, the SPARQL query may responds multiple solutions, if the graph pattern exists more than once.
Listing 4.8 provides an example of a *SELECT* query, which is similar to a Structured Query Language (SQL) *SELECT* query, except that it operates on triples.
The SPARQL query selects all subjects and descriptions that have the predicates

- "http://www.example.org/ontology.owl#title" with value "South Beach" **and**
- "http://www.example.org/ontology.owl#description".

Instead of variables it is also possible to add numeric values or literals (e.g. "South Beach") to the query to precise it in more detail. The result of the query of Listing 4.8 on the data specified in Listing 4.7 is shown in Table 4.1.

```
PREFIX ex: <http://www.example.org/ontology.owl#>
SELECT ?x ?desc
WHERE
  { ?x ex:title "South Beach" .
    ?x ex:description ?desc . }
```

Listing 4.8: RDF/XML Example

It is further possible to add one or more **optional** graph patterns to the *WHERE* clause as well as filter and order statements [PS]. Blank nodes can be denoted by a "_:" (e.g. "$_:a$").

The *CONSTRUCT* query **creates an RDF graph** that corresponds to the graph pattern described in the *WHERE* clause.
In Listing 4.9 a graph is created for all subjects with title "South Beach“. The resulting graph (Listing 4.10) contains additionally the object property ”locatedIn”, which relates to an instance of the class “City“ identified by ”place02“. Thus, it is possible to create RDF graphs programmatically. Blank nodes could be also part of *CONSTRUCT* queries to define structures.

```
PREFIX ex: <http://www.example.org/ontology.owl#>
CONSTRUCT{
   ?x ex:title "South Beach" .
   ?x ex:locatedIn ex:place02 .

   ex:place02 a ex:City .
   ex:place02 ex:title "Miami" .}
WHERE{?x ex:title "South Beach" .}
```

Listing 4.9: SPARQL CONSTRUCT Query

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

  <rdf:Description rdf:about="http://www.example.org/ontology.owl#place01">
    <ex:title>South Beach</ex:title>
    <ex:locatedIn rdf:resource="http://www.example.org/ontology.owl#place02"/
        >
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/ontology.owl#place02">
    <ex:title>Miami</ex:title>
    <rdf:type rdf:resource="http://www.example.org/ontology.owl#city"/>
  </rdf:Description>

</rdf:RDF>
```

Listing 4.10: CONSTRUCT Query Result

The SPARQL *ASK* query returns a boolean value which is true if a specific graph pattern could be matched. The query of Listing 4.11 asks for a triple with the predicate ”title“ and ”South Beach“ as the corresponding value. The query returns “TRUE” on the data of Listing 4.7.

```
PREFIX ex: <http://www.example.org/ontology.owl#>
ASK {?x ex:title "South Beach" .}
```

Listing 4.11: SPARQL ASK Query

The above explanations investigated how knowledge could be represented. Furthermore, it was explained how to extract data from formal semantic descriptions through the SPARQL query language. The SPARQL language is used throughout the thesis to check if specific graph patterns are fulfilled by the given semantic data and to create RDF graphs programmatically.

4.2 Description Logics (DL)

This section describes the knowledge representation by Description Logics in a formal way. Description Logics address the formal compact and effective representation of information, while allowing reasoning tasks to be performed in a computational effective way [BCM+07]. Thus logic-based semantics as well as the emphasis on reasoning as a central service denote Description Logics.
Reasoning is central to Description Logics: "*reasoning allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base*" [BCM+03, p.43]. Furthermore, DL systems are knowledge representation systems, thus they aim to answer queries in a reasonable time.
The KL-ONE system was one of the earliest frameworks that addressed these features and has been based on the experience of semantic networks. It provided a formal logical basis for the representation of classes (or concepts) as well as relationships (often called roles) between the classes [BCM+07]. These principle are central to the family of formalisms of Description Logic [BCM+07, p.47].
In accordance to the above explanations, the basic syntactic building blocks of Description Logics are **atomic concepts**, **atomic roles**, and **individuals**. The creation of complex concepts could be achieved by a defined set of **constructors** that define the expressiveness of the language [BCM+07, p.49].
Description Logics languages are distinguished by their constructors. The basic language is $\mathcal{AL}$ (attributive language), which is a minimal language that is of practical interest, and which serves as a base for the other languages of this family. It is formed according to the following syntax rule [BCM+07, p.52], whereby C and D are arbitrary

concepts. Furthermore, A is an atomic concept and R is an atomic role (or property).

$$
\begin{array}{rcll}
C, D & \rightarrow & A \mid & \text{(atomic concept)} \\
 & & \top \mid & \text{(universal concept)} \\
 & & \bot \mid & \text{(bottom concept)} \\
 & & \neg A \mid & \text{(atomic negation)} \\
 & & C \sqcap D \mid & \text{(intersection)} \\
 & & \forall R.C \mid & \text{(value restriction)} \\
 & & \exists R.\top \mid & \text{(limited existential quantification)}
\end{array}
$$

$\mathcal{AL}$ could express various statements, for instance, let *City* and *Capital* denote two atomic concepts, then the $\mathcal{AL}$ concept $City \sqcap Capital$ denotes all cities that are a capital, while $City \sqcap \neg Capital$ describes all cities which are not a capital.
Further expressiveness could be achieved through more constructors such as (see [BCM+07, p.52, p.534]):

- $\mathcal{F}$ functional properties
- $\mathcal{U}$ union
- $\mathcal{E}$ full existential quantification of roles
- $\mathcal{C}$ negation of arbitrary concepts
- $\mathcal{N}$ number restrictions (cardinally restrictions)
- $\mathcal{I}$ inverse roles
- $\mathcal{R}$ intersection of roles
- $\mathcal{H}$ role hierarchy
- $\mathcal{O}$ nominals
- $\mathcal{Q}$ qualified number restrictions
- $\mathcal{S}$ abbreviation for $\mathcal{ALC}_{\mathcal{R}+}$ that extends $\mathcal{ALC}$ by transitive roles.

OWL-DL and OWL-Lite have been introduced in the previous section. As described previously, OWL-Lite is based on a reduced set of constructors in contrast to OWL-DL. OWL-DL and OWL-Full have the same set of constructors but differ in the usage of them [BCM+07, p.467]. However, the expressiveness of OWL-Full goes beyond the standard Description Logic framework and is denoted by its undecidability [BCM+07, p.480]. In the context of Description Logics theory, OWL-DL is based on $\mathcal{SHOIN}$ and OWL-Lite is $\mathcal{SHIF}$. The possible use of data type properties is denoted by an additional (D).

In general there is a trade-off between expressiveness and difficulty of reasoning. Investigation of the computational complexity and decidability of Description Logics is therefore an important topic.

4.2.1 Structure of DL-based Systems

The logical structure of a DL knowledge base is based on a so called $TBox$ and a $ABox$ ($KB = \langle TBox, ABox \rangle$).
The $TBox$ contains intensional knowledge and is build through the definition of concepts and properties [BCM+03, p.12]. The relationships among the concepts form a lattice like structure. In other words, the $TBox$ contains the terminology of the vocabulary (e.g. a Woman could be defined as a female person by the definition of $Woman \equiv Person \sqcap Female$).
The $ABox$ contains extensional knowledge and denotes knowledge specific to the individuals (e.g the concept assertion $Female \sqcap Person(ANNA)$ or the role assertion $hasChild(ANNA, JACOB)$). Thus, the ABox contains assertions about the named individuals in terms of the defined vocabulary. Furthermore, the $ABox$ dependence on the current circumstances and is part of constant change. This is not the case to intensional knowledge [BCM+07].

4.2.2 Inference

Knowledge representation systems, which are structured by $ABox$ and $TBox$ go beyond the specification of concepts, roles and assertions. They provide **reasoning services**, which are able to extract implicit knowledge [BCM+07, p.67]. Inference is an important topic of Description Logics. In the following the central inference tasks are listed (see [BCM+07, 68] for a formal definition).

- Satisfiability
- Subsumption
- Equivalence
- Disjointness

Satisfiability checks that an expression (newly defined concept based on other concepts) does not denote the empty concept. This means that the reasoner checks if there is an interpretation that satisfies the axioms of the expression. Satisfiability is a key inference task, because other inference mechanisms could be reduced to the checking of unsatisfiability [BCM+07]. **Subsumption** denotes whether a concept is more general

than another concept. **Equivalence** checks, if two concepts are equivalent. In contrast, **disjointness** checks if two concepts are disjoint.
The individuals of a knowledge base make reasoning more complex and require therefor extensions to the TBox reasoning techniques. The reasoning service has to take into account both TBox and ABox and thus, the reasoning problem becomes more complex. Since reasoning algorithms are not the central topic of this thesis, a detailed desription of different alogrithms such as "structural subsumption algorithms" as well as "tableau-based algorithms" is neglected. The interested reader could refer to Baader et al. for a detailed investigation [BCM+07].

The previous two sections described the semantic representation of knowledge. This is the basis for a suitable representation of the mashup data. For an automatic processing of the data, the framework has to provide a shared vocabulary that is used to describe mashup data. Thus, the mashup framework has to define an ontology.

4.3 Web Services

Several mashup definitions outline the importance of web services to mashups (see Chapter 1). In fact, this thesis proposes a mashup framework based on the invocation of different web services. It is therefore suitable to describe their central principles.
Despite the fact that services and web services could be interpreted differently in different domains, this thesis defines web services as functionality that could be engaged over the Web [SH05].
The central intention of the utilization of web services is the creation of a Service-Oriented Architecture (SOA) that supports loosely coupling, implementation neutrality and flexible configuration of independent components [SH05, p.76]. In this context implementation neutrality means that only the interface of the component is important to the architecture and not its internal implementation.
The utilization of standardised web services enable the interoperability of software that has been turned out by different independent developers. Furthermore, the web service standards enable the creation of general purpose tools that could be used to manage the different artifacts [SH05, p.78].
However, interoperability does not automatically support automated agent processing. In fact, standards such as WSDL provide and defined technical description of web services [CCMW] [CMRW], but they do not define the meanings of the messages that has

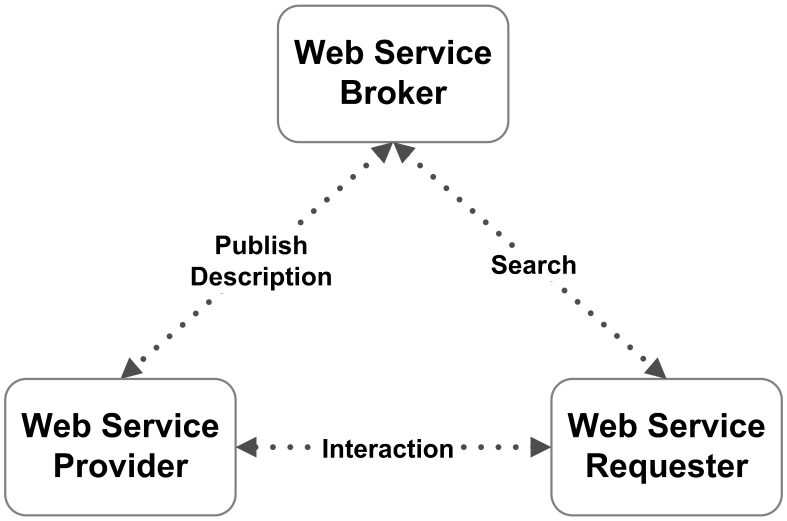

Figure 4.4: The general architecture model for Web Services ([SH05, p.20])

been exchanged or the effects that they do create. This is addressed by the research on Semantic Web Service Description Languages.
Web services could serve as information sources (often called information web services). In this case, they return specific information or content in dependence to the specific request values. However, web services could also create real-world effects. This means that the invocation of a web service could, for instance, perform a ticket ordering or something else. This thesis focusses on the former types of web services. The influences of this assumption are outlined at the relevant points of the subsequent chapters.

4.3.1 General Architecture Model for Web Services

The general architecture of the web service environment is based on a web service provider, a web service requester and a web service broker.
The web service provider creates a description of the implemented web service. The description of the web service is published by a web service broker. Human or software agents, which need a web service, query the broker to find one that is suitable for their needs. The description of the found service is then used to invoke the web service. Figure 4.4 gives an overview about this general model.
In context of the present mashup frameworks, the web service selection is done manually. Most tools provide an internal directory that maintains the available web services. Users could search this directory and add related components (or blocks) for invocation

to their mashup. The interaction between web service provider and requester can be based on a simple request-response schema. Furthermore, the composition of different web services can lead to more complex applications.
As the count of web services is increasing steadily, the manual selection becomes more and more difficult, because the user has to evaluate, if a web service is relevant for his problem. Thus, the directories implement search and filter techniques to reduce the overflow of web services. However, often a standalone web service is not able to fulfill the needs of the agent. In fact, a real productivity gain can be often only achieved through the incorporation of different web service into the mashup application. However, in case of compositions of web services the user has to evaluate how the web service have to be combined to achieve the present goal and at the same time which services should be selected. This problem leads to combinatorial explosion that is difficult to oversee by a human agent.
The aim of this thesis is to overcome this problem by an fully automatic web service selection and composition. However, this requires a service description that is based on an expressive **service description language** and that could be processed and utilized for inference by a software agent.

4.3.2 Considered Types of Web Services

An important type of web services are so called SOAPful web services. The name is derived from the underlying special protocol. The Simple Object Access Protocol (SOAP) is a protocol for exchanging XML messages, which can use, for instance, HTTP, SMTP as its underlying transport protocol. A detailed description of the format of SOAP messages could be found in Singh and Huhns [SH05, p.21-36] as well as in the SOAP specification [SOA].
Also central to mashups are so called RESTful web services. REST means Representational State Transfer and has been proposed by Roy Fielding [Fie00]. The REST architectural style is not a standard, but utilizes existing standards such as HTTP, URI or representations such as XML. REST means that the invocation of a URI responds a representation of a source that transfers the client application into a specific state. Furthermore, like in a state machine, the client application could utilize other URIs of the returned document to get into another state. This is the basic principle that made the Web so popular and it is underlying to many web services[SH05].
In fact, most RESTful services use a defined set of URL parameters to query internal databases and perform the application logic. The HTTP response contains mostly a XML document that denotes the representation of this resource.

The previous explanations outlined shortly the most important types of web services in context of this thesis.

4.3.3 Web Service Description Language (WSDL)

A web service description is central to the service-oriented model, because agents need this description to located suitable services and invoke the service. Therefore, the description of a web service is placed on a directory (the broker) that could be searched by the agent. However, the central question is how a description should look like to achieve interoperability and automated processing by agents.
The Web Service Description Language (WSDL) is a special XML format to describe web services [CCMW] [CMRW]. Version 2.0 has substantial differences in comparison to WSDL 1.1. It supports the description of RESTful web services in a holistic way (WSDL 1.1 only supported GET and POST HTTP requests). However, the current tool and API support for WSDL 2.0 is not mature. Instead, WSDL 1.1 is supported by a variety of tools and is used throughout the industry.
WSDL 1.1 describes services as collections of endpoints (also often called ports). Furthermore, WSDL distinguishes between the abstract definition of endpoints and exchanged messages from their concrete utilization.
The abstract definitions refer to messages and port types. Port types are a collection of operations. The concrete utilization is specified by bindings (e.g. HTTP GET / POST or SOAP 1.1) as well as data format specifications for the exchanged messages [CCMW].
The WSDL document of a web service could be used to automatically configure a web service client. The client is able to interact with the web service, based on the defined message structure. However, WSDL does not define any meanings of the exchanged messages and the created effects of the web services. Therefore, a software agent is not able to automatically select web services based on its semantics.
The introduction of semantic descriptions for web services is addressed by the the research on Semantic Web Services.

4.4 Semantic Web Services

In recent years the research interest in Semantic Web Services (SWS) has increased to a huge research field[KKR08]. Nowadays, web services have gained wide popularity in the mashup community, because they provide an language-independent procedure to invoke functionality over the Web and to retrieve information from different sources.
The existing web service standards provide a good interoperability. However, for an automatic selection of web services as well as the dynamic composition of web services the expressiveness of the existing standards is not enough.
Automatic composition of web services is the central topic of this thesis, because it promises to gather and remix mashup data without user intervention. The semantic description of a web service requires a suitable semantic service description language [KKRK06]. The research community has turned out different standards that are described in the following.

4.4.1 SAWSDL

SAWSDL [FL07] is an **annotation mechanism** that provides the extension of WSDL 1.1 and 2.0 with additional semantics.
SAWSDL is in the status of a W3C recommendation. The approach is based on and closely related to the predecessor WSDL-S [AFM$^+$]. The annotation is achieved through the use of XML extensibility elements. SAWSDL provides two basic types of annotations, the **model reference** and the **schema mapping**.

4.4.1.1 Model References

Model references are used to relate semantic concepts of an ontology to inputs, outputs, operations and XML schema elements etc.. The annotation mechanism itself is independent from the used ontology language of the shared vocabulary.
Listing 4.12 [FL07] annotates an XML schema element, named ”OrderRequest“, by the corresponding concept of an ontology. The schema element is then referenced to an input of a message of the WSDL document, and thus the input is annotated by a semantic concept of an ontology. This annotation provides semantics to the exchanged messages. However, this describes the web service insufficiently. Instead, a "*service should be described by the state change that takes place upon successful execution of the service*" [KKRM05].

Therefore, the model references can be utilized to annotate operations with external representations of the preconditions and effects.

```
...
<xs:element name="OrderRequest"  sawsdl:modelReference="http://www.w3.org
    /2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="customerNo" type="xs:integer" />
      <xs:element name="orderItem" type="item" minOccurs="1" maxOccurs="
          unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
```

Listing 4.12: SAWSDL XML Schema Element Annotation

4.4.1.2 Schema Mappings

Schema mappings are very important, to achieve an effective mediation between the semantic data and specific XML formats of input and output messages of the web service. This achieves data mediation between the agent and the web service.
This mediation is important because a direct alignment of the inputs and outputs of different independent web services is very difficult to achieve.
The often automatically generated technical WSDL descriptions are independent from the agent. This means that inputs and outputs of web services, described by XML schema elements, could be described differently among different web services even if they have equal semantics. The problem is to transform one or more outputs of different web services into the input format of the subsequent web service that should utilize the existing information.
The following examples clarify this problem. The first web service responds a list of persons (Listing 4.13) containing the name and age of the person.

```
<?xml version="1.0">
<output>
 <person>
 <name>
  <first_name>Thomas</first_name>
  <last_name>Fischer</first_name>
 </name>
 <age>24</age>
 </person>
```

```
</output>
```

Listing 4.13: Web Service 1 XML Responded Output

The subsequent web service needs as input another XML format (Listing 4.14), but the same information. For instance, the subsequent web service could respond the address of the person given its name and age.

```
<?xml version="1.0">
<input>
  <forename>Thomas</forename>
  <surname>Fischer</surname>
  <age_of_person>24</age_of_person>
</input>
```

Listing 4.14: Web Service 2 XML Needed Input

A straightforward solution of the problem would be to directly transform the XML document by XSLT. However, this is not a scalable solution for a composition of web services, because a XSLT document has to be created for every possible combination of two or more web services. Furthermore, the inputs of the subsequent service could be based on multiple effects from different predecessor services. This makes a direct transformation very difficult to achieve.
The two types of SAWSDL schema mappings, Lifting-Schema-Mapping and Lowering-Schema-Mapping, overcome this problem. The basic idea is to transform (lift-up) the XML output of a service into a semantic description through XSLT, which is called the Lifting-Schema-Mapping.
Subsequent web services will need other XML formats as inputs. Therefore the semantic data is lowered back to a XML format that could serve as an input of the subsequent web service. The lowering is achieved through the use of SPARQL and XSLT. SPARQL is used to query the semantic data, to retrieve the required triples of the subsequent web service. The result of the query is then transformed by XLST in accordance to the XML schema of the input of the subsequent web service.
In fact, the shared vocabulary provides the means to create mappings for each web service **independently** from the other ones as well as the possibility to **merge** automatically the triple data of different outputs of web services.

Listing 4.15, shows an example Lifting XSLT for the output of web service one (Listing 4.13), which results in the semantic data of Listing 4.16.

```
<?xml version='1.0' ?>
<xsl:transform version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:hash="xalan://exmaple.org/hasher"
  extension-element-prefixes="hash"
  xmlns:ex="http://example.org/ontology#"
  <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes" /
     >
  <xsl:template match="/">
    <rdf:RDF>
    <xsl:for-each select="//person">
      <ex:Person rdf:about="http://example.org/person/{\$companyNameHash}">
        <ex:firstname><xsl:value-of select=".name/first_name"/></ex:firstname
           >
        <ex:lastname><xsl:value-of select=".name/last_name"/></lastname>
        <ex:age><xsl:value-of select="age"/></ex_age>
      </ex:Person>
    </xsl:for-each>
    </rdf:RDF>
  </xsl:template>
</xsl:transform>
```

Listing 4.15: Web Service 1 LiftingSchemaMapping

The lifted data is an individual that is an instance of the class person described in the domain ontology.

```
<rdf:RDF>
  <ex:Person rdf:about="http://example.org/person/123121">
    <ex:firstname>Thomas</ex:firstname>
    <ex:lastname>Fischer</lastname>
    <ex:age>24</ex_age>
   </ex:Person>
</rdf:RDF>
```

Listing 4.16: Web Service 1 Output RDF

For the invocation of the subsequent service the data has to be lowered. The schema mapping of Listing 4.17 lowers the semantic data to the required XML format. At first, the agent uses the predefined SPARQL query to retrieve the required information. In the example the agent retrieves all individuals that are a person and that are specified by their name and age. In fact, this is the information, which is required by the subse-

quent service. The results of the SPARQL query are transformed by an XSLT template to the XML input format of the web service. The result is equal to the required input of web service two (Listing 4.14) and could therefore serve as input of web service two.

```
<?xml version="1.0" encoding="UTF-8"?>
<lowering xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <sparqlQuery>
    PREFIX ex: &lt;http://example.org/ontology#&gt;
    SELECT ?first ?last ?age
    WHERE {
      ?y rdf:type ex:Person.
      ?y ex:firstname ?first.
      ?y ex:lastname ?last.
      ?y ex:age ?age.
    }
  </sparqlQuery>
  <transform>
  <![CDATA[
<xsl:transform version="2.0"
    xmlns:minerva="http://127.0.0.1/ontology/minerva-portals#"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:sp="http://www.w3.org/2005/sparql-results#">
    <xsl:output method="xml" version="1.0" encoding="UTF-8"
      indent="yes" />
    <xsl:template match="/sp:sparql">
        <?xml version="1.0">
        <xsl:for-each select="sp:results/sp:result">
        <input>
        <forename>
           <xsl:value-of select="sp:binding[@name='first']/sp:literal" />
        </forename>
        <surname>
           <xsl:value-of select="sp:binding[@name='last']/sp:literal" />
        </surname>
        <age>
           <xsl:value-of select="sp:binding[@name='age']/sp:literal" />
        </age>
        </input>
        </xsl:for-each>
    </xsl:template>
</xsl:transform>
]]>
</transform>
</lowering>
```

Listing 4.17: web Service 2 LoweringSchemaMapping

The advantage of the specification of such mappings is that for each web service (not for each connection) a lifting and lowering schema mapping could be created that depends only on the specific web service. This ensures that each web service is developed independently from the structure and description of other services. However, it is assumed that all mappings use terms of the shared vocabulary.

4.4.2 SA-REST

Lathem et al. propose the manual creation of mashups from RESTful services [LGS07], which are described by SA-REST, a novel semantic service description language. The SA-REST language is independent from the concepts of mashups, but it could be leveraged to create them. SA-REST is build on the ideas of SAWSDL and has similar semantic support. This means that it uses model references to annotate technical descriptions with service inputs, outputs, operations and faults [LGS07]. The annotations of SA-REST (and SAWSDL) reference to classes of ontologies, to describe the service in a semantic way. Neither SA-REST nor SAWSDL restrict to a specific type of ontology language.
SA-REST incorporates also the concept of lifting and lowering schema mappings to achieve data mediation. This means that the inputs of the RESTful service are derived from SPARQL queries on semantic data and corresponding XSLT, which is called the lowering. The outputs of the services are lifted to a semantic representation through XSLT [LGS07].
The annotation technique is the key point of SA-REST that differs from SAWSDL. Lathem et al. propose to embed the annotation into RDFa [BAB]. RDFa uses XHTML [BIM$^+$] extensions to annotate XHTML pages with semantics. Thus the web service annotation of SA-REST could be added directly to the XHTML page that describes the services to combine the machine and human readable service descriptions. This means that the subjects, predicates are directly added to appropriate XHTML elements (e.g. divElement).
Furthermore it is possible to use GRDDL [Con] to add annotations to the HTML page. GRDDL simply transforms the annotated document through an predefined XSLT into another format, which could be RDF or any other format.

Listing 4.18 [LGS07] provides an example of an GRDDL annotated web page.

```
<html xmlns:sarest="http://lsdis.cs.uga.edu/SAREST#">
...
<meta about=" http://craigslist.org/search/">
<meta property="sarest:input"
  content="http://lsdis.cs.uga.edu/ont.owl#Location_Query"/>
<meta property="sarest:output"
  content="http://lsdis.cs.uga.edu/ont.owl#Location"/>
<meta property="sarest:action" content="HTTP GET"/>
<meta property="sarest:lifting"
  content="http://craigslist.org/api/lifting.xsl"/>
<meta property="sarest:lowering"
  content="http://craigslist.org/api/lowering.xsl"/>
<meta property="sarest:operation"
  content="http://lsdis.cs.uga.edu/ont.owl#Location_Search"/>
</meta>
...
</html>
```

Listing 4.18: SA-REST Service Description

Independently from the used annotation technique, SA-REST provides a means to add service descriptions of RESTful services **directly** to the HTML page, which avoids the separation and maintenance of a technical and a human readable description document.
Lathem et al. state that it is possible to transform any SA-REST description into SAWSDL: "*Due to the fact that SA-REST is a derivative of SAWSDL it follows that a SAWSDL can be generated from an SA-REST annotated page*" [LGS07]. This implicates that RESTful services could be described in WSDL which are annotated by SAWSDL. In fact this was already explained earlier. However, the authors also state that tools based on WSDL are not able to cover RESTful services [LGS07]. This is contradicting to the explanations above. In fact, WSDL 1.1 allows only HTML GET and POST operations. In addition, WSDL 2.0 provides HTTP PUT and DELETE operations. Due to this, for many RESTful information web services WSDL 1.1 seems to be powerful enough to describe them.
Any enterprise mashup framework has also to consider SOAPful services in order to utilize strategic investments in Service-Oriented-Architectures (SOA) and standardization made by companies. The description of SA-REST is intended to support only RESTful services.

The above analysis pointed out that description and semantic annotation of RESTful services based on SA-REST. However, real advantages in comparison to SAWSDL could

not be figured out. In fact, SA-REST has the disadvantage that it does not support SOAPful web services. In context of this thesis, RESTful and SOAPful web services should be used for web service composition. Therefore SA-REST as semantic description approach seems to be not suitable.

4.4.3 OWL-S

The Web Ontology Language for Web Services (OWL-S) [MBH+04] is an upper ontology for web services that contains statements about the web **service profile**, web **service model** and web **service grounding**.
The service profile is intended to support the selection of a web service and describes inputs, outputs, preconditions and effects of a web service.
The web service model describes how the web service could be used by a client. It could be be used to make a more in-depth analysis for the selection of a web service or it could support the composition of web services.
[MBH+04]. Listing 4.19 shows an example of a description of the web service profile. The considered web services returns addresses for a given company name. The inputs and outputs of the described in th web service profile reference to the process of the OWL-S description.

```
...
<profile:Profile rdf:ID="COMPANY_ADDRESS_PROFILE">
 <service:isPresentedBy rdf:resource="#COMPANY_ADDRESS_SERVICE"/>
 <profile:serviceName xml:lang="en">
  Company Address Service
 </profile:serviceName>
 <profile:textDescription xml:lang="en">
  This service provides company address information.
 </profile:textDescription>

 <profile:hasInput  rdf:resource="#_companyName"/>
 <profile:hasOutput rdf:resource="#_name"/>
 <profile:hasOutput rdf:resource="#_street"/>
 <profile:hasOutput rdf:resource="#_city"/>
 <profile:hasOutput rdf:resource="#_state"/>
 <profile:hasOutput rdf:resource="#_zip"/>
 <profile:hasOutput rdf:resource="#_country"/>
 <profile:has_process rdf:resource="#COMPANY_ADDRESS_PROCESS" />
</profile:Profile>
...
```

Listing 4.19: OWL-S Service Profile Example

The **OWL-S model** of a web service is described by process descriptions. A process in OWL-S is defined by its inputs, outputs, preconditions and results. The preconditions and results has to be described by a logic expression. Amongst others the specification supports KIF, PDDL, SWRL and since OWL-S 1.2 [MBD+] also SPARQL as logical languages.
OWL-S defines **atomic**, **simple** and **composite** processes. An atomic process corresponds to an action that is performed in a single interaction. This means that there are no subprocesses and that the process could be directly invoked, if the preconditions are fulfilled. An atomic process is referenced to a **grounding** that specifies how to construct the messages. Simple processes are similar to atomic processes, however there is no reference to a grounding. They are used as special views of atomic or on top of composite processes.
A composite process refers to multiple interactions and could be based on other composite or non-composite processes. The decomposition in it subprocesses is described by control constructs such as ”If-Then-Else“, ”Split“, ”Join”, “Repeat-While”, “Repeat-Until“. The control constructs are similar to a programming language. However it is important to note that the process model could not be directly executed, it is simply a description of how the client could achieve the intended behaviour [MBH+04]. Listing 4.20 shows the process model of the atomic company address service.

```
...
<process:ProcessModel rdf:ID="COMPANY_ADDRESS_PROCESS_MODEL">
<service:describes rdf:resource="#COMPANY_ADDRESS_SERVICE"/>
<process:hasProcess rdf:resource="#COMPANY_ADDRESS_PROCESS"/>
</process:ProcessModel>

<process:AtomicProcess rdf:ID="COMPANY_ADDRESS_PROCESS">

<process:hasInput  rdf:resource="#_companyName"/>

<process:hasOutput rdf:resource="#_name"/>
<process:hasOutput rdf:resource="#_city"/>
<process:hasOutput rdf:resource="#_street"/>
<process:hasOutput rdf:resource="#_state"/>
<process:hasOutput rdf:resource="#_zip"/>
<process:hasOutput rdf:resource="#_country"/>

</process:AtomicProcess>
...
```

Listing 4.20: OWL-S Process Model and Atomic Process

Listing 4.21 shows exemplarily the description of an input and a output.

```
...
<!—Inputs—>
<process:Input rdf:ID="_companyName">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >
http://proton.semanticweb.org/2005/04/protonu#Company
</process:parameterType>
</process:Input>

<!—Outputs—>

<process:Output  rdf:ID="_city">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >
  http://proton.semanticweb.org/2005/04/protonu#City
</process:parameterType>
</process:Output>
...
```

Listing 4.21: OWL-S Definition of Inputs and Outputs

OWL-S supports the specification of preconditions and results. Listing 4.22 expresses the preconditions of the company address service in terms of its required triples (SPARQL *ASK* query). The result is a SPARQL *CONSTRUCT* query that "simulates" the resulting triple structure. The resulting graph contains blank nodes of the created types of information as well as data and object properties. This specifies clearly the type of individuals that are created by the information web service as well as their relations between them.

```
<process:hasPrecondition>
   <expr:SPARQL-Condition>
     PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt;

     PREFIX pup: &lt;http://proton.semanticweb.org/2005/04/protonu#&gt;
     PREFIX psys: &lt;http://proton.semanticweb.org/2005/04/protons#&gt;
     PREFIX ptop: &lt;http://proton.semanticweb.org/2005/04/protont#&gt;
     PREFIX minerva: &lt;http://127.0.0.1/ontology/minerva-portals#&gt;
     ASK  { ?x rdf:type pup:Company ;
               psys:mainLabel ?y .}
   </expr:SPARQL-Condition>

</process:hasPrecondition>

<process:hasResult>
```

```
<process:Result rdf:ID="CompanyAddressResult">
 <process:hasEffect>
  <expr:SPARQL-Expression>
  PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt;
  PREFIX pup: &lt;http://proton.semanticweb.org/2005/04/protonu#&gt;
  PREFIX psys: &lt;http://proton.semanticweb.org/2005/04/protons#&gt;
  PREFIX ptop: &lt;http://proton.semanticweb.org/2005/04/protont#&gt;
  PREFIX minerva: &lt;http://127.0.0.1/ontology/minerva-portals#&gt;
  CONSTRUCT   {   ?x rdf:type pup:Company ;
                     ptop:locatedIn _:a .

                  _:a rdf:type pup:PostalAddress ;
                      psys:mainLabel "" ;
                      ptop:locatedIn _:c ;
                      ptop:locatedIn _:s ;
                      ptop:locatedIn _:l .

                  _:c rdf:type pup:City ;
                      psys:mainLabel "" .

                  _:s rdf:type pup:Street ;
                      psys:mainLabel "" .

                  _:l rdf:type pup:Country ;
                      psys:mainLabel "" .
              }
  WHERE       {    ?x rdf:type pup:Company ;
                      psys:mainLabel ?y .
              }
  </expr:SPARQL-Expression>
 </process:hasEffect>
</process:Result>
</process:hasResult>
...
```

Listing 4.22: OWL-S Definition of Preconditions and Effects

The **grounding of OWL-S** specifies the concrete realization of a web service. The atomic processes of OWL-S are mapped to WSDL operations. Furthermore, the inputs and outputs of an atomic service are mapped to messages in WSDL specific message mappings as well as transformations. Furthermore, the atomic processes could be mapped to related operations of the WSDL file [MBD+].

4.4.4 WSMO

The Web Service Modeling Ontology (WSMO) is a comprehensive framework that aims to automate (totally or partially) the discovery, selection, composition, mediation, execution and monitoring of web services.
WSMO is based on the Web Service Modeling Framework [FB02] (WSMF) and consists of different main elements such as **ontologies**, **goal repositories**, **web services** and **mediators**.
"*Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects...*"[dBBD+05]. Due to this, ontologies are also used to describe the terminology of the web services.
A goal represents a user desire that should be achieved through the invocation of web services. Therefore, a goal specifies wanted capabilities that the human agent is interested in.
A key objective of WSMO is a strict decoupling of the different resources. However, this leads to heterogeneities in data, ontologies, protocols and processes. Therefore mediators are one of the key elements of WSMO [dBBD+05]. WSMO provides four different types of mediators: mediators that link two goals (ggMediator), mediators that resolve mismatches between ontologies (ooMediator), mediators that link web services to goals (wgMediator) as well as mediators between web services (wwMediator)[dBBD+05].

4.4.4.1 Ontologies in WSMO

Ontologies contain various **concepts**,which are the basic elements of the shared conceptualization. WSMO enables the creation of concept hierarchies through the specification of super and sub concepts. Furthermore a concept could be specified by several attributes. The attributes are filled by values at the instance level of the concept [dBBD+05].

```
Class ontology
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type ooMediator
      hasConcept type concept
      hasRelation type relation
      hasFunction type function
      hasInstance type instance
      hasAxiom type axiom
```

Listing 4.23: WSMO Ontology Defintion

To achieve an expressive ontology, **relations** between concepts are also part of WSMO. Similar to the concepts, the relations could be arranged in an hierarchy by the definition of sub relations. Each relation could be specified by non-functional properties as well as parameters.
WSMO further allows the definition of functions. Functions are special relations that are based on multiple domains with a unary range. The value of the range depends on the domain value.
In addition, WSMO enables the specification of instances for relations and concepts as well as axioms that represent logical expressions [dBBD+05]. WSMO is based on WSML [ST08], which is an ontology language. Like OWL, WSML has also different sub languages that differ in their expressiveness. WSML-Core is the basic ontological language. WSML-DL is a description logic similar expressive as OWL-DL [ST08]. The sub languages WSML-Flight and WSML-Rule are directed towards logical programming. WSML-Full unifies WSDML-DL and WSML-Rule by First-Order Logic [ST08]. It is also possible to import different ontologies in WSML. Especially, also OWL-DL and RDF-S ontologies could be imported [ST08].

4.4.4.2 Web Service Description

WSMO web service descriptions contain non-functional properties, imported ontologies, mediators, as well as capability and interfaces of the web service.The ontologies are responsible to achieve an shared understanding and meaning of the web service [dBBD+05].

```
Class webService
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type {ooMediator, wwMediator}
      hasCapability type capability multiplicity = single-valued
      hasInterface type interface
```

Listing 4.24: WSMO Web Service Description Defintion

```
Class capability
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type {ooMediator, wgMediator}
      hasSharedVariables type sharedVariables
      hasPrecondition type axiom
      hasAssumption type axiom
      hasPostcondition type axiom
      hasEffect type axiom
```

Listing 4.25: WSMO Capability Defintion

Important to the web service description is the definition of the functionality or capability. As often in WSMO, **capabilities** could be related to non-functional properties, ontologies and mediators. Important is the possibility of the definition of **preconditions**,**assumptions**, **postconditions** and **effects** as well as shared variables between them. This determines under which circumstances the web services could be executed and how it changes the state of the world.
The definition of **interfaces** for the web service is important to specify the interaction schema with the web service (choreography) as well as the orchestration (functionality from other web services).
The choreography specifies the interaction from the client to the web services, which is based on a set of different states that are achieved through several state changes.
The orchestration specifies the use of other web services. Similar to the choreography this description is state based and provides in addition the support of mediators (e.g. mediation between web services).[dBBD+05]
WSMO services could be grounded based on WSMO or also based on SAWSDL. WSMO defines endpoint description properties that reference from a WMSO web service to the appropriate WSDL service. The "withGrounding" property of WSMO is utilized to reference from a interface of WMSO to corresponding part of the WSDL document.

4.4.5 WSMO-Lite

WSMO-Lite has been created to define a light-weight service ontology, which is directly build on the newest W3C standards [VKF]. Besides a light-weight service ontology, WSMO-Lite defines a annotation mechanism for WSDL. Instead to WSMO, it does not define formal user goals and mediators. WSMO uses the WSML language for the description of semantic model. WSMO-Lite allows the use of any ontology language with an RDF syntax (e.g. OWL).
The information model of the service ontology is represented by a domain ontology.

Functional descriptions of web services are described as capabilities. Such capabilities define preconditions that have to hold before the invocation as well as effects that are created through the invocation of the service. Furthermore, WSMO-Lite models non-functional descriptions that are represented by an ontology [VKF].
WSMO-Lite does not specify any **semantic behavioural descriptions**. If such a one is needed, other existing technologies have to be adopted. The annotation mechanism of SAWSDL is utilized by WSMO-Lite, but does not influence the WSMO-Lite service ontology itself. The concrete semantics of preconditions and effects are **not** defined as part of WSMO-Lite. Thus, different logical languages could be used [VKF].

WSMO-Lite is currently a working draft. To the best of the authors knowledge, there are no tools and semantic web service test collections available that support WSMO-Lite and that could be utilized in the thesis.

4.4.6 MicroWSMO

MicroWSMO [KVG08] is a framework that focusses on the description of RESTful services. It defines a microformat for RESTful APIs as well as a service model based on RDFS to express semantics. MicroWSMO is only available as a working draft, therefore it is not considered in more detail. However, independent from the final approach, it would be preferable to be able to utilize SOAPful and RESTful web services.

4.5 Semantic Web Service Composition

Web services are language-independent and allow an easy integration of heterogeneous systems through standards-based interoperability. However, to yield powerful applications, the web services have to be composed, because it is likely that a single web service is often not able to fulfill all user needs.
The task of putting services together to achieve a desired functionality is called web service composition [SH05]. The present mashup frameworks are often based on a manual wiring of a specific flow of web services. The limitations of these frameworks were described in Chapter 3. In contrast, automatic web service composition promises to achieve a real efficiency gain in information gathering through a significantly reduced time for selection and composition of web services.
Web service composition is a kind of a planning problem and involves **search** and **logic inference** of Artificial Intelligence (AI). Planning is the task of the creation of a sequence

of **actions** that achieve a desired **goal** [RN03, p.375]. This means that a planner has to create a sequence of web services that could be invoked by the software agent to achieve its present goal. Actions in AI planning based approaches are specified by **preconditions** and **effects**. The preconditions have to hold before the agent invokes the action. The effects describe how state changes after the execution of the action [RN03, p.379].
In advance to the concrete planning process, the majority of planners transforms the goals, inputs and outputs of web services as well as preconditions and effects in a First-Order-Logic (FOL) problem representation.
The classical planners use the STRIPS language as representation language [RN03, p.377]. STRIPS describes states, goals and actions by first-order logic literals. The STRIPS language has several restrictions, which allows the transformation of the actions into purely propositional literals. This makes the planning problem more simple and efficient, however, for some real domains it is not expressive enough [RN03, p.379]. Therefore, extensions such as the Action Description Language (ADL) have been created.
The planning formalisms have been unified in the Planning Domain Description Language (PDDL), which has sub languages for STRIPS, ADL and Hierarchical Task Networks (HTN) [RN03, p.377] (explained below).

4.5.1 Algorithms

The following planning algorithms are not only specific to web services and thus are used in different planning problems. Planning algorithms could be based on forward or backward **state-space search** [RN03, p.382]. In a forward-based state-space search planning problem (I, A, G), the planner has to find a sequence or chain of actions A that transfer the initial state I into an desired goal state G. Similarly, backward based state-space search starts at a given goal state G and searches for a sequence of actions A that this is based on the initial state I.
However, forward- and backward-chaining have important differences. The first considers irrelevant actions, the approach is often called undirected search [KSKR05], and thus is too inefficient for practical problems [RN03, p.384]. Instead, backward chaining considers only relevant actions. An action is called relevant, if it contributes to the main goal or it contributes to the achievement of a subgoal of a subsequent action. However, Russel and Norvig state that without the introduction of appropriate **heuristics** both approaches are not very efficient [RN03, p.386].
In this context, Lin et al. [LQH+06]propose an example of a DL based planner based on backward-chaining, which utilizes heuristics to prune the search space.
State-space search does not utilize problem decomposability and does not search for

different subgoals independently. Furthermore, it is based on total ordered-plan search [RN03]. This problem is addressed by **Partial-Order Planning** (POP).
Planning graphs could be utilized to achieve better heuristic estimates for total-order and partial-order planners. They are created incrementally, starting from the initial state. The GRAPHPLAN algorithm could directly create a plan from the planning graph [RN03] by backward search.
The SATPLAN algorithm translates the planning problems into propositional logic and applies algorithms for satisfiability.
Another important approach to planning are Hierarchical Task Networks (HTNs). In HTN planning [RN03] the high level plan is decomposed into sub plans, which have to be predefined, until primitive actions are present that could be executed by the agent.

4.5.2 Closed-World Assumption

The existing SWS composition planners (e.g. [LQH+06]) are based on the **closed-world assumption**. This means "*that any conditions that are not mentioned in a state are assumed as false*" [RN03, p.377]. However, from their nature description logics such as OWL are open world. Thus, if something it is not stated it is not assumed to be false. The information in ABoxes of DL is therefore generally assumed to be incomplete [BCM+07, p.75].

4.5.3 Classification of Planning Approaches for Web Service Composition

This thesis focuses on the characterization of existing approaches of web service composition and their current limitations. In accordance to Schumacher et al. [SHS08] planning systems could be classified as follows:

- Static vs. Dynamic Composition
- Functional- vs. Process-Level Composition

While in a **dynamic composition approach** the web services are planned at invocation time, the static composition first generates the plan and then invokes the web services. **Functional-level** composition means that a web service is considered as an atomic entity. This means that the web service is specified by its inputs, outputs, preconditions and effects (IOPE) and requires only a simple request-response interaction. Atomic web services are also those ones, which are provided as a black-box and thus the underlying behaviour and interactions are not visible. Instead, **process-level** composition consid-

ers the internal interactions of a web service [SHS08], which would correspond to a more complex interaction schema.
A detailed and present comparison of a representative set of SWS composition planners could be found in Schumacher et al. [SHS08, p.79-104] and is thus not reflected by this thesis. Moreover, this thesis concentrates on the open problems for SWS composition that has been identified by the research community.

4.5.4 Open Topics of Semantic Web Service Composition

The research literature has pointed out the following open topics:

1. Appropriate incorporation of user preferences in the matching process [KKR04] [SHS08, p.93-94] [LYGW06]
2. Scalability of planning algorithms [KSKR05] [SHS08, p.93-94]
3. Adaptive discovery (adaptive composition) [SHS08, p.93-94]

User preferences are important to both automatic web service selection [KKR04] as well as dynamic web service composition, because the offers of a web service will often differ from the requests. Furthermore, the objectives are manifold and the preferences (importance) of each objective are different among users. Therefore, only when the user preferences are properly described and are part of the composition request, the services composition can be really executed without user's intervention [LYGW06].
Li et al. propose an extended OWL-S model for the incorporation of user preferences. The approach models different user preferences by the definition of rule sets [LYGW06].
König-Ries and Klein [KKR04] propose an approach based on fuzzy sets. The general intention is the measure the degree of membership between a service offer and the fuzzy sets if the service request that represent the requester's preferences. Thus, the membership value represents the matching value.
The scalability of planning algorithms is another open research topic. In this context, scalability means the ability to consider large sets of web services in the planning process. Furthermore, it could focus also on the scalability with respect to length of the plans or the number of users that work with a planning system.
Adaptive discovery means that the agent is adapting the discovery of web services with respect to changed user interest, tasks etc..

The previous explanations described open topics of semantic web service composition. As stated in the requirements preferences, adaptive discovery and adaptive composition as well as scalability are important to this mashup framework, because the mashups should adapt based on the interests, expertise and tasks of a user. Further-

more, the increasing number of available RESTful and SOAPful web services demands scalable mashup solutions. In order to address the current open research topics, limitations of existing mashup frameworks as well as stated requirements, this thesis proposes its own approach to web service composition planning. The approach, which is described in detail in Chapter 6, addresses the scalability of web service composition planning as well as the incorporation of user preferences.

4.6 Multi-Criteria Decision Theory

As pointed out in the previous chapter, the incorporation of user preferences into the planning process is an open research topic. It is therefore important, to provide an sounding framework that is able to weight different alternative web service compositions in dependence of the user preferences. The objectives of a human agent could be very different and conflicting. Thus, a selection of a composition alternative out of a set of different ones is not as straightforward as it might looks.
It is important to note that decision theory does not prescribe how to achieve the different alternatives. Instead, it is intended to give a sounding framework which determines the weighting of a alternative. Economic research has come up with a variety of approaches to handle conflicts of objectives during the evaluation of different alternatives.
Table 4.2 provides different web service composition alternatives, specified by the values of the different objectives. The objectives correspond to the information criteria that has been stated in the requirements analysis (see Chapter 2). The start and goal state of the planning problem are not relevant at this point, because the objective functions are already processed. It could be assumed that a planner turns out such a list of alternatives or that different alternatives have to be weighted during the planning. In fact, the question is, which of the alternatives should be selected, with respect to the user preferences.
The economic research has proposed different approaches to handle such goal conflicts, to weight different alternatives, and to prune the set of alternatives. In this context, an alternative A_i is a vector $\vec{x}_i$ with values x_i^h of the defined objectives $h \in \{CORRECTNESS, COMPLETENESS, \ldots, PRICE\}$.
The following sections explain shortly different methods that could be used to prune the set of available alternatives (Dominance Concepts) as well as to select (Trade-off Models, Multi Attributive Utility Theory (MAUT)) an alternative, while adhering user preferences.

Table 4.2: Web Service Composition Decision Matrix

	Completeness in %	**Correctness in %**	**Plan Length**	**Total Availability**	**Total Accessibility**	**Minimum Integrity**	**Avg. Served Requests**	**Avg. Number of Failures**	**Minimum Compliance**	**Minimum Security**	**Total Price per Request**
	Max	Max	Min	Max	Max	Max	Max	Min	Max	Max	Min
A_1	100	100	5	0.70	0.90	5	13	2	5	3	0.5
A_2	85	100	4	0.90	0.90	5	13	2	5	3	0.1
A_3	100	50	6	0.70	0.90	5	13	2	5	3	0.5
A_4	100	100	5	0.70	0.90	4	26	2	5	4	0.6
A_5	100	100	6	0.90	0.60	5	13	0	0	3	0.1

4.6.1 Dominance Concepts

Dominance concepts offer the possibility to exclude alternatives from the decision problem. The strict dominance determines all alternatives that are **efficient**. An alternative is efficient, if it is not dominated by another alternative.
Or more formally, an alternative A_i is **inefficient** if there is another alternative A_q such that for all objectives $h = 1, \ldots, H$, $x_q^h \succeq x_i^h$ holds true and there exists at least one h such that $x_q^h \succ x_i^h$. In this case, A_q dominates A_i.
In the example decision problem of Table 4.2, A_3 is dominated by A_1. All other alternatives are goal efficient.
The advantage of dominance concepts is the reduction of the set of alternatives. However, if there are a lot of different dimensions, often many alternatives are not dominated. The remaining alternatives can be treated by other multi-attributive methods.

4.6.2 Trade-off Models

4.6.2.1 Linear Combination

It is possible to weight the different goals through a linear combination.

$$f(\vec{x}_i) = \sum_h w_h * x_i^h$$

The value of the objective h at the alternative $\vec{x}_i$ is denoted by x_i^h. The decision maker incorporates his preferences into the weightings $w_h \in [0,1]$ of this linear combination. However, this approach has significant shortcomings. The dimensions of the goals as well as the scales are different. Thus, such a linear combination compares apples and oranges.
For instance, the completeness values (scale $[0,100]$) influence the total value of the linear combination much more than the the values of the total availability $[0,1]$. This distorts the preferences between the objectives that have been incorporated into the weightings w_i.
The economic research recommends to measure the objectives on a unique dimension for all the objectives. Such a measure is the utility and this is considered by the Multi-Attributive Utility Theory (MAUT) explained subsequently.

4.6.2.2 Goal Programming

In goal programming, the decision maker defines a desired target vector $\vec{t}$ with the objective values $t^1, \ldots, t^H$. Or in other words, the decision maker specifies his ideal alternative.
A solution candidate is then evaluated in dependence of the **distance** between candidate and goal vector. The best alternative is the one with the minimum distance to the target vector.

$$min \sum_{i=h}^{k} |x_i^h - t^h|$$

In this context, x_i^h denotes the value of objective h of alternative i.
This approach has several difficulties. First, the decision maker has to know the ideal vector in dependence to the problem. Second, the importance of different objectives is distorted by the difference of the scales and dimensions of the objectives. Third, an appropriate distance measure has to be defined.

4.6.2.3 Lexicographic Ordering

In a **lexicographic ordering** the decision maker creates an ordering of goals with an decreasing importance. The decision is made under successive consideration of the goals in dependence to the ordering. In each loop not optimal alternatives are declined. For instance, the following ordering could be state by the decision maker :

$$COMPLETENESS \succ CORRECTNESS \succ COMPLIANCE \succ \ldots \succ PRICE$$

. In the first iteration all not optimal alternatives concerning the *CORRECTNESS* are declined. For Table 4.2 this would delete alternative A_2. In the second iteration the *CORRECTNESS* is considered. Since alternative A_3 has been deleted by the dominance, there is could be not other alternative declined. This process is executed until all objectives has been considered or only one alternative is remaining.
Lexicographic ordering could be utilized to reduce a set of available alternatives. However, the difficulty of this approach is to find a suitable ordering.

4.6.3 Multi-Attributive Utility Theory (MAUT)

Multi-Attributive Utility Theory describes the preferences for different objectives by values of utility functions. A utility function specifies for each possible value x of an objective the utility $u(x) \in \mathbb{R}$ for the human agent. It is also possible to specify a utility function for non-quantitative, thus qualitative goals.
The creation of a utility function requires an **complete** and **transitive preference order** of the values of the alternatives of a specific goal.
The decision maker has a **preference order** between alternatives A_i and A_j, if he prefers A_i over A_j ($A_i \succ A_j$) or he prefers A_j over A_i ($A_j \succ A_i$) or he is indifferent between A_i and A_j ($A_i \sim A_j$). The preference order is **complete** if there exists such a preference for each pair of alternatives. The preference order is **transitive** if $A_i \succeq A_j$ and $A_j \succeq A_z$ leads to $A_i \succeq A_z$.

The Multi-Attributive Utility Theory (MAUT) specifies the total utility by an additive utility function.

$$U(\vec{x}) = \sum_{h=1}^{H} \lambda_h * u_h(x_h) \text{ with scaling factor } \lambda_h > 0 \text{ and } \sum_h \lambda_h = 1$$

The additive utility function sums the different utility values of the different objectives. Thus the preferences are specified by utility values. Different utility scales are adjusted by so called **scaling factors** λ_h. The scaling factors are **not** comparable with weightings, because a weighting implicates the importance of a specific objective. Instead, in MAUT the preference is provided by the different utility functions of the different objectives. The different scales of the utility functions are balanced by the scaling factors to avoid a distortion of the preferences.
The theoretical backgrounds for the creation of the utility functions and the calculation of the scaling factors is not part of this thesis, because this should be typically inves-

tigated with the modeling of the user interests, tasks and experience. Nevertheless, MAUT is a theoretical exact method for the measurement of utilities, if the prerequisites are fulfilled. A detailed description of the prerequisites of MAUT, the creation of utility functions and the calculation of scaling factors could be found in Klein and Scholl [Sch04].

This section has outlined different approaches that enable the pruning of a set of different alternatives as well as the selection of a "best" alternative. In the context of web service composition, this means the selection of the plan that meets the needs of the agent. For this thesis, those approaches are important that create a preference value for the different alternatives (e.g. simple linear combinations). However, the straightforward usage of the values of the different objectives for the weighing is not appropriate since, the different scales and dimensions distort the relation of the importance of the objectives. Therefore the objectives should be measured by the utility, which could be aggregated by to the total utility of the alternative (web service composition). The total utility could be utilized in the optimization algorithm proposed in this thesis (see Chapter 6) to drive the optimization.

4.7 Conclusion

This chapter has described several theoretical backgrounds that are important to achieve the automatic generation of mashups. It can be concluded that the mashup framework should utilize and ontology to enable automatic processing and inference on data. Furthermore, the web services have to be described by an semantic service description language that covers the description of the capabilities of the web services.
The automatic composition of the different web services refers to AI planning, which has several open topics. Therefore,this thesis proposes its own approach for web service composition. The next chapter explains the resulting architecture of the framework, which is based on the previous investigations.

CHAPTER 5

Mashup Framework Architecture

This chapter outlines the architecture of the mashup framework. The architecture is based on the requirements stated in Chapter 2. Furthermore, the approach addresses the limitations of existing mashup frameworks that were described in Chapter 3. Section 5.1 provides an overview about the architecture. The subsequent sections describe the main modules of the mashup agent.

5.1 Framework Overview

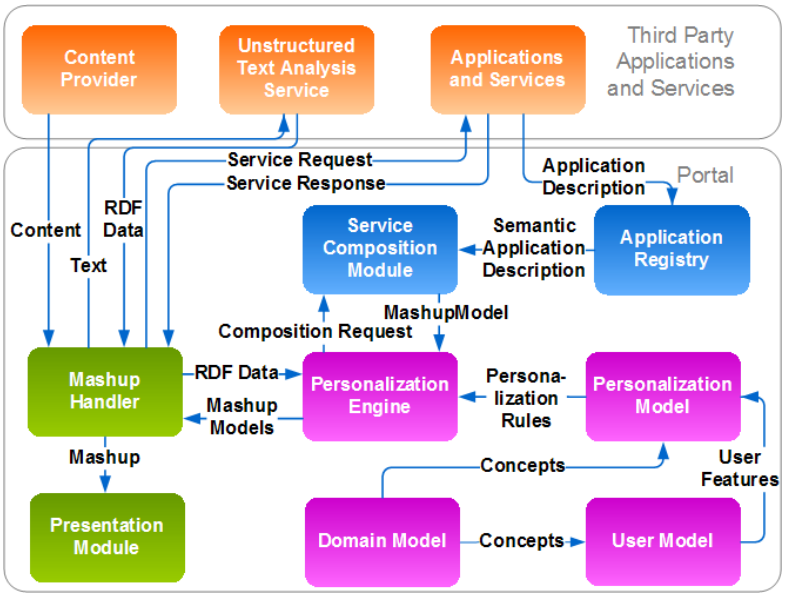

Figure 5.1: Mashup Agent

The mashup framework can be characterized as an agent, which is denoted by a **knowledge base** as well as an **agent program**. The agent program is based on different mod-

ules that enable an user-sensitive automatic **composition** of information web services as well as their **execution** without user interaction. The gathered data is aggregated and returned as a mashup.

The knowledge base $KB = \langle TBox, ABox \rangle$ of the mashup agent is based on Description

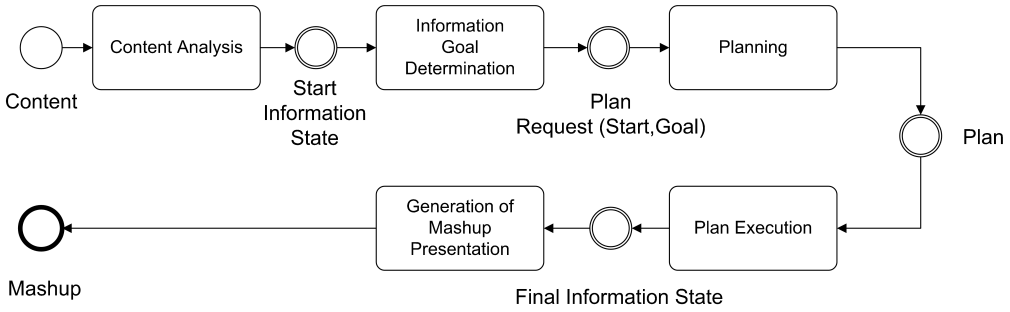

Figure 5.2: General Mashup Process

Logic. The domain model of Figure 5.1 represents the TBox of the agent knowledge base, which is used throughout the framework for reasoning purposes. The ABox of the knowledge base is **dynamic** and represents the mashup data, which is gathered and maintained by the agent program.

Figure 5.2 describes the general process of the agent program. The initial content retrieved from the content provider is just a plain string. The "**Mashup Handler**" transfers the initial content to an external **unstructured text analysis** web service that is able to extract the meanings of the content. The unstructured text analysis service returns an semantic representation of the entities of the content. The semantic representation of the entities is achieved through assertional statements and thus is part of the ABox of the agent knowledge base. At the same time, these statements represent the start information state (start ABox state) of the mashup agent.

Based on the data of the **user model** as well as some **personalization rules**, the **personalization engine** defines a goal state that should be achieved by the agent. The goal state represents the background information that should be presented in the mashup. (The formal representation of the goals is explained in Chapter 6.)

The start and goal state are given to a **planning module for web service composition**. This module creates a composition of web services (a plan) that is able to gather the necessary data to achieve the final goal state. The "**Application Registry**" serves the planner with the web services that have to be considered in the planning process. Furthermore, it maintains the references to the endpoints of the web services, which are located somewhere in the Web.

The generated plan of the "**Service Composition Module**" is then dynamically invoked by the "**Mashup Handler**", which transforms the outputs of the web services into an

semantic representation. The semantic statements utilize the terms of the shared vocabulary of the mashup agent. In fact, the final mashup data is completely represented by assertional statements (ABox), which use the predefined terms of the TBox (see "Domain Model" Figure 5.1).
The mashup data (ABox) is then given to the "**Presentation Module**" that generates an appropriate presentation for the data.

The previous explanations outlined the general mashup process and introduced the responsible modules, which are detailed in subsequent sections.

5.2 Semantic Service Description Language

A semantic service description language is the prerequisite for a **dynamic selection** and **composition** of different web services. Furthermore, the capabilities and expressiveness of the approach influence the development of the planning module. However, also the **execution** of the created plan has to be adhered. **Data mediation** is therefore also a central topic.
The agent utilizes OWL-S and SAWSDL. SAWSDL is used to annotate the schemes of the WSDL files by schema mappings for lifting and lowering of data to achieve the data mediation between the atomic web services (see Chapter 4).
The planning module utilizes OWL-S web service descriptions. The message mappings to WSDL as well as the transformations of OWL-S are not utilized since this is already achieved by the SAWSDL lifting and lowering schema mappings.
The availability of a test collections is one plus of OWL-S. The test collection is specified in Chapter 8. However, OWL-S provides also an expressive ontology that allows the specification of preconditions and results in different logical languages. Thus, the utilization of OWL-S seems to be appropriate to show the feasibility of automatic generation of user sensitive mashups based on dynamic service composition. In contrast to WSMO, OWL-S is supported by an widespread use of the underlying web ontology language (OWL), which provides tools that could be leveraged. In addition, the availability of the test collection was important for the selection of OWL-S. SA-REST has not been selected because it does not support SOAPful web services.
The research community argues that there is only little effort in the comparative evaluation of different SWS description approaches [KKR08]. Therefore, it is not clear for which applications a specific SWS description language is more suitable and this could also not answered in this thesis. Thus, OWL-S has been selected, because it seemed to

provide the best premises.
In general, the strict separation of the execution and the planning module enables the development and testing of other planning approaches that are based on other SWS languages such as DSD, WSMO etc. (see Section 4.4).

5.3 Knowledge Base

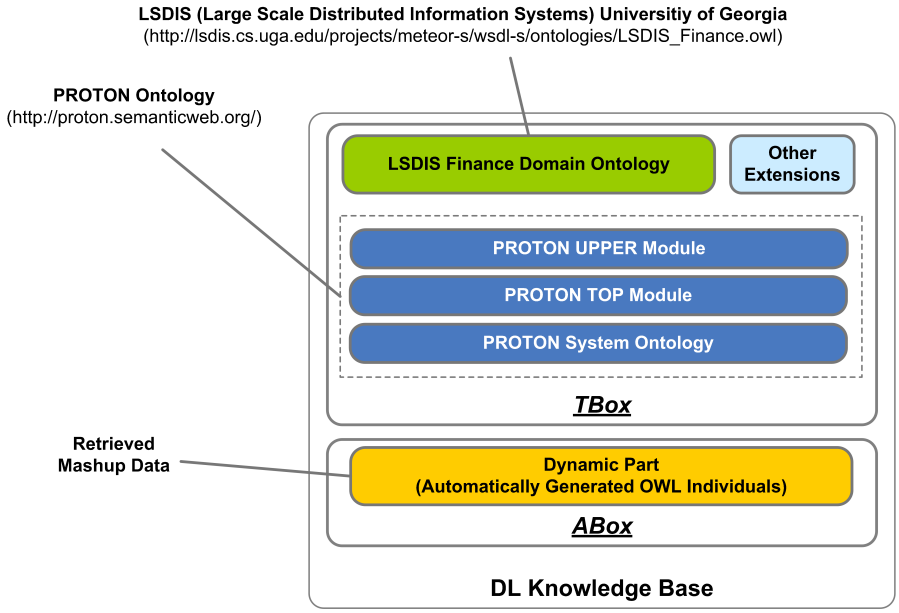

Figure 5.3: The Mashup Agent Knowledge Base

The knowledge base $KB = \langle TBox, ABox \rangle$ is an important aspect of the mashup agent to achieve a uniform understanding of the processed data. The $TBox$ is based on the PROTON upper level ontology[1] as well as the LSDIS Finance Ontology[2].
The PROTON ontology has been developed in the Semantic Knowledge Technologies (SEKT)[3] project. The ontology is represented in OWL-Lite and split into different modules (e.g. system, top, upper, knowledge management). The LSDIS Finance OWL ontology is connected to the upper level ontology to achieve the financial specialization. Furthermore, the $TBox$ is manually extended by some other terms that were needed for the descriptions of the web services.
The ontology engineering approach is similar to the methodology described by Swartout et al. [SRKR97]. The project utilizes a predefined upper level ontology that is extended

[1] `http://proton.semanticweb.org/`
[2] `http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/LSDIS_Finance.owl`
[3] `http://www.sekt-project.com/`

for special purposes and could be pruned as needed.
The agent knowledge base is dynamic for the $ABox$. Therefore, it is empty in the initial state of the agent. The state is changed in case of the **generation of instances** of the ontological classes and properties. Instances are generated at three subprocesses of the architecture: in the extraction of semantic data from the content provider (unstructured text analysis), in the service execution phase (data lifting) or in the planning phase (instance simulation). Details on the instance generation are explained in subsequent parts of the thesis.
Due to the above explanations, the $TBox$ is equal for all instances of the mashup agent, while the $ABox$ depends on the contents of the mashup provider as well as the invoked web services and is therefore maintained separately. Currently, an proactive sharing of these information among different agent instances is not intended, but could be a topic for further research.

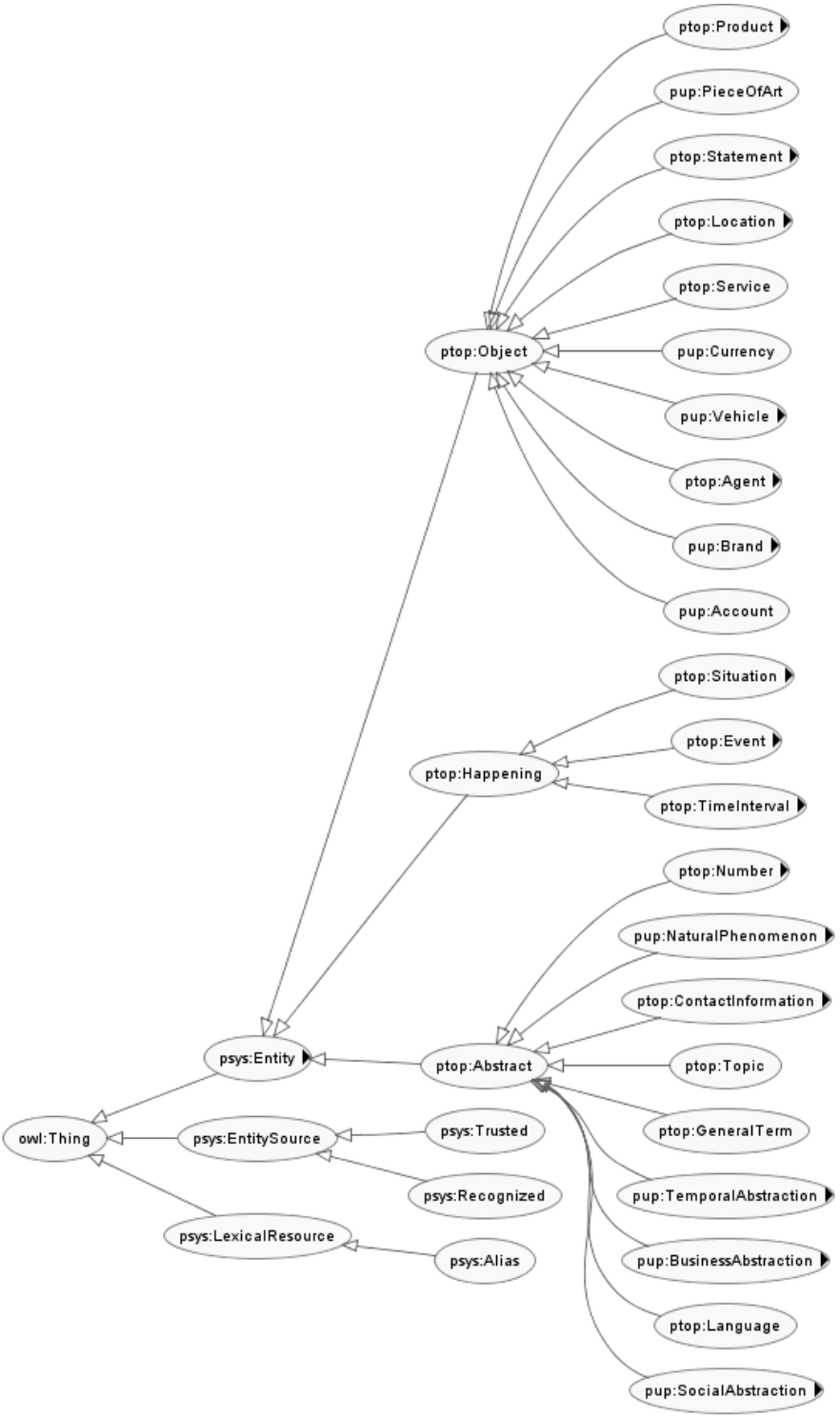

Figure 5.4: Excerpt of Proton Top Level Concepts

5.4 Mashup Handler

The "Mashup Handler" is responsible for the invocation of the external web services. Furthermore, it also invokes the unstructured text analysis web service.

5.4.1 Unstructured Text Analysis

The provision of personalized background information that augment a given content is central to the framework. Unfortunately, most of the content that could be retrieved from the Web or local files and databases is not described semantically. The unstructured text analysis service has to extract and annotate entities of a given text by concepts of a shared vocabulary. For instance, the text "Siemens AG increases earnings" should be annotated by the ontology concept "Company" for the entity "Siemens AG". This means that the component has to identify "Siemens AG" as an instance of the class "Company". In fact, it would be ideal to have the whole content annotated by semantic concepts as well as relations (properties) between the identified entities.
The Apache Unstructured Information Management Architecture (UIMA)[4], which was originally developed by IBM[5], as well as the Calais Web Service[6] by ClearForest, a Thomson Reuters Company, provide unstructured text analysis functionality.

5.4.1.1 Apache UIMA

UIMA is an open source framework that provides a platform for the analysis of unstructured contents such as text, audio or video. The pluggable architecture enables the reuse of newly developed and existing components [Fou08]. Furthermore, the UIMA architecture takes care of the chaining of different analysis components to achieve a full working application that transforms unstructured content to structured ones.
UIMA comes along with a set of sandbox components such as the OpenCalais Annotator, the Simple Server etc.. The OpenCalais Annotator is a component that makes the Calais Web Service (Section 5.4.1.2) available to UIMA. The Simple Server is important, because it allows to provide the UIMA processing capabilities as a RESTful web service[Fou08].
UIMA provides a powerful platform for the development of content analysis engines

[4] http://incubator.apache.org/uima/
[5] http://www.alphaworks.ibm.com/tech/uima
[6] http://www.opencalais.com

Table 5.1: Excerpt of Calais Entities, Events and Facts

Entities	Events/Facts
Anniversary	Acquisition
City	Alliance
Company	AnalystEarningsEstimate
Continent	AnalystRecommendation
Country	Bankruptcy
Currency	BusinessRelation

that could be chained to analyse the content in detail. However, the UIMA framework is not used in the current prototype, because the current analysis capabilities of Calais are enough to show the feasibility of the proposed framework. Nevertheless, UIMA seems to be a suitable platform for unstructured text analysis that should be considered in the future.

5.4.1.2 Calais Web Service

The Calais Web Service creates semantic meta data for any type of textual content. It is currently free for commercial and non-commercial use. The web service uses methods like natural language processing and machine learning to analyze the content and extract entities, facts and events. Table 5.1 provides an excerpt of current identifiable elements.
The Calais Web Service could be called by SOAP 1.1/1.2 or by HTTP POST. Amongst others, the output format could be set to RDF/XML. Furthermore, the web service requires as input the type of content such as “TEXT/RAW”, “TEXT/HTML”, “TEXT/XML” or “TEXT/TXT”. The type of content could be utilized to neglect the HTML tags of the content and escape necessary characters.
The RDF/XML response of Calais provides statements about the found entities, events and facts, which are represented as instances of classes specified in the Calais ontology. Listing 5.1 shows an instance of a company.

```
<rdf:Description rdf:about="http://d.opencalais.com/comphash-1/9dd2192a-4cd2
    -3b9a-ac2f-b6a0d1fed773">
   <rdf:type rdf:resource="http://s.opencalais.com/1/type/em/e/Company"/>
   <c:name>IBM</c:name>
</rdf:Description>
```

Listing 5.1: RDF/XML Example

Currently, the Calais ontology is not public. Naturally, the namespaces and local names **differ from the knowledge base** of the mashup framework. As a result, reasoning about the Calais response fails as long as there is no ontology mediation.
The mediation is achieved in the following way. Classes of the Calais ontology are created manually in the agent domain ontology. Then the classes are set to be **equivalent** (by "owl:equivalentClass") to the corresponding class of the domain ontology.
The main label of each entity of Calais is described by the property "c:name". Therefore, this property has been added to the knowledge base and specified to be equivalent to the related property "psys:mainLabel" of the domain ontology (see Listing 5.2). Thus, the reasoning services of the knowledge base process the instances of the Calais ontology as instances of the domain ontology.
The above explained ontology mediation approach seem to be suitable, because the ontology of Calais is not published yet, and thus could not be directly imported and aligned. Furthermore, the count of entities which could be extracted is "relatively" small and could therefore be added manually to the domain ontology. However, if the ontology of Calais is published a direct alignment should be considered. Listing 5.2 shows a mediation example.

```
<?xml version="1.0"?>
<rdf:RDF>
  <rdf:Description rdf:about="http://s.opencalais.com/1/type/em/e/Company">
    <rdfs:subClassOf rdf:resource="http://proton.semanticweb.org/2005/04/
        protons#Entity"/>
    <owl:equivalentClass rdf:resource="http://proton.semanticweb.org/2005/04/
        protonu#Company"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Calais Company Term</rdfs:comment>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://s.opencalais.com/1/pred/name">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <owl:equivalentProperty rdf:resource="http://proton.semanticweb.org
        /2005/04/protons#mainLabel"/>
    <rdfs:domain rdf:resource="http://proton.semanticweb.org/2005/04/protons#
        Entity"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        calais entity name</rdfs:comment>
  </rdf:Description>
</rdf:RDF>
```

Listing 5.2: Mediation of Calais and Domain Ontology

5.4.2 Web Service Execution

The invocation of a web service is done dynamically through an automatic configuration of the implemented web service client by the given WSDL file. The system supports SOAPful and RESTful web services. The data mediation between the web services is achieved by lifting and lowering schema mappings referenced in the WSDL file (SAWSDL). Figure 5.6 provides an overview about the process of lifting and lowering

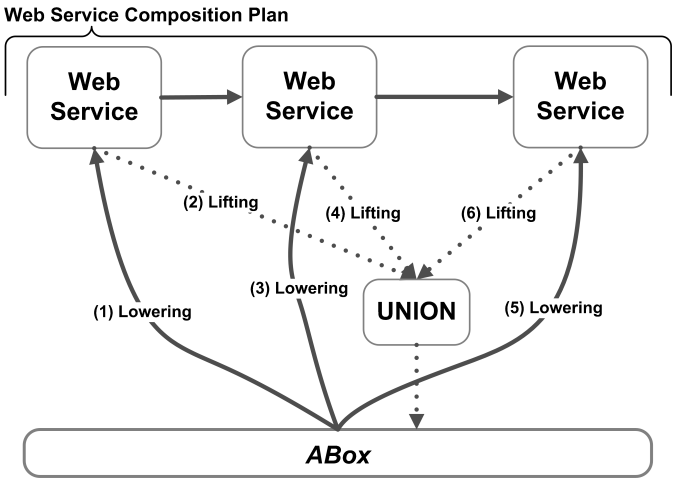

Figure 5.5: Generation of the Assertional Statements of the TBox through Lifted Web Services Outputs

of data (see also Chapter 4). For each web service of the plan the present semantic data is queried by SPARQL to retrieve the inputs of the service. The resulting output data is lifted to achieve assertional statements that could be added to the $ABox$ of the mashup agent knowledge base. The central principle is outlined in Figure 5.5.
The previous explanations have turned out how unstructured text could be annotated by semantic concepts and how dynamic web service execution as well as the generation of the $ABox$ data is achieved.

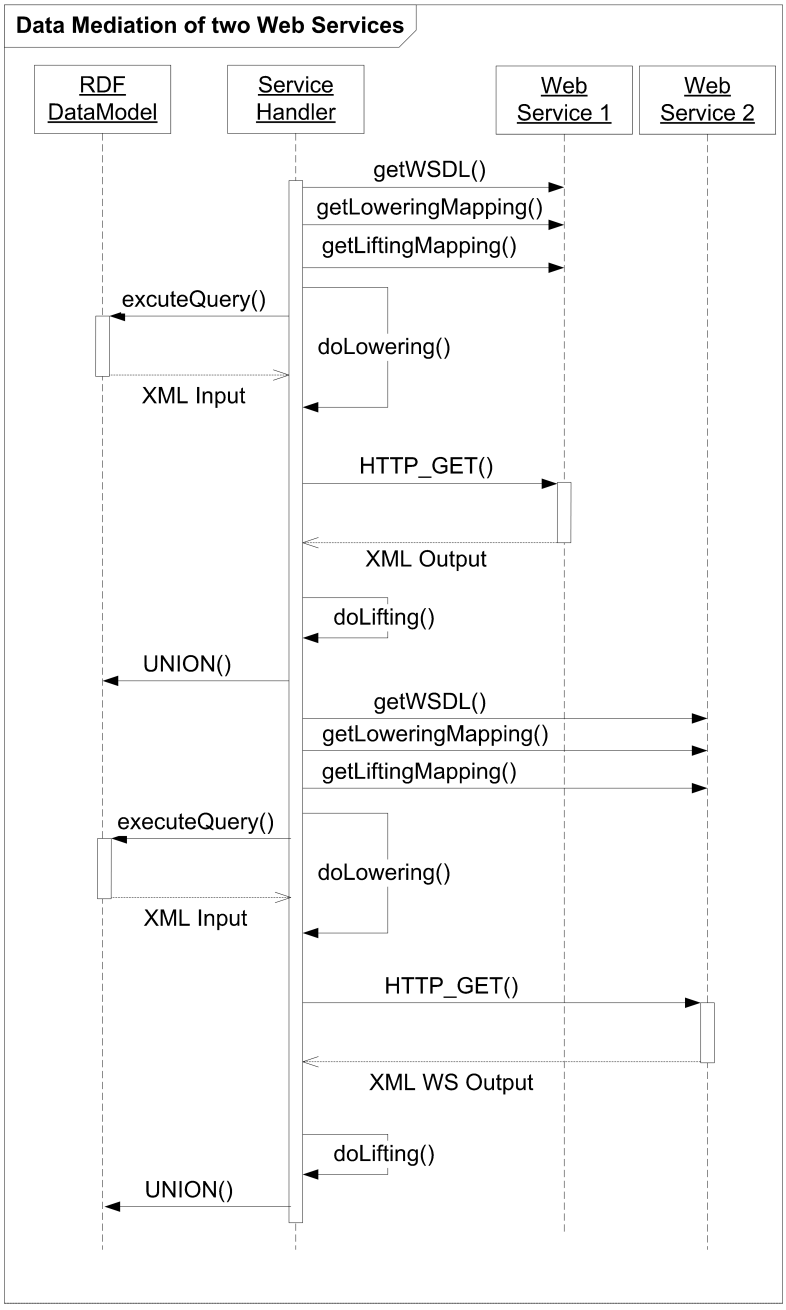

Figure 5.6: Sequence diagram of SAWSDL Lifting and Lowering of two RESTful Services

5.5 Application Registry

The "Application Registry" maintains the end-point references as well as the main semantic properties of the information web services. OWL-S has been selected as the web service description approach in combination with SAWSDL. The registry maintains the following data about the web services:

- ID: Unique identifier of the web service,
- OWL-S URL: The URL of the OWL-S description,
- SAWSDL URL: The URL of the SAWSDL description,
- Inputs: A set of input parameters,
- Outputs: A set of output parameters,
- Preconditions: A set of preconditions,
- Effects: A set of effects.

It is assumed that all web services are described as atomic services. Thus, the invocation is based on a simple request-response interaction. Therefore, each atomic web service of the registry could be described by a set of inputs, outputs, preconditions and effects (IOPEs). The IOPE data is maintained in the registry, to achieve fast access to these information. At runtime, a dynamic parsing and reasoning about the OWL-S description seems to be not suitable, because this not very time efficient [7]. It would lead to a slow down of the planning process, if the system requests the IOPEs several times. Instead, the OWL-S descriptions are preprocessed into an internal data structure of the registry, which leads to a significant faster access to the required information, because the data is maintained in the main memory.

5.6 Web Service Composition Module

It is the aim of the mashup agent to augment existing content by background information. The existing content has been analysed by an unstructured text analysis service, which created initial assertional statements of the $ABox$ about the entities of the content. These statements represent the start information state of the mashup agent. The aim of the web service composition module is to find a plan that transfers the mashup agent from this start information state to a desired goal information state represented by a formal goal (see Chapter 6). The goal information state represents the desired

[7] This has been identified in first tests of the reasoning about the web service descriptions

background information of the final mashup.
The research has pointed out that **scalability** is an ongoing and important objective for web service composition [KSKR05] [SHS08, p.93-94]. In accordance to the stated requirements, scalability is also important to the mashup framework, because the count of available SOAPful and RESTful services is increasing and an appropriate planning performance of the systems has to be ensured.
The planning engine is based on an **evolutionary algorithm** for information web service composition. The evolutionary planner performs a stochastic and parallel search at different points of the search space. The intention is to utilize the information of different points of the search space to achieve a more scalable search, even for large sets of available information web services.
In addition, the approach offers the possibility to incorporate the **user preferences** for price, performance etc. of web service compositions through the utilization of multi-criteria decision theory (see Section 4.6).
A reuse of an existing available planner has been considered and discarded, because a reuse without adaption would not directly address the open research topics. Furthermore, an adaptation of an existing system by this evolutionary approach appeared to require a lot of integration efforts as well as study about the selection of a suitable base system. Instead, it seemed to be suitable to create a lightweight planner that is developed especially for automatic generation of mashups and thus is close to the specific planning problem.
The reuse of an external matchmaker has also been considered, but has been identified as not appropriate. In accordance to Schumacher et al.[SHS08, p.91], a planner based on a sequential composition of consecutive calls of a logic-based semantic service matchmaker may not finds a desired solution, if the matcher does not adhere to the planning state information as well as combined effects from more web services.
In this context, the planner matches for each consecutive pair of planned services S and S' the outputs of S with the inputs of S'. This means that the preconditions of S' have to be fulfilled in the specific planning state. In this context the planning state also contains the effects of S [SHS08, p.91].
However, as mentioned above, if the matcher could not maintain state information, combined effects from different services would be neglected. In fact, the desired mashup information state will often be be based on combined information effects. Thus, the reuse of an separate and external matchmaker seemed to be not promising.

The explanations have outlined that the proposed approach for planning of web ser-

vice composition addresses existing open research topics, which are important to the automatic creation of mashups. The details of the planning module are described in Chapter 6.

5.7 The User Model

The user model is important to achieve the adaption effect among different users of the web portal. The creation of an appropriate user model involves the specification of the features that have to be modeled such as **knowledge**, **interests**, **goals**, **background** and **individual traits** [BKN07, p.5]. In general, the adaption effect of the mashup is expressed by changed mashup content.
The user model could also contain the utility functions of a user for different attributes of web services such as price, performance etc. (see Section 4.6) to achieve an adaption effect on the web service composition in dependence to attributes of the web services.
The provision of context related information could be utilized to achieve an adaption for different environments (e.g. mobile applications).
However, the modeling, creation and update of user models is not part of this thesis. The prototype supports the adaption of data based on a set of predefined task, which could be manually changed by the user. Each task represents a desired goal that is used by the planner to create a plan that retrieves the desired mashup data.

5.8 Personalization Module

The personalization module defines the present goal of the agent. A goal relates to a specific task of the user, which has been predefined in a task model. The aim of the personalization modules is to predict the present task of the user in dependence of the provided content, user model and contextual information.
The personalization module should also be responsible for the update of the user model, because the interests, tasks etc. are changing over time.
The realization of the personalization module is not in the scope of this thesis. Therefore, the prototype system defines only a set of tasks (goal information states) that could be manually changed by the user.

5.9 Presentation Module

The presentation module generates the mashup presentation based on the final $ABox$. In the context of this framework, this means that the presentation module has to transform the RDF triples that represent the $ABox$ into a light-weight representation (e.g. HTML + different JavaScript libraries) that could be consumed by a human agent. Castells et al.. [CFL+04] propose a special information gathering system for financial information that visualizes financial information represented as RDF statements by a model-based approach. For each class of the corresponding ontology one or more possible presentation models are created that specify the appearance (e.g. order, structure) of the RDF data [CFL+04].
Another RDF presentation approach is the Fresnel [BLP05] display vocabulary for RDF. Fresnel specifies what information of an RDF graph should be presented as well as how the information should appear. Fresnel is a browser-independent display vocabulary and is a purely declarative language. In general, Fresnel defines so called Lenses that define which data should be retrieved. Amongst others, this could be specified as a SPARQL query. The format vocabulary specifies how the selected data should appear. If the RDF presentation vocabulary becomes standard, also the presentation of the semantic data could be shared between independent applications.
It is not in the scope of this thesis to investigate and evaluate in detail different approaches for the presentation of RDF data (which represent the mashup data). However, to shows the feasibility and advantage of the automatic mashup generation, a simple model-driven approach has been developed. The approach utilizes the mentioned separation of the selection and formatting of data and is explained in Chapter 7, which provides an overview about the implementation of the mashup framework.

5.10 Conclusion

This chapter has described the architecture of the mashup framework for automatic generation of mashups. The architecture covers the mash-up of contents with background information $[Req.\ F1]$.Furthermore, the purposes of the different modules of the architecture has been explained. Important for the mashup framework is the difference of the $TBox$ and the $ABox$ of the knowledge base. The $TBox$ is based on the terms defined in the OWL ontology. The $ABox$ refers to instances of the classes of the ontology. These instances represent the mashup data at each state of the agent. Therefore, the mashup data is completely described semantically and addresses requirement

[*Req*. *F*7] of the requirements analysis (see Chapter 2).
The background information are retrieved through the invocation of information web services ([*Req*. *F*3]). This chapter has outlined how the data mediation between the web services is achieved to enable an dynamic execution of plans [*Req*. *F*6]. The planning of the web service composition is addressed by an evolutionary algorithm that is described in Chapter 6 ([*Req*. *F*5] [*Req*. *F*6]).
The personalization module and user model address in principle the state requirements on adaptivity([*Req*. *F*2]), but are not in scope of this thesis. Furthermore, the presentation module provides basic visualization techniques, which have to be extended in the future work to address the requirements on rich analyse capabilities and intuitive data manipulation ([*Req*. *F*9).

CHAPTER 6

Planning of Information Gathering by an Evolutionary Algorithm

The mashup framework agent gathers and remixes background information based on a set of available information web services. It is likely that the complex information needs could not always be served by a single web service. Therefore, the mashup agent **plans** the information gathering. In general, this means that the agent specifies which web services have to be invoked as well as the ordering of their invocation.
The proposed planning module searches for a plan based on an **evolutionary strategy**, which traverses the search space through a special stochastic hill-climbing. This means that the algorithm evaluates and maintains simultaneously a population of states (plans) of the search space. New states (plans) are created from old states by special operators for mutation and recombination, which are based on the principles of the natural evolution. The incorporation of preferences and goals is achieved by a special evaluation function that weights the generated plans.
Section 6.1 introduces the general concepts of evolutionary algorithms. Section 6.2 relates the evolutionary concepts to the problem of web service composition planning and describes the resulting evolutionary process. Section 6.3 defines the final optimization problem. The considered states of the state space are evaluated by an objective function drives the optimization process and which is described in Section 6.4. Section 6.5 provides a detailed description of the evolutionary operators, which are important for the evolutionary strategy of the search process. Finally, Section 6.6 concludes this chapter with a short summary.

6.1 General Concepts

Evolutionary algorithms solve optimization problems based on the principles of the **natural** evolution. The algorithms are different in how they imitate natural processes like mutation or recombination, thus a variety of different models exist today[Wei07]. For that reason, it is suitable to analyse the common evolutionary principles and factors.

The natural evolution has developed highly complex **strategies** for the creation, retainment, and adaptation of species, based on the foundations of the chemical evolution [Wei07, p.2-9]. The **individuals** of a species build a common **population**. The **phenotype** (or appearance) of an individual is defined by the **genome**, which is the total of all **genes** of an individual. The genes are part of a chromosome and are important for the inheritance and variance of the genetic code. A gene could have often several occurrences that are called **alleles**. For instance, the hair color gene could have the occurrences black, brown etc..

Evolution is only present, if the **frequency** of alleles is changing significantly. For instance, there is a significant change towards individuals with a brown hair. The **evolutionary factors** for **mutation** and **selection** influence the frequency of the alleles, and thus are important to drive the evolution.

Selection denotes which individuals are selected for recombination as well as which individuals remain in the population of the next generation. The former case is called **parent selection**, whereas the latter process is called **environment selection**.

The parent selection influences the capability to find a partner for the creation of children. It is important to the species to get good descendants to retain the species in the future. The environment selection determines if an individual (parent or child) survives and is present in the next generation of the population [Wei07, p.11-12].

As explained above, selection influences the frequency of different alleles through different **counts** of alleles in descendant populations. Furthermore, it is based on the phenotypical characteristics. This means that the appearance of the specific individual as well as its performance within the population influence the selection.

Mutation is another important evolutionary factor. It is the foundation for a frequent change of the population of the evolutionary process. Mutations are errors of children that occur during the recombination of the parent individuals. The change of the allele frequency could be small or even very big and this depends on the mutation probability. It is often preferred to have only small mutations, which change the allele frequency through several mutation steps within a variety of generations. In contrast, big changes often go along with very elementary negative performance impacts. Individuals with

a bad performance often have not such a high chance to survive in the population very long [Wei07, p.11].
In contrast to selection and mutation, the **recombination** does not change the frequency of alleles in the population. However, genes are highly dependent from each other [Wei07, p.11], and the **interconnections** between them could lead to very different **phenotypical characteristics**. Therefore, recombination is also an important evolutionary factor that serves new phenotypical characteristics [Wei07, p.11] and drives the evolution of the species.
The previous explanations described the evolutionary strategy based on selection, mutation and recombination, which is utilized in search algorithms. The next section gives an short overview about the different approaches to search. Furthermore, it outlines the importance of an evolutionary strategy for the problem of web service composition planning.

6.1.1 Search

Intelligent agents aim to maximize their performance measure, by a **search** through the **state space** of the underlying problem. For instance, the proposed mashup agent searches for a suitable plan that helps to achieve the information goal.
The objective of search is to define an algorithm that traverses the state space efficiently. At this point it seems to be important to denote that the states of the search space are different from the information states that were mentioned in previous chapters. Plans are generated during the search and represent states of the search space. The solution state (plan) is then used to invoke the web services to achieve the desired goal information state.
In general, search algorithms could be classified as **uninformed search** or **informed search** (often called heuristic search) [RN03, p.73]. In an uninformed search the agent is blind. This means that the search algorithm simply generates states of the search space and evaluates if a specific state is a goal state or not. The uninformed search agent is not able to determine how good a state is, thus it is not able to compute a gradual state weighting. Instead in an informed search the agent is able to **evaluate** if one state is more promising than another. Thus, informed search strategies require the definition of an **evaluation function**.
Example strategies of uniformed search are breadth-first, depth-first, depth-limited search or iterative deepening search. Popular informed search strategies are greedy best-first, branch & bound as well as **local search algorithms** [RN03].
Local search algorithms are important to problems in which the path to the solution is

irrelevant. For instance, portfolio optimization requires only the final portfolio and not the order in which the assets are added to the portfolio during the optimization.
Example strategies of local search algorithms are **hill-climbing**, **stochastic hill-climbing** as well as **evolutionary strategies** that are a special form of stochastic hill-climbing.
Russel and Norvig [RN03] outline that local search algorithm only require a **constant amount of memory**, because they only maintain the information of the current state and not the path to the state. Furthermore, these algorithms are able to search more efficiently in large state spaces [RN03]. In fact, such an scalability is important for web service composition planning, because the state space is increasing as more and more web service become available.
Standard **hill-climbing** agents continually get into the direction of an increasing value of the objective function, in case of a maximization objective. Furthermore, they do not perceive any predecessor states. Hill-climbing only considers the immediate neighbour states, and thus operates **local**. The progress of the standard hill-climber towards an

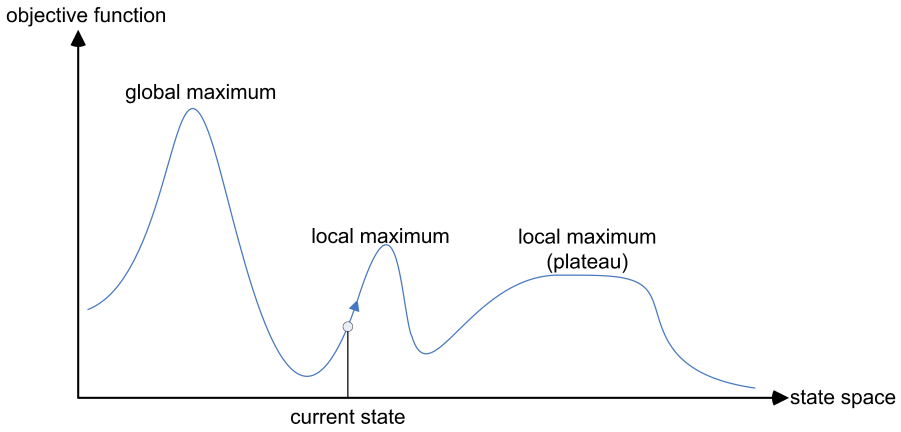

Figure 6.1: One-dimensional state space landscape evaluated by an objective function

optimum is often rapid. However, the found solution is not necessary the global maximum. Search algorithms could be trapped in local maxima, as shown in Figure 6.1. A local maximum is defined as a peak that is higher weighted than each of its neighboring states [RN03, p.113]. A plateaux is a special local maxima that is flat and a hill-climber may finds no way off the plateau. Sequences of local maxima are called ridges. If a standard hill-climber is once trapped in a local maxima it will have no chance to escape from it. This drawback is addressed by several variants of hill-climbing.
Stochastic hill-climbing selects randomly among the possible uphill moves and tends to converge not as fast as the standard algorithm. This leads sometimes to better solutions [RN03].
Another favorite stochastic hill-climber approach is **simulated annealing**, which is able

to perform **up and downhill** moves. Simulated annealing selects randomly a move, if it improves the situation the move is accepted. Otherwise, the algorithm accepts also moves which not improve the situation, but only with a specific probability [RN03]. This offers the algorithm the chance to escape from local maxima.
Evolutionary algorithms are stochastic hill-climbers that perform a **parallel** search on the state space. Thus, they adhere at the same time different states of the state space, which are modified (mutated) and recombined to create new states that address the problem better than its parent states.
In addition to the mentioned approaches, there is also a group of local search strategies that are special to continuous state spaces [RN03]. However, continuous state spaces are not relevant to this thesis, because this would require a derivable objective function. The present state space is a discrete set that is formed by the permutation of the set of web services.

6.1.2 Related Work

Bleul et al. [BWG07] propose an evolutionary algorithm for web service composition within the Web Service Challenge[1]. The knowledge base of the system is based on a taxonomy implemented in XSD Schema [WBG07]. In contrast this thesis proposes an DL knowledge base, which is more expressive than a taxonomy. The service composition is based on the input and output parameters of the web services [BWG07]. Instead, the proposed approach also considers the preconditions and effects of web services.
Furthermore, this thesis has a different approach to the handling of goal conflicts. Bleul et al. [BWG07] propose the computation of multiple objective functions based on a hierarchical evaluation, which is similar to a lexicographic ordering of the objectives (see Section 4.6). In contrast, this thesis handles the goal conflicts directly and proposes in addition the incorporation of user preferences into the objective function.
Furthermore, this thesis considers also the execution of plans and its actions and utilizes real world web services for the proof of concept.

[1] `http://www.ws-challenge.org/`

6.2 Evolutionary Process

The overall objective of the proposed algorithm is to generate a plan that composes different atomic information web services. In general, the evolutionary algorithm evolves a population of plans to find one that is suitable for the given problem.

The problem of web service composition planning is relaxed to the search for an ap-

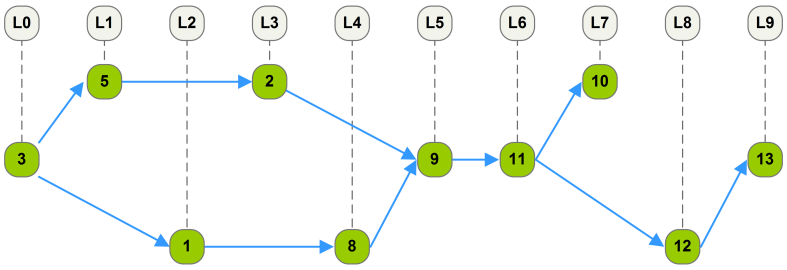

Figure 6.2: Directed-Acyclic-Graph (DAG) of a Web Service Composition

propriate Directed Acyclic Graph (DAG). Figure 6.2 shows an example of such a DAG. The Graph $G = (V, E)$ is based on a set of nodes, whereby each node represents an atomic web service. The web service is denoted by its identifier defined by the application registry. An arrow from a node A to node B denotes that the web service of node A should be invoked in advance to the web service of node B. This is important because this enables the web service of node B to utilize the information created by the former web service. Or in other words, the outputs of the former web service could be used in the latter one. In accordance to the previous explanations, the DAG represents the phenotype of a plan.

The algorithm starts with a random set of possible solution candidates (web service compositions, plans) and performs a simulated evolution. Based on the calculated performances of the plans, the promising plans are selected for a pairwise recombination. The newly generated children are then mutated in different ways with a predefined probability. The mutation of plans changes the web services identifiers as well as the topology. The topology determines the position of a web service in dependence to the other web services. In addition, growing and shrinking of plans is also considered, because the final plan length is not known in advance. The mutation of the individuals leads to plans that are not acyclic. Furthermore, these mutated plans could contain a specific web service more than one time. Therefore, the system uses an operator that repairs the plans. The repaired individuals are then weighted and added to the population. However, the plan population has only a predefined size, which leads to another selection. The environment selection determines which plans survive for the next generation of the population. Figure 6.3 and Algorithm 1 outline the evolutionary process.

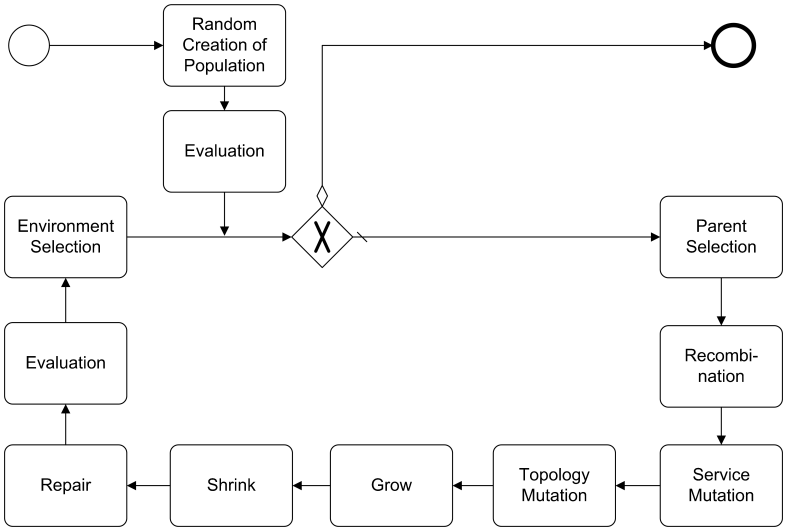

Figure 6.3: Process of the Evolutionary Algorithm

Algorithm 1: Evolutionary Algorithm for Web Service Composition Planning

```
1  g ← 0                                                    /* Index of the Generation */
2  P(g) ← create random plan population of size μ
3  weighting of P(g)
4  while Termination condition not true do
5      P_1 ← select parent plans for λ children out of P(g)        /* Parent Selection */
6      P_2 ← recombine the parent plans of P_1                      /* Recombination */
7      P_3 ← mutate service identifiers of the plans of P_2 with probability prob_S
8      P_4 ← mutate topology structure of the plans of P_3 with probability prob_T
9      P_5 ← grow the plans of P_4 with probability prob_Gr
10     P_6 ← shrink the plans of P_5 with probability prob_Sh
11     P_7 ← repair the plans of P_6
12     weighting of P_7
13     g ← g + 1
14     P(g) ← select μ plans out of P_7 ∘ P(g − 1)          /* Environment Selection */
15 return best plan out of P(g)
```

The previous explanations provided an overview about the evolutionary process for web service composition, which is detailed in subsequent sections.

6.2.1 Genotype Representation

This section proposes various genotypical representations of directed-acyclic graphs and outlines the most suitable one. This is important because the evolutionary operators do not directly change the directed-acyclic graph (phenotype), they change the representation or encoding (genotype) of the graph. Furthermore, such an investigation is important to support an efficient implementation of the evolutionary operators. Encodings could introduce new local optima that are not present to the phenotypical problem. For instance standard-binary encodings are exposed to Hamming cliffs that divide the phenotypical adjacency landscape and make an optimization more difficult [Wei07, p.54-55]. In such a situation often the Gray Code is used, because it has the property that all neighbour values of a discrete search space dimension could be represented by a binary string with Hamming distance one [Wei07, p.54-55].
The web service identifier of the following encodings are not transformed into a binary representation, which would require a lot of transformations during the search process. The following examples refer to the DAG of Figure 6.2.

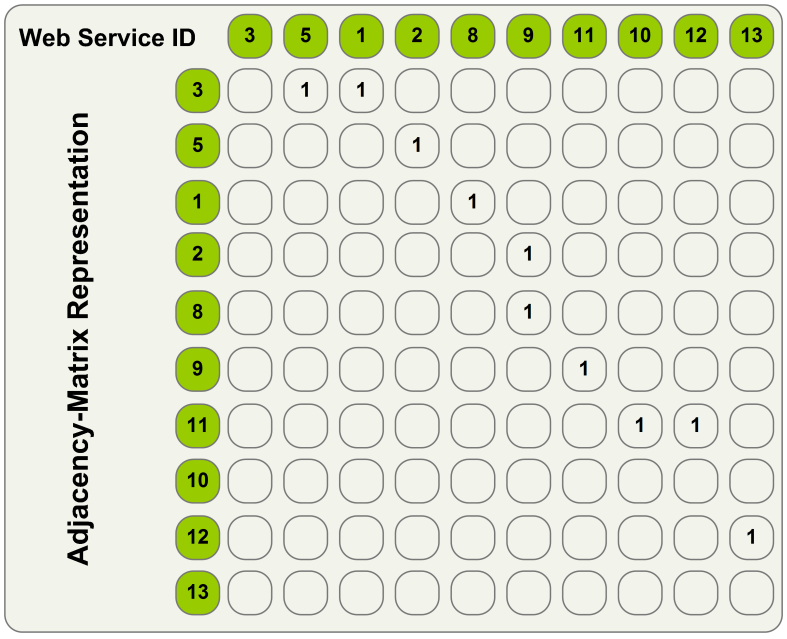

Figure 6.4: Adjacency-Matrix Chromosome Encoding

Cormen et al. propose two standard representations for a graph $G = (V, E)$, the **adjacency list** and the **adjacency matrix** [CLRS01, p.22]. Both encodings support directed as well as undirected graphs.
In the adjacency-matrix representation (Figure 6.4), the vertices are numbered and rep-

resent the corresponding web service identifiers. The set of graph nodes V represents the web services of a specific plan, which form a $|V|x|V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & if\ (i,j) \in E, \\ null & otherwise. \end{cases}$$

A tuple $(i,j) \in E$ represents a directed arc from node i to node j. This encoding needs $\Theta(V^2)$ memory and is independent from the number of edges. However, Figure 6.4 shows that for sparse graphs, those for which the count of directed arcs $|E|$ is significant smaller than $|V|^2$, the algorithm has to maintain unnecessary memory. In this context most of the cells have a null value. However, this problem could be addressed through the representation of the values of the matrix as bit strings.
Nevertheless, it seems to be complicated to define efficient evolutionary operators for this representation. Especially, the recombination of different matrices could be more difficult than in other representations.
In the following adjacency-list encoding, the web service nodes of the DAG are stored in a list K. The notation $K[i]$ represents the element at index i of the list. The topology of the DAG is represented by a second list L, which contains sets of nodes. Thus, $\forall i,j : i = 0,..,length(K) - 1\ ,\ j \in L[i]$ there is an directed arc $(K[i],j) \in E$ in the graph.

The list representation is preferable, because the implementation of recombination

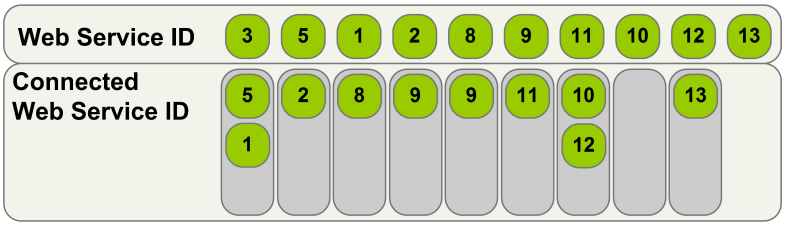

Figure 6.5: Adjacency-List Chromosome Encoding

operators could be based on the manipulation of lists and sublists. Furthermore, the amount of memory of the adjacency-list is $\Theta(V + E)$, which performs for sparse graphs better than the matrix representation.
In the context of directed and acyclic graphs, this thesis proposes a special version of the adjacency-list representation, which provides an simple determination of the acyclic feature of a specific graph. Figure 6.6 represents this encoding. In difference to the previous representation, the topology contains no web service identifier information. This is important, because mutation changes the web service identifiers. In the former representation this would require a change of the identifier at all related positions of the topology. In the proposed representation this is not necessary, because the topological information are separated from the web service keys.

The genotype contains a list K of web service keys and a second list T that contains the topology. The list T is a list of sets of **topological levels**. A topological level is equal to a position (index) of a service in the list K. (In Figure 6.6 the topological levels are denoted by a "L", however, this should only outline the principle. In the concrete representation the level is simply an integer value.) A topological level $l \in T[i]$ corresponds to a directed arc $(K[i], K[l]) \in E$.
As mentioned previously, this encoding also specifies a light-weight rule that determines the existence of cycles. An encoded plan of length L has no cycles, if

$$\forall l \in T[i] : l > i \quad i = 0, 1, 2, \ldots, L$$

holds true. This allows an efficient repair of the genotypes towards directed and acyclic graphs.

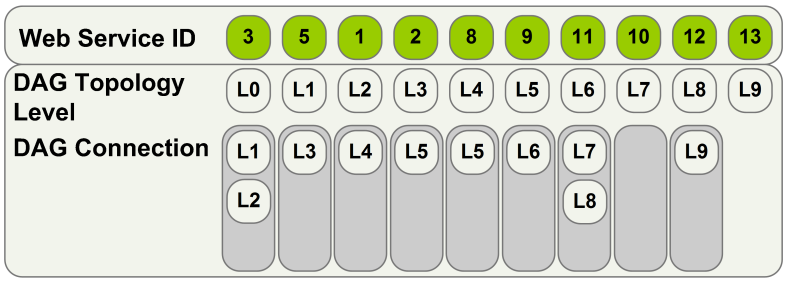

Figure 6.6: Adjajency-List Chromosome Encoding 2

In accordance with Weicker [Wei07], the differences of the phenotype and genotype are visualized in Figure 6.7.
Instead of a direct evaluation of the phenotypical plan representation by the function f, the weighting of a plan is based on the genotypical evaluation function f_{WC}. Furthermore, the decoder function *decode* determines the appearance of a plan based on the genotypical data. The processed weighting (utility) of the individual is stored in the weighting value of the individual, which is used in the selection of the individuals.
The evolutionary operators for recombination and mutation modify the genotype information. In fact, this genotypical information determines the performance of the individual plan that is weighted by the evaluation function f_{WC}.
In accordance to the explanations above, the final genotype G is a tuple $G = (G.R, G.F)$, whereby $G.R \in \mathcal{G}$ denotes the representation of the DAG and $G.F \in \mathbb{R}$ denotes the value of the objective function (or evaluation function) of the individual.

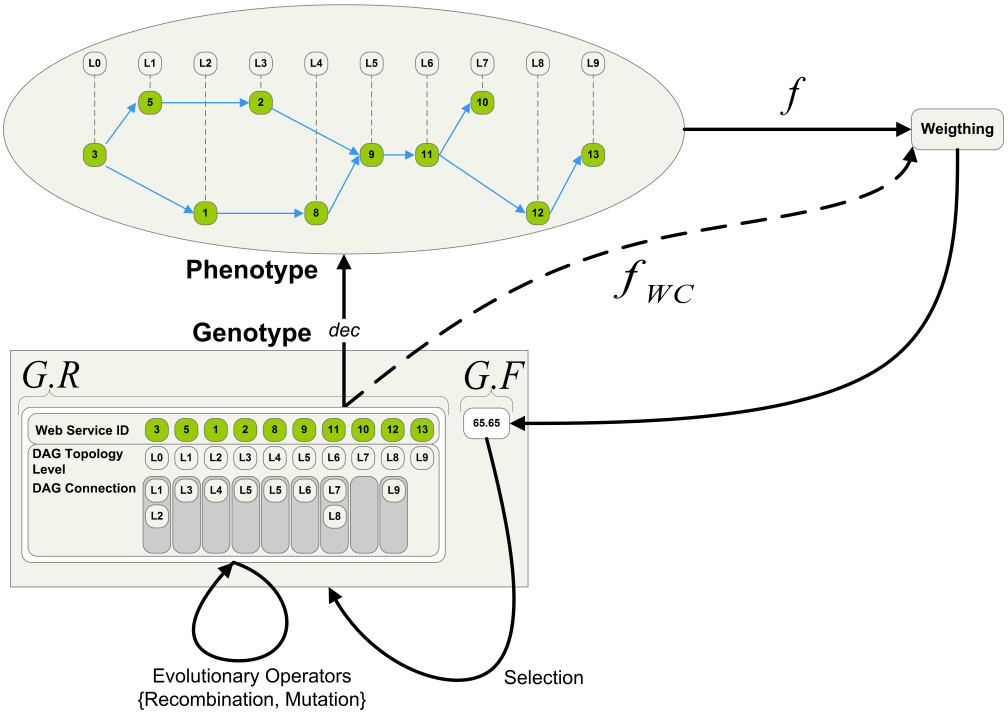

Figure 6.7: Genotype vs. Phenotype Representation

This section has investigated the genotypical representation of the directed-acyclic graphs. Each plan, represented by a DAG, is a state of the state space of the search problem. The next section defines the final problem and explains the calculation of the objective function that drives the search.

6.3 Formal Problem Definition

This section defines the final search problem for web service composition planning, which represented as an optimization problem of a objective function. A web service composition is assumed to be represented as a directed acyclic graph (DAG) $G = (V, E)$ The set of nodes $V = \{v_1, ..., v_n\}$ represents n web services, which are connected through a set of directed connections $(k, l) \in E \subset VxV$. In this thesis the web services of the DAG are assumed to be different. This means that no web service is more than ones in the plan. This has not necessarily to be the case for all web service compositions. However, to simplify the weighting of appropriate states this is assumed in the following.
The planner has to find a specific DAG that transfers the software agent from an initial state to the desired goal state. In the following a specific solution (adjacency-list representation) state in the state space is denoted by x to simplify the notation of the problem [2].

A single-objective optimization problem $(\Omega, f, \succ)$ is defined by the state space Ω, a

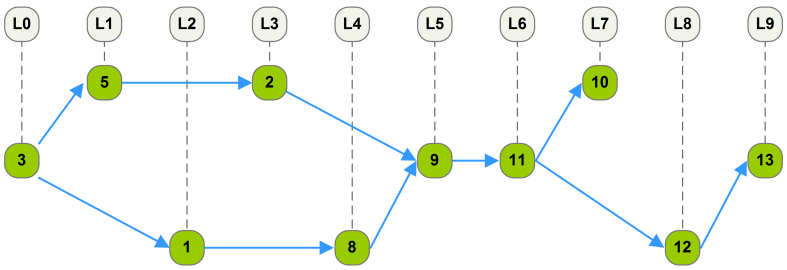

Figure 6.8: Directed-Acyclic-Graph (DAG) of a Web Service Composition

weighting function $f : \Omega \rightarrow \mathbb{R}$ as well as a relation for comparison $\succ$. The weighting function assigns a rating to each solution candidate (state) $x \in \Omega$.

The set of global optima is defined by :

$$\chi = \{x \in \Omega \mid \forall x' \in \mathbb{R} : f(x) \succeq f(x')\} \text{ (see [Wei07, p. 21]).}$$

However, planning is based on multiple objectives such as **correctness**, **completeness**, **length**, **reliability**, **price** etc. of plans. As discussed in Chapter 2, these objectives have to be adhered for the planning process, which makes the search for a solution much more difficult than in single-objective optimization problems.
While single-objective problems may only have one unique optimal solution, a multi-objective problem could have a set of possible solutions vectors. Multi-objective problems arise in many areas such as economics, finance, engineering and computer science

[2] Each solution refers to two lists that represent the web services and the topological information.

and are addressed by decision theory, Operations Research (OR), computer science and related disciplines. The complexity of multi-objective problems is sometimes difficult to solve in traditional approaches of OR [CLV07] and motivate the use of multi-objective evolutionary algorithms.
Different objectives are computed by objective functions. Since the objectives are often conflicting (e.g. completeness versus length) as well as operate on different scales (e.g. price versus performance), the question is how to align the different objectives appropriately (see Section 4.6).

The general multi-objective problem searches for a solution x^* (a state of the state space) that optimizes the vector function $\vec{f}(x) = [f_1(x), f_2(x), ..., f_k(x)]^T$.

$$\vec{f}(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_k(x) \end{bmatrix}$$

It is clear that in most cases no optimum exists such that

$$\forall_{x \in \Omega}(f_i(x^*) \geq f_i(x))$$

holds true. This means that there is not solution x^* that strictly dominates all other solutions. Instead, multi objective problems are denoted by a so called **Pareto-optimum**, which represents the set of possible solutions that are **efficient**. This means that the Pareto-optimal set contains only solutions that are not dominated by an other solution (see Section 4.6).
The selection of a suitable solution out of the Pareto-optimal set is then based on the preference structure of the specific agent. Thereby, the goal conflicts can be handled **a priori**, **a posteriori** or **progressive** in the optimization.
In an a priori preference articulation, the preferences of the decision maker are directly incorporated into the objective function of the problem. This avoids a search spread over the complete Pareto-frontier.
In an a posteriori preference articulation the search algorithm has to return the Pareto-optimal set to the decision maker, which uses decision theory to select a suitable alternative. Furthermore, this selection could be also done by the software agent with one of the methods specified in Section 4.6. However, Bleul et al. [BWG07] state that the Pareto-optimization has disadvantages for evolutionary web service composition planning, because it spreads the search over the complete Pareto frontier and slows down

the performance. In fact, it seems to be appropriate to specify the preferences a priori, because the preference information should be available from the user model, in the context of this framework.
In a progressive preference articulation the optimization algorithm interacts directly with the decision maker. This is not considered here, because this contradicts with the stated requirement for an automatic generation of plans without user interaction.

The final web service composition optimization problem is defined as $(\zeta, f_{WC}, >)$.

$$f_{WC}(x) = w_1 * compl(x) + w_2 * corr(x) + w_3 * length(x)$$

The search space ζ represents all possible permutations of web services. Furthermore, the multi-objective (weighting) function f_{WC} should be maximized for $x \in \zeta$. The weighting function f_{WC} plays an important role, because the algorithm derives the direction of the optimization from it and is based on several sub functions which depend on the different objectives of the search.
In accordance to the definition above, the goal conflicts are handled a priori. The main objectives *CORRECTNESS*, *COMPLETENESS* and *LENGTH* are weighted by a specific linear combination to drive the optimization. These objectives are assumed to be equal for all users of the system. This seems to be suitable, because plans that are not correct or complete and which have a huge mass of unnecessary web services could not be invoked and thus have no utility for the user. Therefore, the static weightings w_1, w_2, w_3 of the linear combination do not change and are important to drive the search.
The sub function $compl(x)$ processes the *COMPLETENESS* of a solution candidate x in dependence to the desired goal state. Furthermore, it is important to achieve an correct plan that could be invoked by the software agent, this is specified by the *CORRECTNESS* function $corr(x)$. In addition, the plan should only contain necessary services to avoid unnecessary workload and costs. This could be achieved by short compositions. Therefore, the length function $length(x)$ has to be a part of the weighting function.
Alternatively, we could define an extended objective function f_{WC*}, which adheres the specific QoS preferences of a user $u \in \varphi$. φ represents the set of users (or better user models) of the web portal.

$$f_{WC*}(x, u) = w_1 * compl(x) + w_2 * corr(x) + w_3 * length(x) + w_4 * utility(x, u)$$

The composition utility $utility(x, u)$ measures the total composition utility for a specific user. It incorporates different QoS objectives of the composition, which are handled by

Multi-Attributive Utility Theory (MAUT) and therefore consider the user preferences. The composition utility contains the following sub objectives:

- *RELIABILITY*
- *SECURITY*
- *PRICE*
- *PERFORMANCE*
- *INTEGRITY*
- *ACCESSIBILITY*
- *AVAILABILITY*
- *COMPLIANCE*

The extended version of the algorithm is not considered in the remaining sections, because an implementation would require a suitable user model that specifies the preferences. The definition of such a user model involves also the automatic creation and update of the user model. Since, this is not part of this thesis, the extended version has not been considered in the implementation and the investigation concentrates on the described weighting function f_{WC}. Nevertheless, an preference based weighting of different solution candidates promises to be feasible in such an evolutionary algorithm, because higher weighted plans are preferred by the algorithm and thus the plans with the higher utility should be part of the final population.

The above descriptions defined the optimization problem. The multiple objective function of the problem is represented by a linear combination and is important to drive the optimization. The following section describes the calculation of the objective functions.

6.4 Calculation of the Objective Functions

This section describes the calculation of the different objective functions of the proposed total weighting function f_{WC}.
In general, the function f_{WC} has to provide a **gradual weighting** instead of an absolute [Wei07, p.23], because otherwise the search would be uninformed. This implicates that also the objective functions should provide a gradual weighting. For instance in an absolute weighting of the correctness, the function only determines if a possible web service composition is correct or not correct. This leads to a lack of information about partial good solutions, and thus the direction of the search could not be specified as detailed as in a gradual weighting.
The proposed planner is a **static planner** that first generates the plan and then invokes it. Dynamic planning with the proposed evolutionary process seems to fit not very well, because this would in general cause the invocation of a lot of unnecessary web services. Furthermore, the invocation of the web services would slow down the whole search. Therefore the proposed planner is static. The objectives could therefore be only specified by the descriptions of the web services.
The web service descriptions are important to evaluate the specific actions of the plan in context of the whole plan. This means the descriptions are used to evaluate if a plan is **correct** and **complete**. It is assumed that the correctness and completeness are measured in percent of completeness and percent of correctness. Therefore these objectives have to be **maximized** and their weightings in the total function have to be greater than zero ($w_1, w_2 > 0$). The length objective of a plan counts the web services. This objective has to be **minimized** to achieve short plans, thus the weighting of the objective has to be smaller than zero ($w_3 < 0$).
For the evaluation of the correctness and completeness of a plan, it would be ideal to check if the state transitions by the actions (information web services) of the plan could be correctly applied and if the final state is the desired information state. Figure 6.9 shows the partial ordered list of web services that could be derived from the DAG.
It is necessary to have an expressive web service description to specify the correctness

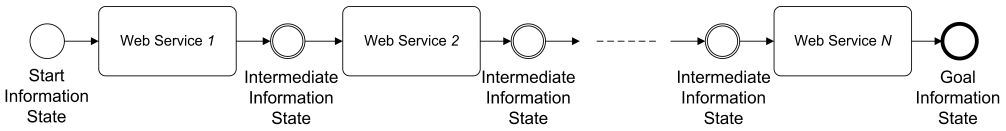

Figure 6.9: Change of the Information State through Information Web Services

and completeness of such plans. This is in accordance with König-Ries et al. [KKRM05], who state that it is important to have an unambiguous functional description of the of-

fered web services as well as the required functionality by the requester. Furthermore, the web service description should be pure state orientated.
The previous explanations implicate that the description of transferred messages is not enough to specify the behaviour of the web service. Thus, the composition of web services by considering only the inputs and outputs of web services is not exact, because it neglects the preconditions and effects of web services. Therefore, it would be better if the inputs and outputs are part of the preconditions and effects, which itself should be state oriented.
The proposed planner is a **functional-level** planner that considers atomic web services that are based on a simple-request response interaction schema. Due to this, for each web service one specific set of inputs, outputs, preconditions and effects have to be considered in the planning process, thus also in the evaluation of the current plans.
As explained above, the description of web services and their evaluation should be state oriented, which could not be achieved by the evaluation of transferred messages. However, the consideration of inputs and outputs could serve as an **heuristic**, which is important for many search processes. Especially for information web services this is suitable, because they create **no real world effects**. This means the preconditions and effects of the services are relaxed to the state of the required information before the invocation and the information state of the world after the execution. The state have not to model effects such as “purchase”, “renting“ etc., which would make the description of the preconditions and effects more complex. Instead, it is possible to describe only the information state before the execution and the resulting information state after the execution of the web service.
In Chapter 5 the information states has been assumed to be represented as states of the $ABox$ of the knowledge base. This means the information states are represented as RDF statements that refer to the terms ($TBox$) defined in OWL. Therefore, considering only the inputs and output concepts of a web service is still an heuristic, because specifying the inputs and outputs of a transferred message is not as expressive as the specification of triples before and after the execution of a web service. In fact, the triples of an OWL vocabulary do not only specify the concepts of instances, they also define object properties between the instances.
This thesis proposes a **simulation** of the information states for a correct evaluation of the plan. The simulation of the different information states is based on a simulation of the triple structures that are required by a web services (precondition) and are generated through its invocation (effect). In this context, checking the precondition of a web service is performed by a SPARQL ASK query that proves if the current information state provides the required triples. Furthermore, the SPARQL $CONSTRUCT$ query is

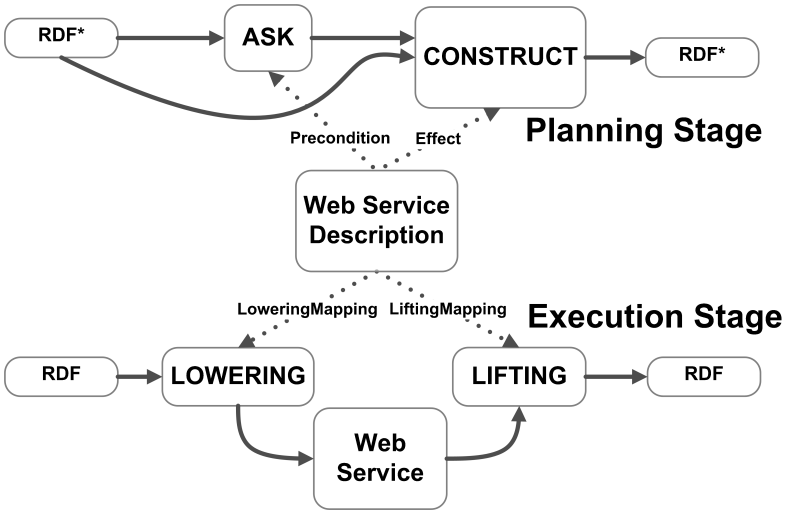

Figure 6.10: Information State Simulation in the Planning Stage vs. Invocation in the Execution Stage

used to **simulate** the generated triples of the resulting information state.
Therefore, the preconditions and effects of the web services are described as SPARQL *ASK* and *CONSTRUCT* queries within the OWL-S web service descriptions.
The inputs, outputs, preconditions and effects of a web service are derive from its OWL-S description. In Chapter 5 the preprocessing of these information into the registry was proposed. In fact, the evolutionary algorithm processes the objective function as often as states have to be evaluated during the search. Therefore, a fast access to these information is required to avoid an unnecessary slow down of the search process.

The next section describes the exact simulation of the states of the plan in detail. The subsequent sections utilize this simulation to specify the evaluation of the correctness and completeness of plans. The simulation of the plan is costly and could therefore be not done for every considered state of the search process. Therefore, the subsequent sections also describe the heuristic calculation of the objectives, which are important to drive the optimization.

6.4.1 Information State Simulation

Planning is the task of the creation of a sequence of **actions** that achieve an desired **goal** [RN03, p.375]. The planner simulates the behaviour of the different actions and thus is able to determine if the final information state is **complete** and if the ordering of the actions is **correct**. The simulation is based on the simulation of the structure of the *ABox*

statements that are created and requested by the different web services.
The simulation of the information states is shown in Figure 6.10. In the planning process for each web service the preconditions describe the structure of required information and the effects describe the structure of generated output statements.
The preconditions are described by a SPARQL *ASK* query, which evaluates if the structure of required input statements could be served by the available *ABox* structure. If this is true, an SPARQL *CONSTRUCT* query is used to generate the structure of triples that are created by the invocation of the web service.
In general, this is similar to the execution process described in Chapter 5. This means that triples are required as inputs of the web service and output triples are generated by the web service. However, in case of the simulation, the triples contain only the types of instances as well as blank nodes that are used to create the object properties between different instances.
Listing 6.1 and 6.2 provide an example description of a precondition and a effect description of a web services.

```
<process:hasPrecondition>
   <expr:SPARQL-Condition>
     PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt;
     PREFIX pup: &lt;http://proton.semanticweb.org/2005/04/protonu#&gt;
     PREFIX psys: &lt;http://proton.semanticweb.org/2005/04/protons#&gt;
     PREFIX ptop: &lt;http://proton.semanticweb.org/2005/04/protont#&gt;
     PREFIX minerva: &lt;http://127.0.0.1/ontology/minerva-portals#&gt;
     ASK  { ?y rdf:type minerva:Stock .
            ?y psys:mainLabel ?name .
          }
   </expr:SPARQL-Condition>
</process:hasPrecondition>
```

Listing 6.1: SPARQL-based Goal Representation

The example is based on a stock quote web service described by OWL-S. The stock quote service requires input instances of the type "minerva:Stock" that are specified by their "mainLabel". The SPARQL *ASK* query is used throughout the planning to evaluate if this precondition is fulfilled for a given *ABox*.
Listing 6.2 specifies the result of the stock quote web service. If the precondition is fulfilled, then the SPARQL *CONSTRUCT* query is used to generate the output information. In this case, the stock quote web service creates a "minerva:hasQuote" object property for the exsiting stock instance. The object property relates a stock quote instance that is denoted by a blank node. The blank node specifies that it is of type "minerva:StockQuote" and that it is described by data and price.

```
<process:hasResult>
<process:Result rdf:ID="CompanyAddressResult">
<process:hasEffect>
<expr:SPARQL-Expression>
  PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt;
  PREFIX pup: &lt;http://proton.semanticweb.org/2005/04/protonu#&gt;
  PREFIX psys: &lt;http://proton.semanticweb.org/2005/04/protons#&gt;
  PREFIX ptop: &lt;http://proton.semanticweb.org/2005/04/protont#&gt;
  PREFIX minerva: &lt;http://127.0.0.1/ontology/minerva-portals#&gt;
  CONSTRUCT   { ?y rdf:type minerva:Stock ;
                   minerva:hasQuote _:a .

                _:a rdf:type minerva:StockQuote ;
                   minerva:date "" ;
                   minerva:price "" . }
  WHERE {       ?y rdf:type minerva:Stock .
                ?y psys:mainLabel ?name . }
</expr:SPARQL-Expression>
</process:hasEffect>
</process:Result>
</process:hasResult>
```

Listing 6.2: SPARQL-based Goal Representation

It is important to note that in the simulation only one instance of "minerva:StockQuote" is created, instead in the real world invocation the service could respond a set of stock quotes. However, during the simulation only the structure of available information is important to determine the correctness or completeness of the composition.

Algorithm 2 provides an overview about the general steps of the simulation process. The $ABox_{StartModel}$ represents the available semantic information provided by the content (the semantics were extracted by the Calais web service). The partial ordering of the web services, which could be derived from the DAG, is used to simulate the plan invocation. For each web service the algorithm checks the precondition (ASK query). If the precondition is true, then the $CONSTRUCT$ query is executed and the resulting simulated statements are added to the simulated $ABox$ ($ABox_{Simulated}$).

The previous explanations outlined how the information states of the plan could be simulated by SPARQL ASK and $CONSTRUCT$ queries, which simulate the change of the $ABox$ described by the terms of the $TBox$. The expressiveness of this simulation is based on the expressiveness of the described instances of the $ABox$. In the context of

Algorithm 2: Simulation of Information States during the Planning Process

Data: $KB = \langle TBox, ABox_{StartModel} \rangle$, Genotype of the DAG

Result: $ABox_{Simulated}$

```
1 ABox_Simulated ← ABox_StartModel
2 for all web services of the partial order derived from the DAG do
3     query_ASK ← webService^i.precondition          /* get the precondition */
4     bool ← execute(query_ASK)          /* evaluate if the precondition is true */
5     if preCondition == true then          /* web service precondition is true */
6         query_CONSTRUCT ← webService^i.effect          /* simulate the effect */
7         ABox_Simulated ← ABox_Simulated ∪ execute(query_CONSTRUCT)
8 return ABox_Simulated
```

this framework, OWL has been used as the description logic. Thus, the expressiveness of the information state simulation could not be more than what could be expressed in OWL.

However, at this point we could also state that the SPARQL processing makes the simulation very costly and it is not appropriate to simulate each plan. Heuristics are therefore needed that drive the optimization towards a promising plan (state), which is then evaluated by the more exact simulation (see Figure 6.11). The next section specifies the concrete objective functions and their heuristics.

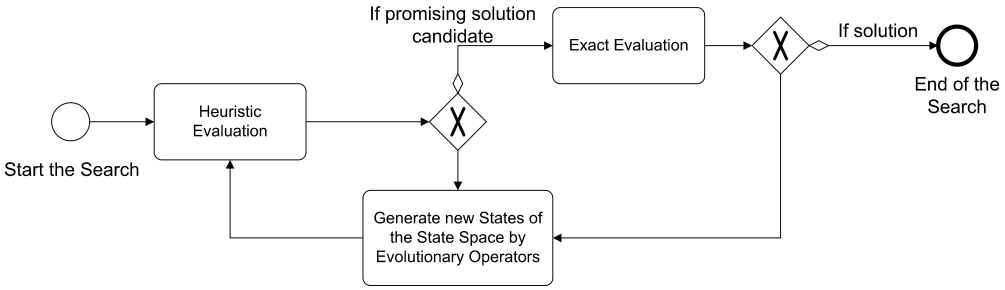

Figure 6.11: Heuristic versus Exact Evaluation of Plans

6.4.2 Calculation of the Objective Completeness

The completeness measures to which degree the **goal** of the planning process is achieved. The previous explanations outlined how the output triples of the information web service could be simulated, this is used to exactly evaluate the completeness of a graph. The goal of the final plan is specified by an SPARQL *ASK* query that asks for those

triple structures that should be created by the invocation of the plan.
Listing 6.3 shows a goal represented as a SPARQL *ASK* query. The goal specifies that the agent is interested in a set of specific triple patterns.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX pup: <http://proton.semanticweb.org/2005/04/protonu#>
PREFIX psys: <http://proton.semanticweb.org/2005/04/protons#>
PREFIX ptop: <http://proton.semanticweb.org/2005/04/protont#>
PREFIX minerva: <http://127.0.0.1/ontology/minerva-portals#>
ASK   { _:a rdf:type pup:Company ;
      psys:description "" ;
      minerva:hasWikiPage "" ;
      minerva:hasLogo "" ;
      minerva:depiction "" ;
      ptop:locatedIn "" .

  _:b rdf:type pup:PostalAddress .
}
```

Listing 6.3: SPARQL-based Goal Representation

The pattern describes that the agent would like to have information about a company. Furthermore, the company instances should have data type properties that contain information about the description, WikiPage, logo and pictures of the company. In addition each of the desired individuals should have an object property to a postal address. The SPARQL and RDF based representation of the information goal describes the desired structure of the information state on the basis of triples and is thus as expressive as the underlying approach for semantic description of information.
Algorithm 3 outlines the calculation of the exact completeness function.
The agent first simulates the plan and than executes the *ASK* query that represents the

Algorithm 3: Exact Calculation of the Completeness of a Plan

Data: Goal $query_{ASK}^{Goal}$, $KB = \langle TBox, ABox_{StartModel} \rangle$, Genotype of the DAG
Result: Completeness $c \in \{0, 1\}$

```
1 ABox_Simulated ← simulate(KB = ⟨TBox, ABox_StartModel⟩ , Genotype)
2 b ← execute(query_ASK^Goal) on KB = ⟨TBox, ABox_Simulated⟩
3 if b == true then                                   /* plan is complete */
4 |   return 1
5 else                                              /* plan is not complete */
6 |   return 0
```

goal on the simulated *ABox*. However, the SPARQL *ASK* query returns only *TRUE*

or *FALSE* and provides no gradual weighting. Therefore such a representation of the goal would make the agent blind. This means that algorithm would perform an uninformed search. As explained previously, a **gradual** evaluation of the goal fulfillment is required.

One option is to process the partial completeness of the triple pattern of the RDF graph, which could be difficult and costly to process. Furthermore, the simulation itself is very costly.

Therefore it is more promising to specify an heuristic for the calculation of the completeness, which is gradual and could be processed without much effort. Furthermore, the goal representation is refined to a **list of wanted concepts and data type properties**. This means that only desired output concepts and data type properties are stated. In the following this thesis denotes concepts and data type properties as **resources**.

Algorithm 4 outlines the processing of the heuristic function. The heuristic function

Algorithm 4: Heuristic Calculation of the Completeness of a Plan

Data: $KB = \langle TBox, ABox_{StartModel} \rangle$,
List of wanted output resources *wantedResources*,
Genotype of the DAG

Result: Completeness $c \in [0, 1]$

```
1 for all web services of the partial order derived from the DAG do
2     outputResources ← outputResources ∪ webService^i.outputs
3 c ← containsAll(outputResources, wantedResources)
4 return Completeness c
```

gathers the output resources of all web services and merges them into one set. The operation "'containsAll"' checks, if for all wanted resources $w \in wantedResources$ there is a resources $c \in outputResources$ such that c subsumes w ($c \sqsubseteq w$). This means that it checks that each output resource of the list of wanted concepts is addressed by an either equal or more specific output resource of the plan.

The heuristic leverages the hierarchy of the DL knowledge base to evaluate the subsumes relationships for the concepts as well as concrete roles (data type properties). In fact, subsumes is one of the central reasoning services of a DL knowledgebase.

Instead to the exact completeness function, the heuristic provides gradual results based on the percentage of served wanted concepts of the plan.

The previous explained heuristic does not take into consideration that there are often web services that do not directly contribute to the goal state, but instead contribute to the overall plan. This means that intermediate web services have to be considered, even

if their output resources are not directly stated in the goal. Therefore another additional heuristic has been created that counts the **number of promising web services**.
A promising web services is a service which creates an output that is part of the planning goal or that is required as an input of another promising web service.
Algorithm 5 outlines the calculation of this heuristic. For each web service the algorithm checks if it directly or indirectly contributes to the goal. The treshold is important, because the important services could not be checked in one run.
Both, the heuristic for the completeness and the count of important web services drive

Algorithm 5: Heuristic Calculation of the Count of Promising Web Services

Data: Genotype of the DAG,
List of wanted output resources *wantedResources*
Result: Count of promising web services $count \in \mathbb{Z}$

```
1 importantWebServices ← create an empty set for the important web services
2 i ← 0
3 for i ≤ treshold do
4     for all web services of the partial order derived from the DAG do
5         if webService^i has an output resources that subsumes a resource of wantedResources
          then
6             add webService^i to the set of importantWebServices
7             add the inputs of webService^i to the wantedResources
8 count ← importantWebServices.size
9 return count
```

the optimization towards the objective of completeness.
The previous explanations have described an exact calculation of the completeness and two heuristics that drive the optimization process. The next section describes the calculation of the correctness of the plans.

6.4.3 Calculation of the Objective Correctness

The correctness of a plan determines if it could be invoked correctly such that the preconditions of each web service are fulfilled[3]. Therefore, correctness ensures that the web services are ordered correctly.

[3]Postconditions have been neglected, because the information web services typically have no post conditions.

Algorithm 6: Exact Calculation of the Correctness of a Plan

Data: $KB = \langle TBox, ABox_{StartModel} \rangle$,Genotype of the DAG

Result: Percentage of correctness $c \in [0, 1]$

1 $ABox_{Simulated} \leftarrow simulate(KB = \langle TBox, ABox_{StartModel} \rangle, Genotype)$

2 $count \leftarrow$ get the count of web services with fulfilled preconditions from the simulation

3 $c \leftarrow count /$ number of web services of the plan

4 **return** (Percentage of correctness c)

Similar to the calculation of the completeness it is possible to evaluate the correctness on the basis of the simulation of the plan. Since, the simulation checks the preconditions of each information web service of the plan with an SPARQL ASK query, the algorithm is able to evaluate if all preconditions are fulfilled.
It is important to note, that the SPARQL ASK query of the precondition checks different RDF triple patterns of the simulated semantic data (simulated $ABox$). This means that there is still the possibility that the plan could not be invoked fully, even if the simulation has evaluated it as a correct plan. The reason is straightforward, the simulation assumes that for a given input, the web service is able to create an answer. However, in the real world this is not true in any case. The **accessiblilty** is a measure of the success rate for requests (see Chapter 2) and gives and additional motivation for the incorporation of QoS attributes directly into the planning process.

Algorithm 6 outlines the calculation of the exact correctness. The algorithm simply counts the number of services, which have fulfilled preconditions and thus could be in general invoked with the available information. As stated above, the simulation of plan needs a lot of processing time. Therefore, the correctness is also supported by an heuristic function.
The heuristic correctness function is based on the inputs and outputs of the web services. Thus, it checks if for each web services if the inputs are served by information from the mashup content or by an output of one or more predecessor web services. Of course, this is only a heuristic, but it is used to drive the search in a promising direction. Algorithm 7 outlines the calculation of the heuristic evaluation of the correctness of a plan.

This section has explained the calculation of the correctness of the plan, which ensures the general invocability of the web services. The heuristic correctness of the plan is an important measure to drive the optimization process towards a promising state. If a heuristic correct and heuristic complete solution is found, the algorithm checks the ex-

Algorithm 7: Heuristic Calculation of the Correctness of a Plan

Data: $KB = \langle TBox, ABox_{StartModel} \rangle$,
Available Input Resources from the Content *inResources*,Genotype of the DAG
Result: Percentage of correctness $c \in [0, 1]$

```
1 count ← 0                          /* count of web services with fulfilled inputs */
2 for all web services of the partial order derived from the DAG do
3     serviceOutputResources ← get output resources of predecessors of webService^i
4     if containsAll(webService^i.inputs, inResources ∪ serviceOutputResources) == 1
5     then                           /* all inputs of the web service are served */
6         count ← count + 1
7     else
          /* do nothing,inputs are not fully served */
8 c ← count / number of web services of the plan
9 return (Percentage of correctness c)
```

act correctness of the plan. If the simulation confirms the solution of the heuristic the algorithm has found the desired plan. The next section explains the calculation of the length objective, which is important to achieve short plans.

6.4.4 Calculation of the Objective Length

The length of a plan is an important objective, because it is assumed that unnecessary long plans lead to higher execution times and unnecessary costs. Therefore, the length of the plans should be **minimized**. The objective function f_{WC} is maximized by the algorithm. Therefore, the weighting of the length objective has to be smaller than zero ($w_3 < 0$) to minimize the length.
The calculation of the length is simple, because the plan length is denoted by the count of web services in the genotypical encoding, which could be retrieved easily.
First tests of the algorithm as shown that the algorithm performs better, when the count of unimportant web services is considered instead of the count of all web services. The calculation is still simple, since the count of important services has been already specified for the completeness, thus the objective could be processed

$$countOfUnImporatantService = length - countOfImportantWebServices\ .$$

This section explained the calculation of the objective functions of the optimization

problem. It is important to note that the exact calculation of the objectives is often to costly and slows down the performance of the algorithm. It is therefore suitable to determine appropriate heuristics that drive the optimization towards a promising state of the state space. In this context, a promising state is a state that is heuristic correct and complete. Ones a suitable state is achieved, the exact calculation is performed to ensure the quality of the plan.
The stated heuristics work well with the proposed algorithm (see Chapter 8) and their performance depends mainly on how fast the subsumes relation between the resources could be processed. As explained in Chapter 4, subsumption is an important inference task in DL knowledge bases. Therefore, it is important for future research to increase the performance of these heuristics or to find other heuristics to improve the speed of the planning process.

6.5 Description of Evolutionary Operators

The previous section defined the optimization problem and described the calculation of the objective function. This section describes the operators that are used to generate new states (plans) within the search process.

6.5.1 Selection

The selection is mapping from a population of size r into a population of selection candidates of size s, thereby the weighting $\in \mathbb{R}$ influences the mapping of the selection. The selection operator is therefore formally defined as:

$$Sel^{\xi} : (\mathcal{G} \times \mathbb{R})^r \rightarrow (\mathcal{G} \times \mathbb{R})^s \, .$$

Furthermore, ξ is a specific state of the random number generator and $\mathcal{G}$ denotes the genotype. The random number generator influences the generation of new states as well as the behaviour of the evolutionary operators. One instance of the random number generator creator is used by the whole planning module. This is preferable, because the random number generator could be optionally set to a specific seed value. The seed value is a unique key for a sequence of random numbers. Normally, this option is not used by the framework. However, the seed value could be used to reproduce results and investigate the sensitivity of parameters during the evaluation. Therefore,the utilization of the seed value makes different analysis comparable.

6.5.1.1 Parent Selection

The parent selection determines the parent individuals that are used for the recombination.
Each individual (state) of the population should have a chance to be selected, because otherwise the effort of its maintenance would be not appropriate [Wei07, p.66].
The selection influences the diversity of a population. For instance, if the selection is strict, it selects only the best individuals, and thus the algorithms converges very fast towards a specific state. This could lead to a population that contains only equal individuals. Such a population is called a converged population. However, as explained earlier, a fast convergence could trap the search in a local optima [Wei07]. In a converged population, the recombination operator is not able to create new children. Thus, the change depends then on the mutation, which could lead to the case that the algorithm is not able to escape from the local optima. Diversity is therefore important for the population to consider different states of the state space and not only one state several times (The concrete diversity measure of the genotype is described in Chapter 8).
The research has turned out a variety of selection strategies such as **fitness proportional selection** and **tournament selection** to avoid an too fast convergence of the algorithm [Wei07].
The proposed algorithm is based on a tournament selection strategy. In the tournament

Algorithm 8: Tournament Selection

Data: population P,tournament probability $prob_t$
Result: the selected plans SP

```
1  SP is a empty list of the selected plans
2  i ← 1
3  for i ≤ selectionsize do
4      T ← create a random tournament between two random plans
5      if next random number r ∈ [0,1] < prob_t then      /* highest performance wins
       tournament */
6          get the plan with the greater weighting
7      else                                  /* smallest performance wins tournament */
8          get the plan with the smaller weighting
9      add the winner to SP
10     i ← i + 1
11 return (the selected list of plans SP)
```

selection two random plans (states) of the population are selected to perform a tournament against each other. The **tournament probability** denotes the probability that the plan with the higher performance (weighting) wins the tournament, and thus is selected for recombination. Thus, a higher tournament probability tends to drive the selection to the individuals with the better performance. In each tournament the winner is added to the candidates for recombination. The tournament supports the retainment of diversity, because also worse plans have a chance to be part of the recombination phase if the worse plan wins the tournament against the better plan. In addition, the tournament could be itself only contain worse plan, such that in every case a worse plan is selected for recombination. Algorithm 8 outlines the basic principle of the tournament selection.

6.5.1.2 Environment Selection

The environment selection is another form of selection. Recombination and Mutation create children (states) that are added to the population. However, the population has a predefined size and therefore not all individuals could survive. The environment selection determines which of the individuals are present in the next generation of the population. Possible strategies are the deletion of the individuals with the smallest performance, the deletion of the oldest individuals or random deletion.
This evolutionary algorithm deletes directly the worse individuals. Thus, the best performing individuals survive. However, the deletion adheres that each individual could be only ones in the population. This is very important to remain the diversity of the population.

This section has explained the parent and environment selection of the evolutionary algorithm. The parent selection is important to determine the set of individuals that are recombined. The recombination operator is described in the next section.

6.5.2 Recombination

The recombination is mapping from a population of size r into a population of children of size s. The recombination operator is formally defined as:

$$Rec^{\xi} : (\mathcal{G})^r \rightarrow (\mathcal{G})^s \, .$$

The recombination is only meaningful if $r \geq 2$ and $s \geq 1$. The operator does not change the allele frequency. Moreover, it is responsible to create new interdependencies between different genes. The interdependencies are important to create new phenotypical characteristics. In fact, the nodes between the graph are highly interdependent, because the resulting ordering specifies, if a genotype is correct.
Algorithm 9 outlines the principle of the two-point crossover that has been implemented. The *sub* function is assumed to operate on the list of web services as well as on the topology of the genotype. The two-point crossover cuts the genotypes at two indexes and recombines the intermediate parts of the genotypes. An example of a double crossover is shown in Figure 6.12. In this case, the topological levels four and five are interchanged between the two different plans.

Algorithm 9: Recombination through a Two-Point Crossover

Data: selected plans P, CP is a empty list of the selected plans
Result: the child plans CP

1 $temp \leftarrow P$
2 **for** *tuple* $(G1, G2)$ *of genotypes of* $temp$ **do**
3 $\quad temp.remove(G1)$ and $temp.remove(G2)$
4 $\quad point1 \leftarrow$ new random number $\in [0, min(G1.length, G2.length))$
5 $\quad point2 \leftarrow$ new random number $\in [0, min(G1.length, G2.length) - point1) + point1$
6 $\quad G1_{new} \leftarrow G1.sub[0, point1) \& G2.sub[point1, point2) \& G1.sub[point2, G1.length)$
7 $\quad G2_{new} \leftarrow G2.sub[0, point1) \& G1.sub[point1, point2) \& G2.sub[point2, G2.length)$
8 $\quad CP.add(G1_{new})$ and $CP.add(G2_{new})$
9 **return** (the recombined children CP)

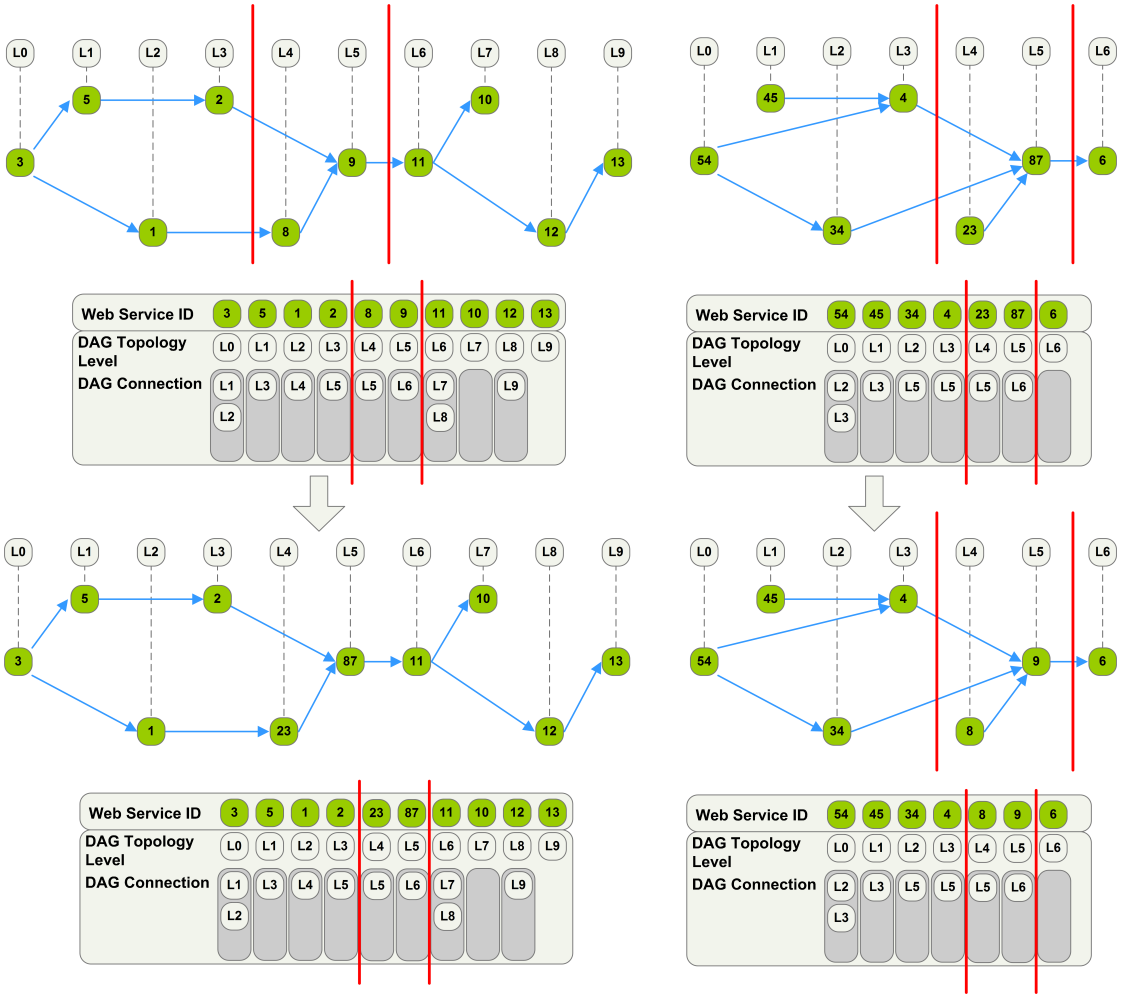

Figure 6.12: Two-Point Crossover

6.5.3 Mutation

The mutation is a mapping from a population of size r into a population of size r. The mutation operator is formally defined as:

$$Mut^{\xi} : (\mathcal{G}^r) \rightarrow (\mathcal{G}^r) \ .$$

ξ is a specific state of the random number generator and $\mathcal{G}$ denotes the genotype.

The general mutation has several occurrences. The **mutation of web service identifiers** (Figure 6.13) changes the list of considered web services of the genotype with an predefined probability. This mutation changes the plan significantly, because the new web service has another web service description and this influences the weighting of the genotype.

The topology mutation changes the topological information of the DAG. This mutation disarranges **connections** (Figure 6.14) as well as whole **topological levels** (Figure 6.15). The change of the topological information influences also the weighting function, because this mutation could lead to a change in the correctness of the considered plan.

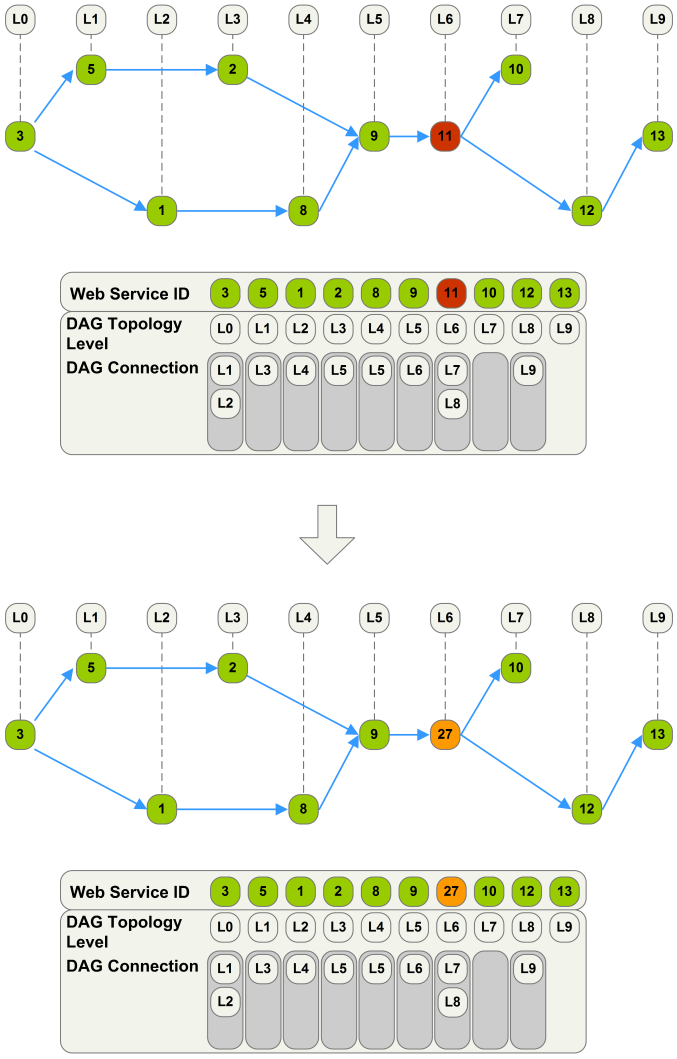

Figure 6.13: Mutation of the web service identifier

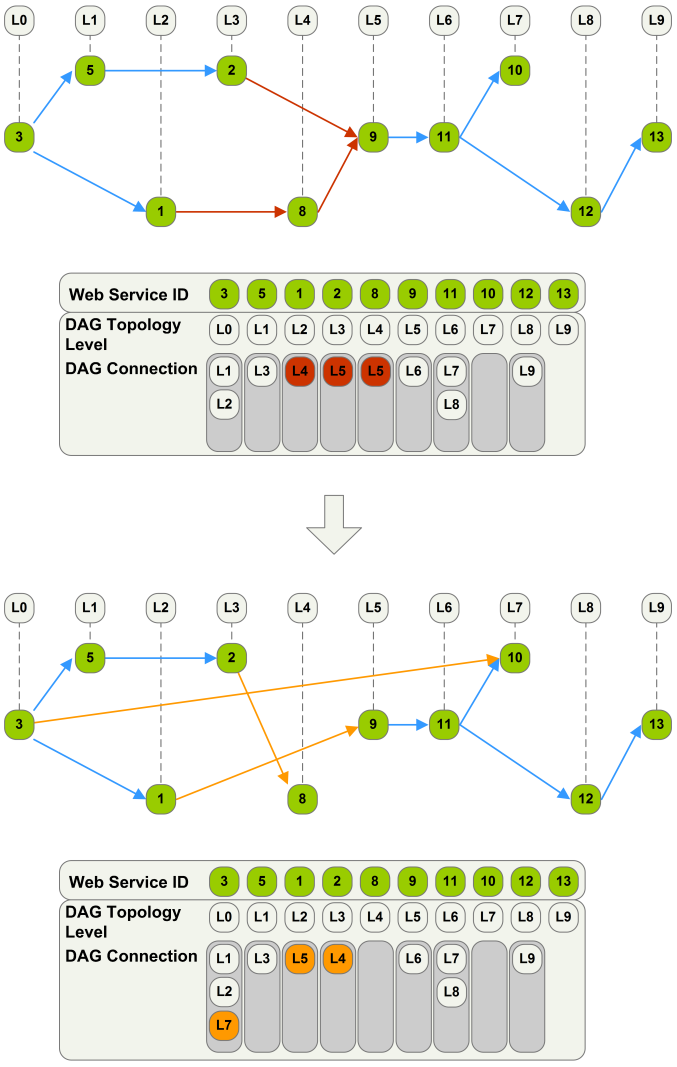

Figure 6.14: Mutation of the topology connections through the disarrangement of specific edges of the DAG

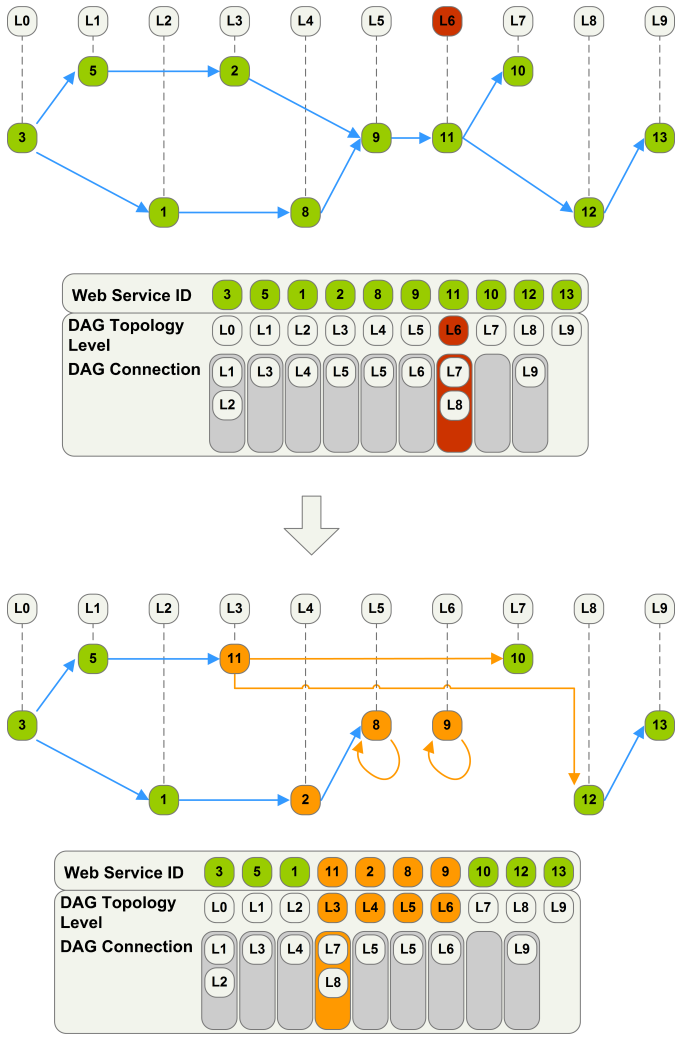

Figure 6.15: Mutation of the topological level through a disarrangement of a whole level of the DAG

6.5.3.1 Growing and Shrinking of Plans

Growing and shrinking are special types of topological mutations of plans.

$$Grow^{\xi} : (\mathcal{G}) \rightarrow (\mathcal{G})$$

A solution is grown by adding a new topological level at a random position of the DAG. The new topological level contains a random web service and random connections. Algorithm 10 outlines the computation of the genotypical grow. The grow probability

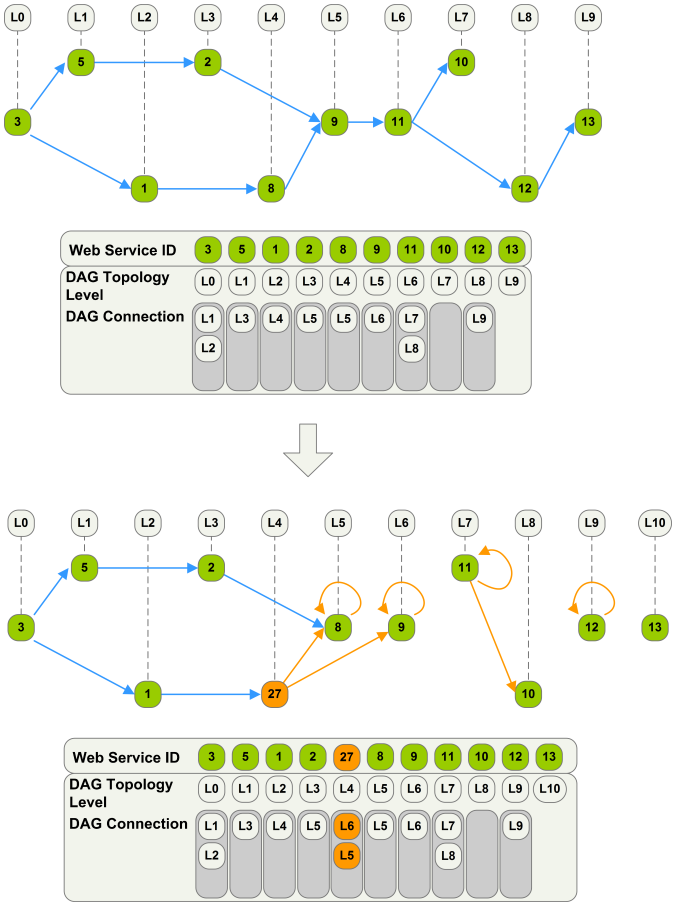

Figure 6.16: Growing of the solution by adding a new topological level at a random position of the DAG

denotes the probability that a genotype of the population is grown. The newly added web service is randomly selected from the available ones which are registered in the web service registry.

A solution is shrunk through the random deletion of a topological level of the genotype. The prinicple is outlined in Figure 6.17.

Algorithm 10: Growing of Plans

Data: population, $growProbability$, $maxServiceIdentifier$

Result: population

```
1 for each genotype of the population do
2     if next random number ∈ [0,1] ≤ growProbability then
3         newPosition ← next random number ∈ [0, planLength)
4         serviceIdentifier ← random number ∈ [0, maxServiceIdentifier]
5         topology ← random topological level for connection
6         add serviceIdentifier to list of web services at index newPosition
7         add topology to the topology at index newPosition
```

$$Shrink^{\xi} : (\mathcal{G}) \rightarrow (\mathcal{G})$$

Algorithm 11 outlines the random shrinking of a genotype of the population. The shrink probability specifies the probability that a genotype of the population is shrunk. In addition, the topological level that should be deleted is also specified by a random number which is derived from an uniform distribution. Thus, not only the last position of the plan is deleted also inderemedatiate positions could be deleted to shrink the plan.

Algorithm 11: Shrinking of Plans

Data: population, $shrinkProbability$

Result: population

```
1 for each genotype of the population do
2     if next random number ∈ [0,1] ≤ shrinkProbability then
3         position ← next random number ∈ [0, planLength)
4         remove position from list of web services
5         remove position from list of topological levels
```

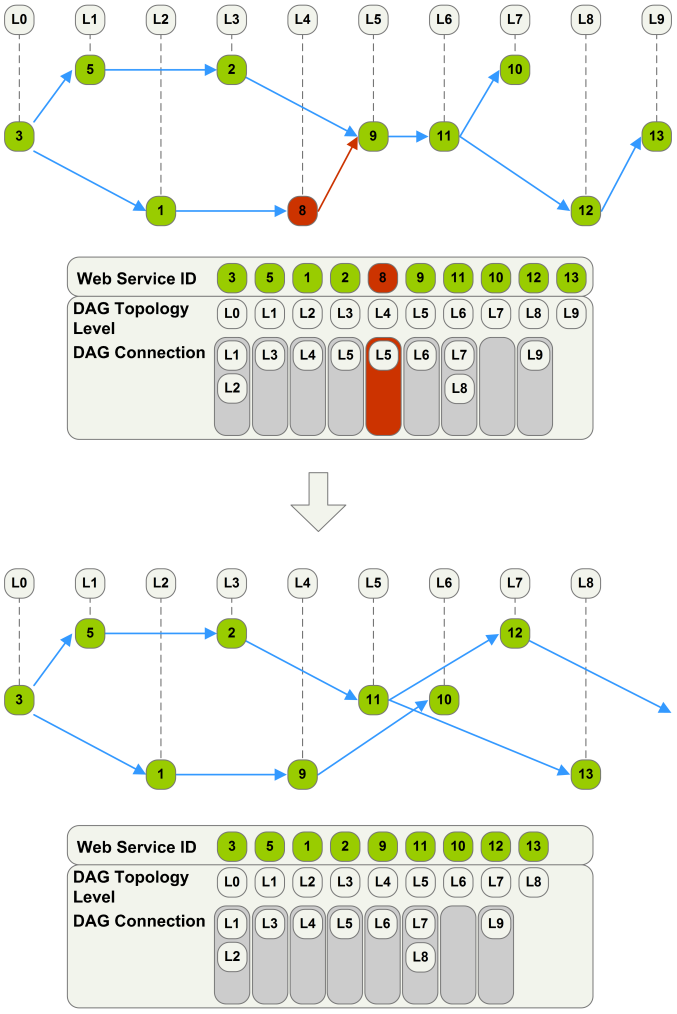

Figure 6.17: Shrinking of the solution by a whole topological level

6.5.4 Repairing of Plans

The repair operator is mapping from a population of size r into a population of children of size r. The operator is formally defined as:

$$Repair^{\xi} : (\mathcal{G}) \rightarrow (\mathcal{G}) \, .$$

During the evolutionary process, different operators are applied on the individuals. These operators create errors that are not in accordance with the general genotypical representation. This means that the mutation and recombination create plans which contain cycles or which contain duplicate web services. The repairing operator repairs the genotypes such that they are acyclic and contain no duplicate web services. For instance, if there are duplicate web services, one service is mutated randomly to another service. Therefore, the repair operator does not solve the problem. It simply changes the genotypes randomly such that they are compliant with the requirements.

This section has explained the evolutionary operators of the algorithm. The next section explains the implementation of the operators and the other components of the planning module.

6.6 Conclusion

This chapter has described the planning module of the mashup framework. The planning uses and evolutionary strategy to perform an efficient search for web service compositions. The algorithm addresses the requirement (see Chapter 2)for an scalable and performant approach for web service composition ([$Req.\ NF4$]). Furthermore, this chapter has described the weighing of different web service composition alternatives based on multiple objectives that relate to the information criteria defined in Chapter 2.

CHAPTER 7

Implementation

In a software architecture, the reduction of the complexity of the system as well as the provision of easy adaptivity are key objectives that could be achieved by the central principle of **loosely coupling**[VAC+05, p.113]. This means that the components of the architecture should be connected as few as possible, which should lead to an easy adaptivity of components independently from the other ones. The mashup framework supports loosely coupling, because the mashup agent is responsible to create an ad-hoc composition of the registered information web services that are invoked to retrieve the mashup data.
Modularity is the general basis of an loosely coupled system and is also related to separation of concerns and information hiding. Information hiding achieves that only those informations are presented to other components which are relevant [VAC+05, p.124]. The modularization of the framework components supports the requirements [*Req. NF*8], [*Req. NF*9] and [*Req. NF*10] of the requirements analysis (see Chapter 2). Each module could be accessed by a defined interface, and thus the internal code of the modules could be changed without affecting the other ones. Furthermore, the framework could be extended by other modules. Each framework module contains functionality that is clearly separated and thus supports the adaptivity, the reuse and the enhancement of the architecture.
The agent program is developed as a portlet web application that could be deployed on the IBM WebSphere Portal. The domain model (TBox of the knowledge base) as well as the web service descriptions are located on a separate application server. Therefore, they could be maintained and changed independently from the application logic of the agent program.
The agent program is based on two main classes. The class "MashupFrameworkPortlet" is an extension of a generic portlet that is responsible to handle external requests. The portlet conforms to Version 2.0 of the Java Portlet Specification [Hep08]. The class

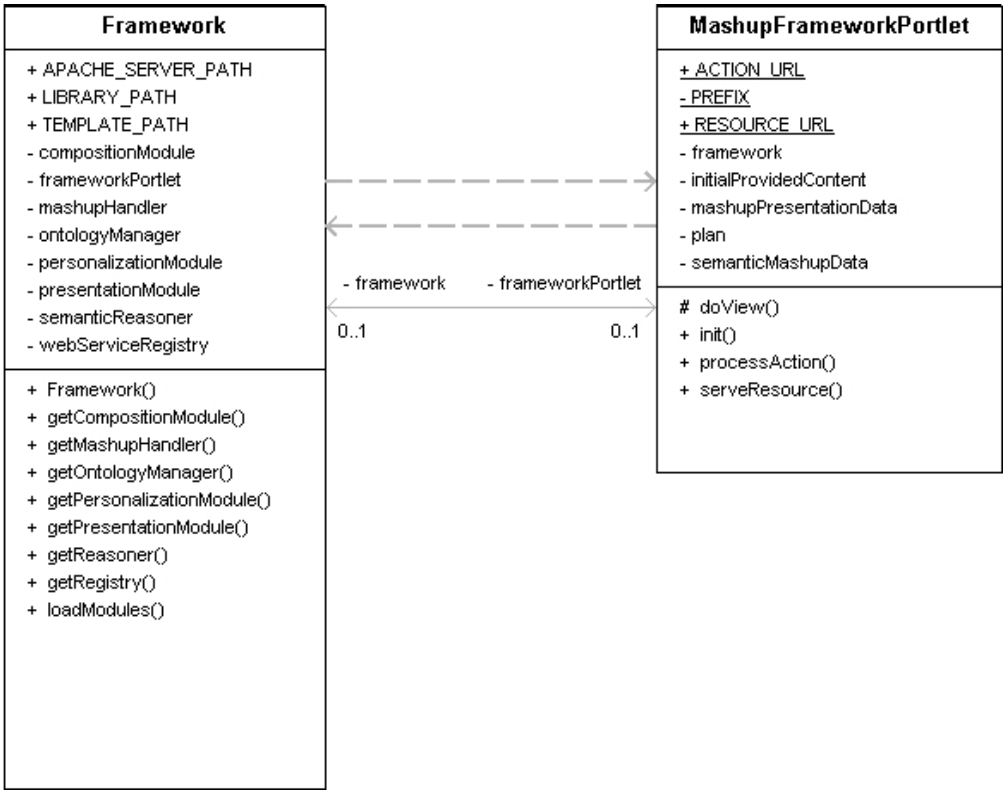

Figure 7.1: Class Diagram Framework Base Classes

"Framework" ensures that the different modules are loaded correctly and maintains the references to them. After the mashup framework is properly initialized, the portlet container could invoke the portlet to handle request from the client.
The agent program could be invoked by an action request [Hep08]. Furthermore, the portlet container could make a serve resource request [Hep08] to get the mashup presentation based on the present ABox state of the agent.
The naming of the components of the architecture, the documentation and the structure of the interfaces etc. is based on a consistent format.
Figure 7.2 outlines the packet structure of the implementation, which is in accordance to the modules defined in the architecture.

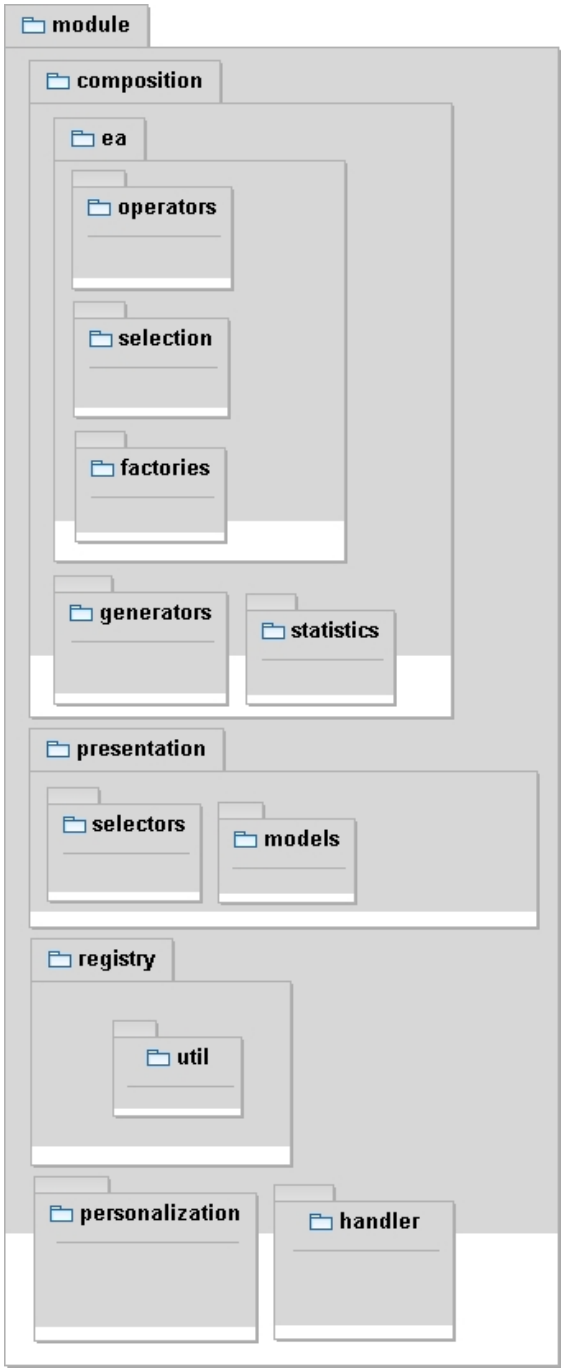

Figure 7.2: Mashup Framework Packet Diagram

7.1 Used Software and APIs

- **IBM WebSphere Portal** [Web] Version 6.1
- **Apache Tomcat** [APAa] Version 6.0: Apache Tomcat is a Servlet container developed by the Apache Software Foundation. It implements the JavaServlet and JavaServer Pages specifications and provides a web server environment to run Java code. The web service descriptions for OWL-S and WSDL as well as the ontologies are served through the web server. Thus, they are not directly part of the mashup agent and could therefore adapted and extended without much effort. Furthermore, the web server is responsible to run a set of self-implemented web services that were written in Java.
- **Jena**: The application utilizes the Jena framework [JEN] that provides an API to handle RDF and OWL. Furthermore, it provides RDF parsers for reading and writing purposes as well as an implementation of SPARQL called ARQ to query RDF documents.
- **Pellet**: The Pellet [PEL] open source[1] reasoner is used in addition to the Jena framework to support extensive OWL-DL reasoning. Pellet supports a variety of reasoning services and optimization technigues.
- **JAXB** [JAX] Version 2.1: The Java Architecture for XML Binding (JAXB) is used to map Java classes to a XML serialization. This means that Java objects could be marshalled into a XML serialization and vice versa.
- **Apache Axis2/Java** [APAb] Version 1.4: Apache Axis2 is a core engine for web services. Apache Axis2 supports SOAP 1.1 and SOAP 1.2 as well as RESTful web services (see Requirement [*Req.F*4].
- **SAWSDL4J** [SAW]: The API is an implementation of the SAWSDL specification and extends the WSDL4J API.
- **WSDL4J** [WSD]: The Web Service Description Language for Java Toolkit (WSDL4J) allows the creation, representation, and manipulation of WSDL documents.
- **Xalan-Java** [XAL] Version 2.7.0: Xalan-Java is an XSLT processor that could transform XML documents into other XML document types. Xalan is used for the XLST transformation of the SAWSDL schema mappings.

[1] Clark & Parsia LLC `http://clarkparsia.com/support` provides also commercial support contracts for Pellet.

The Eclipse 3.3 IDE[2] has been utilized for the development of the web application. The semantic documents has been developed with Protégé[3].

7.2 Planner Implementation

The evolutionary planner has been implemented in Java, in accordance with the other modules of the architecture. The classes are part of the package of the planner. The utilization of an existing Java API for evolutionary algorithms has been considered and discarded, because the complex genetic encoding requires the implementation of new operators and weighting functions. Therefore, it seemed to be suitable to implement the algorithm independent from an existing system.

Figure 7.3 and Figure 7.4 show the main classes of the planning engine.
The main logic of the evolutionary algorithm is encapsulated in the "EvolutionaryAl-

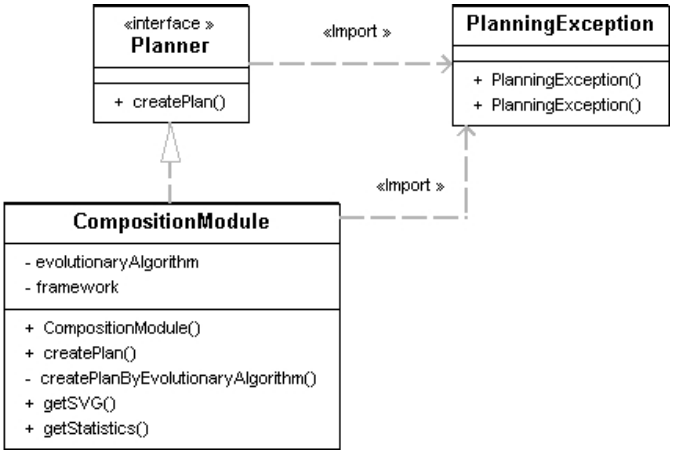

Figure 7.3: Class Diagram of the Planner

gorithm" class. In accordance to the previous explanations, the encoding of the DAG is implemented in the class "AdjacencyListChromosome". The class "WeightingCalculator" is responsible for the performance measurement of a specific plan.
The evolutionary algorithm drives the optimization based on the evolutionary factors for selection, mutation and recombination. Each of the operators is implemented in a related class.

[2] http://www.eclipse.org/
[3] http://protege.stanford.edu/

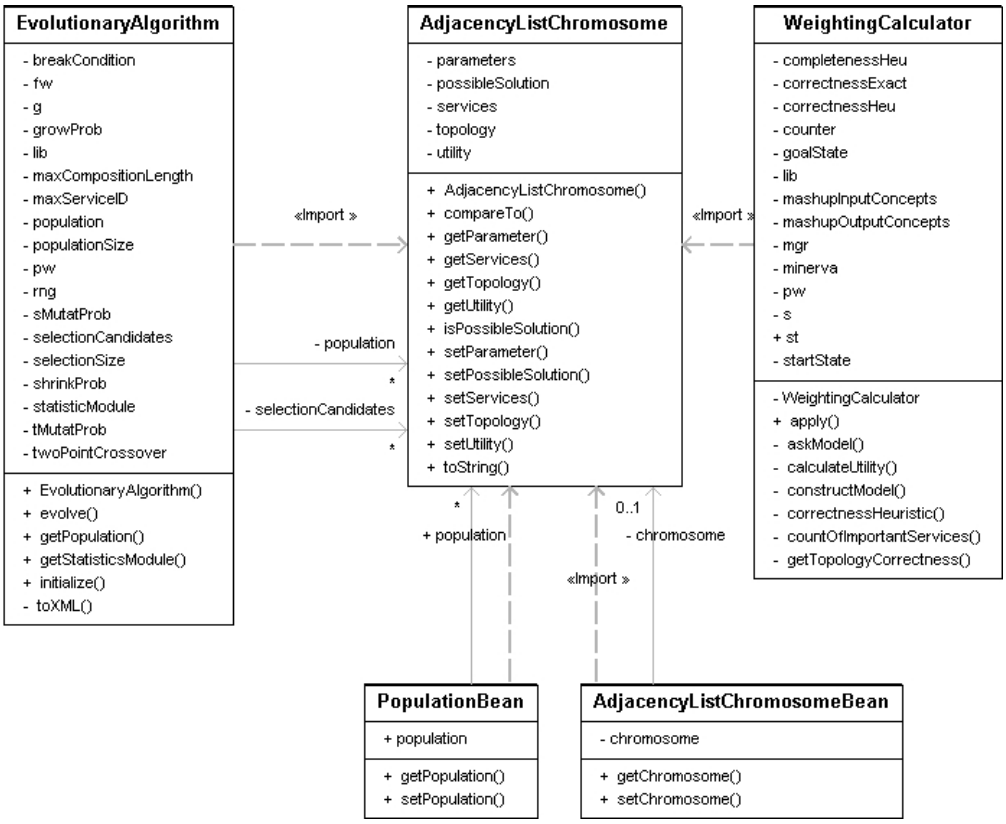

Figure 7.4: Class Diagram Evolutionary Algorithm

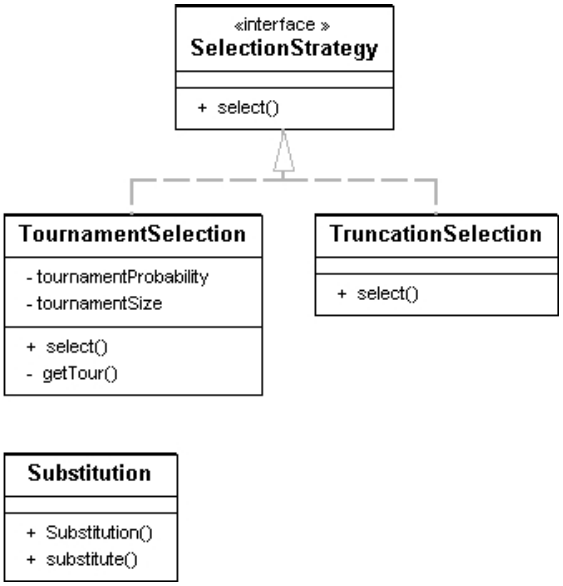

Figure 7.5: Class Diagram Selection

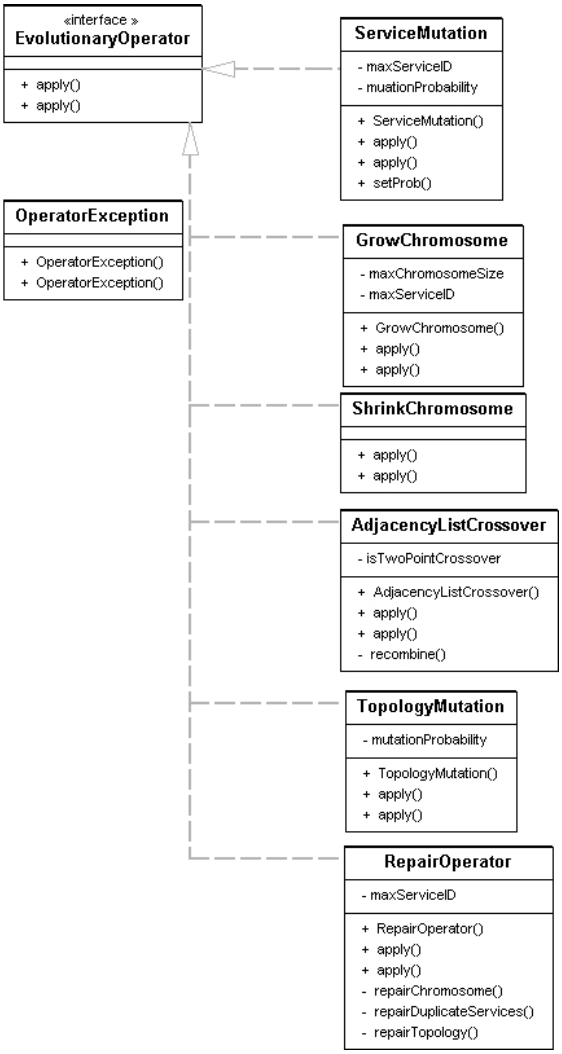

Figure 7.6: Class Diagram Operators

Random numbers are used throughout the algorithm to generate the initial population and to manipulate it. The random numbers are always derived from **one** Java random number generator that creates numbers with a uniform distribution. Therefore it is possible to set so called seed values. A seed values specifies exactly the sequence of numbers that are created randomly. Therefore, such a seed value is important to run the algorithm with the same start conditions for different parameters. This enables an correct comparison of the different results.

The previous figures and explanations outlined shortly the classes of the the planning package. A detailed description of the classes could be found in the Java documentation of the implementation.

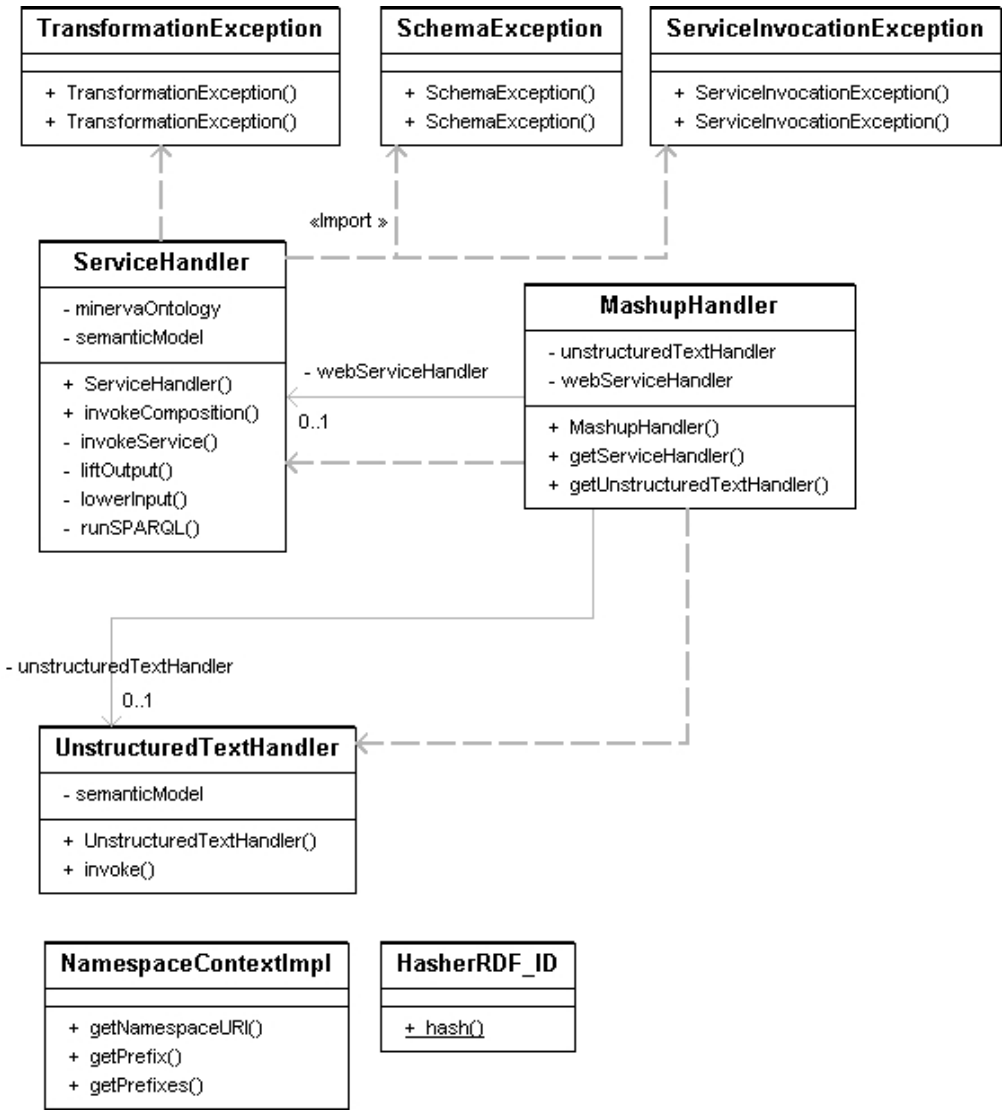

Figure 7.7: Class Diagram Mashup Handler

7.3 Mashup Handler

The "Mashup Handler" is responsible for the invocation of the unstructured text analysis web service as well as the "Service Handler". The functionality is encapsulated in two associated classes (see Figure 7.7).
The "UnstructuredTextHandler" class contains a simple wrapper of the Calais web service. It uses an Axis2 "ServiceClient" that is configured by the public available service description[4].
The "Service Handler" class uses also an Axis2 "Service Client". Furthermore, it main-

[4] `http://api.opencalais.com/enlighten/?wsdl`

tains the $ABox$ state during the plan execution and is responsible for the lifting an lowering of the data.
The class "HasherRDF_ID" is utilized in the lifting schema mappings of the WSDL files. The Xalan XSLT processor provides the hash function as an extension function to the XSLT documents. Since, the instances of the ontology are generated by the XSLT transformations, the hasher is important to achieve unique URI for the different instances of the $ABox$.

7.4 Web Service Registry

The registry is based on two classes: "WebServiceRegistry" and "WebService". The class "WebService" defines the attributes of a web service that are maintained by the registry. The class "WebServiceRegistry" gives access to the list of all web services. The preprocessing of the OWL-S descriptions assumes that the web services do not change their functional parameters, because these changes have to be propagated to the OWL-S descriptions and the registry as to be updated. However, such an dynamic change could not been found to be present at existing real world services. The interfaces of these web services are static. Thus a frequent change of the IOPEs (and therefore of the OWL-S description) of a specific web service at runtime could be neglected.
The present registry does not detect web services which are down in the meanwhile. Thus, it does not deactivate them automatically. However, this could be implemented as an extension and does not affect the feasibility of this approach. For instance, a predefined message could be send at different times to check the availability[5].

[5]This approach seems to be of course only preferable for free web services

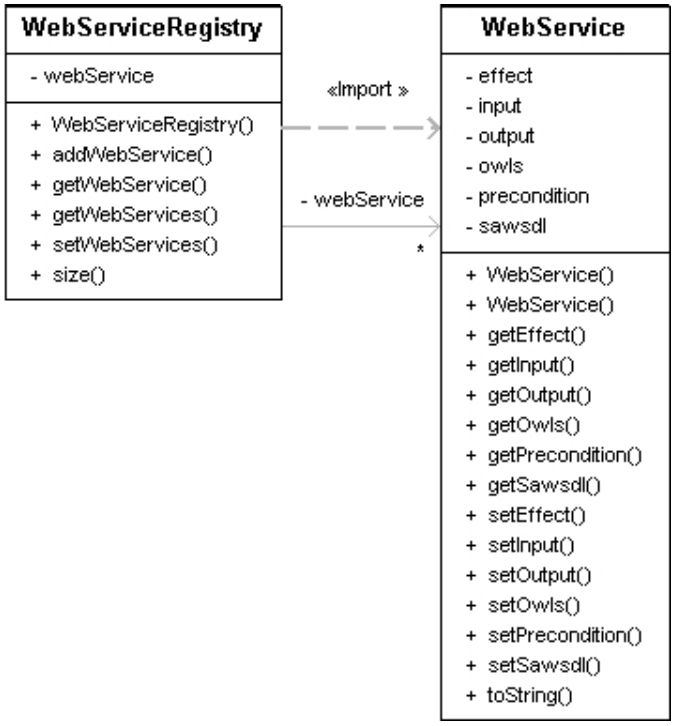

Figure 7.8: Class Diagram Web Service Registry

7.5 Presentation Module

The presentation module generates the mashup presentation based on a model-driven approach. The model driven approach defines several presentation models such as table or map. The presentation models contain the the data of the specific presentation style. Furthermore, a presentation model could have a set of sub models to support nested visualizations. For instance, a mapping model typically is supported by a set of point models, which are added as sub models to the map model. In addition, the points could also have several sub information represented by different kinds of presentation models (e.g. point contains table models). The visualization of each presentation model is defined in a template and could be generated by a template engine. For instance, a map model could be used to generate a GoogleMap or a YahooMap. The type of map only depends on the template that has been selected. Figure 7.9 shows the class diagram of the implemented presentation models.
The relevant data of the presentation models is retrieved by different selectors.
A selector makes a SPARQL query on the RDF data of the mashup (the $ABox$ of the knowledge base). The selectors could be separated in instance selectors and data selectors. Instance selectors return the instances of a specific concept. For example, the company selector returns the URIs of all company instances. The list of these instances is then given to a data selector (e.g. table selector) that retrieves all the information for this instance and adds it to the presentation model.
The previous explanations pointed out an approach to generate dynamically a presentation for RDF data. The difficulty of the generation relies in a suitable transformation of the RDF triples into an artifact that could be consumed by a human agent. Since the presentation module is not in the direct focus of this thesis, it has been only developed to denote the general usability of the framework.

7.6 Conclusion

This chapter has described the implementation of the mashup framework. The implementation of the proposed approach was subject of an evaluation that is described in the next chapter.

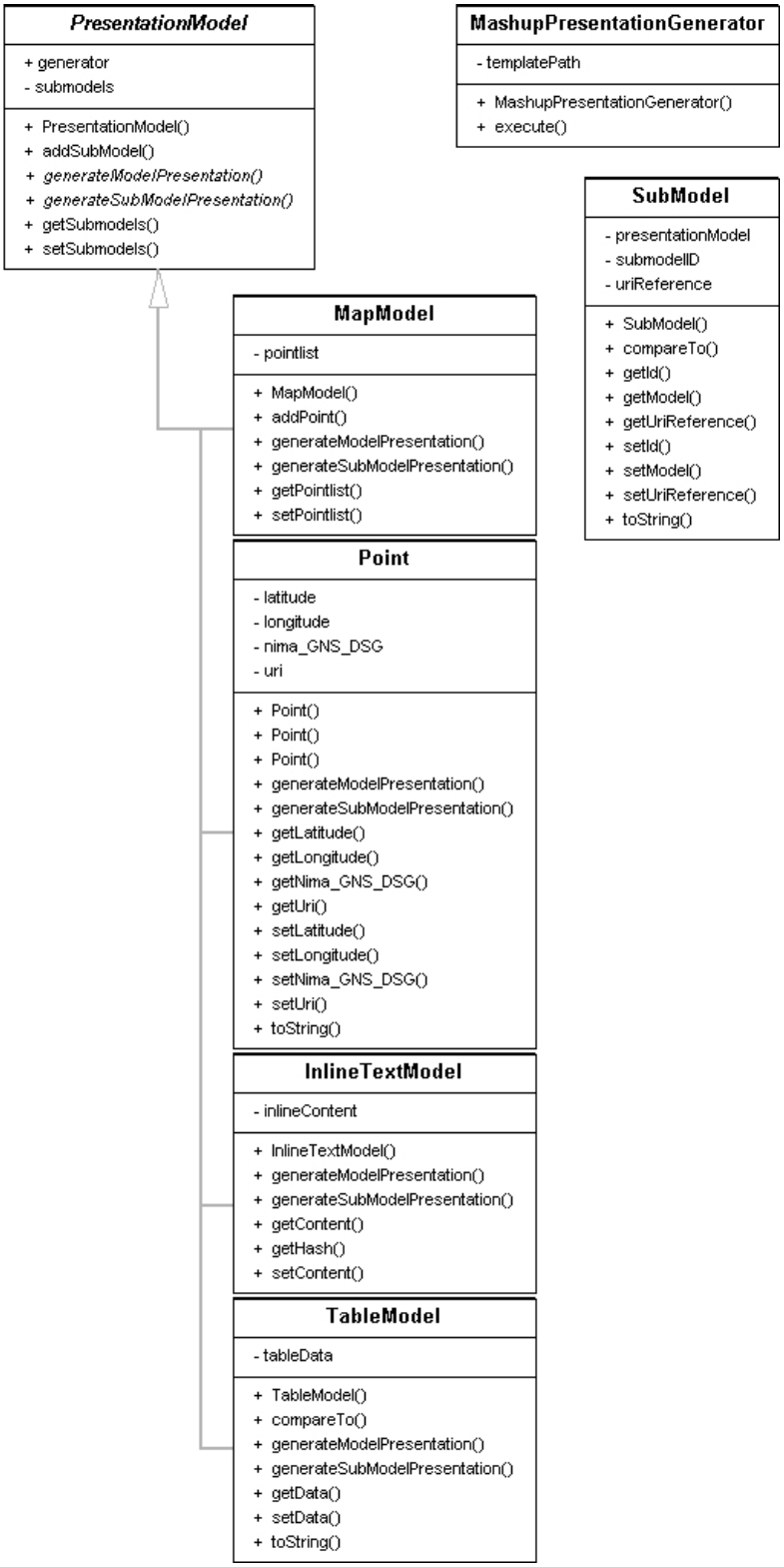

Figure 7.9: Class Diagram Presentation Models

CHAPTER 8

Evaluation

The proposed framework for automatic generation of semantic mashups has been subject of an evaluation. Section 8.1 describes the conditions of the evaluation such as the used web service test collection, hardware as well as the random number generator. As explained in Chapter 6, the planning engine is based on a stochastic hill-climber that traverses the search space by an evolutionary strategy. It can be not guaranteed that the algorithm finds a solution, because the random generation and manipulation of states does not guarantee that every state of the state space will be considered. Therefore the planner has been part of a quantitative evaluation, which is described in Section 8.2. Besides the planning of the information gathering, the invocation of created plans is an important part of this framework, which is investigated in Section 8.3. Section 8.4 concludes this chapter with a short summary about the results of the evaluation.

8.1 Evaluation Conditions

This section describes important conditions that have to be considered in the evaluation of the framework as well as in the comparison with other frameworks.

8.1.1 Semantic Web Service Test Collection

The investigation of the planning and execution of the mashup agent has to be based on a web service test collection. The requirements on the test collection are versatile, but are based on the stated requirements of Chapter 2. The web service test collection:

- has to contain semantic web service descriptions for each web service.
- has to contain description of web services based on WSDL to enable the configuration of the web service client of the agent.

- has to contain SAWSDL annotations for lifting and lowering schema mappings to achieve the data mediation between the web services.
- has to contain SOAPful and RESTful web services to utilize both important types.
- has to be large to evaluate the scalability of the planning approach.
- should utilize financial and related web services that support the stated use cases.

The manual creation or a large test collection that adheres all requirements is very time consuming, because the correct semantic description of existing web services as well as their pretesting needs a lot of time. In addition, it would be useful to utilize a standard test collection that allows the comparison of different planning and execution approaches. Unfortunately, there is no big available repertory of **different** web service test collections. This is in accordance with Küster and König-Ries [KKR08], who stated that the past investments in test collections does not reflect the huge research activities in semantic web services.
The Online Portal for Semantic Services (OPOSSum) [KKRK08] has been created to provide a comprehensive semantic web service repository. Furthermore the existing web service test collections are mainly based on OWL-S and could not be seen as a standard, which makes it difficult to test it against other SWS description languages [KKRK08]. However, an evaluation of the framework is needed to investigate the feasibility, scalability and performance of this approach. Therefore OWL-S has been selected as the semantic web service description language (see Chapter 5).
This thesis proposes the following approach for the creation of the test collection, which is based on a set of 1000 web services descriptions. 990 of the descriptions has been retrieved from OPOSSum [KKRK08] and are described by OWL-S 1.1. This part of the test collection is abstract. This means it contains a huge number of semantic descriptions, but do not relate to WSDL descriptions of existing web services. This is a problem, because the execution of the framework could only be tested by real world web services. Therefore the other web services descriptions has been created manually. They are based on OWL-S 1.2 and SAWSDL, because the definition of preconditions and results in SPARQL is not available in OWL-S 1.1. The OWL-S 1.2 descriptions refer to real SOAPful and RESTful that are used to test also the execution of the plans.
The test collection has also some assumptions that have to be stated at this point. Since the OWL-S 1.1 and 1.2 inputs, outputs, preconditions and results are preprocessed by the mashup agent registry, both types could be handled. However, the OWL-S 1.1 web service descriptions of the test collection contain no preconditions and results defined in SPARQL. Therefore, the exact calculation (simulation) of the plan is not possible, if the considered plan contains such web services. Nevertheless, the heuristic planning

process could be executed, because it relies only on the input and outputs of a web service.
Another problem with the test collection is based on the different utilized ontologies. The retrieved OWL-S 1.1 web service descriptions use not the terms of the domain ontology of the framework to described inputs and outputs. Instead, they are based on several other ontologies. These are loaded in addition to the domain ontology into the ontology manager of the agent. It has to be considered, that this influences the subsumes inference of the planning process, because the different ontologies are not mediated with the framework ontology. Therefore, the subsumes relation will find no subclass relations between web services described by **resources of different ontologies**. Nevertheless, the descriptions are considered in the planning process. Moreover, this **does not reduce the state space** that has to be traversed by the evolutionary algorithm, because the application does not know in advance that the outputs of a specific web service do not subsume with other resources.
So far, the test collection has only minor influence on the evaluation of the framework, because the planning problems could be stated in a way that the correct solution could only incorporate web services that could be also invoked. However, the algorithm does not consider the case that the test collection contains several equal web services. This means web services that provide the same functionality, but different QoS attributes. The current implemented weighting function (f_{WC}) is not able to distinguish between such equal web services and thus will select both. Therefore it is better to extend the weighting function such that it is able to incorporate the QoS preferences of the user. This should avoid an selection and composition of a set of equal web services if the penalty for too long plans is high enough.
As outlined in Chapter 6, the user model and the formal concretion of user preferences is a requirement that is not part of this thesis and could be done in the future work. The proposed web service test collection has therefore no multiple equal web services.

8.1.2 Used Hardware

The evaluation of the mashup agent has been performed on an Intel(R) Core 2 CPU 6400 with 2.13Ghz and 2GB RAM. The main memory for the execution of the algorithm has been restricted to a maximum of 1024 MB.

8.2 Analysis of the Planning Module

This section analyses the planning process. The next section describes the resulting state space and defines the different considered planning problems.

8.2.1 Definition of Planning Problems

8.2.1.1 State Space

The search algorithm traverses the discrete state space by stochastic hill-climbing combined with an evolutionary strategy. The state space is increasing as more and more web services become available. It is assumed that each web service could be only ones in the plan. The state space is defined by the set of all permutations of atomic web services registered at the library, if only the resulting partial ordering of the DAG is considered. For instance, if the registry contains 1000 web services $S = 0, 2, \ldots, 999$ then each of the 1000! permutations is a state that could be a possible solution of the problem. Thus, the discrete state space would have $1000! = 4,0e + 2567$ states. Instead if the registry contains only 100 web services, the number of states would be $100! \sim 9,3e + 157$. It is therefore important to evaluate the scalability of the agent.

8.2.1.2 Planning Problem [P1] - Traveling

```
  <name>travel</name>
  <mashupInputs>http://proton.semanticweb.org/2005/04/protonu#City</
      mashupInputs>
  <mashupInputs>http://proton.semanticweb.org/2005/04/protonu#Company</
      mashupInputs>
  <mashupOutputs>http://proton.semanticweb.org/2005/04/protonu#Hotel</
      mashupOutputs>
  <mashupOutputs>http://proton.semanticweb.org/2005/04/protonu#Street</
      mashupOutputs>
  <mashupOutputs>http://proton.semanticweb.org/2005/04/protont#longitude</
      mashupOutputs>
  <mashupOutputs>http://proton.semanticweb.org/2005/04/protont#latitude</
      mashupOutputs>
  <mashupPrecondition></mashupPrecondition>
  <mashupEffect></mashupEffect>
</task>
```

Listing 8.1:]Task Description - Planning Problem [P1]

The first planning problem addresses the traveling use case (Listing 8.1). The goal of the task is to get information about hotels, streets as well as longitude and latitude information. The available mashup inputs are also stated. However, if the provided content contains no such information, the planning process considers this. The mashup preconditions of this task model could be used to denote the triple required and generated triple structures. Since, the algorithm does not use the exact completeness (it uses the exact correctness), these values are not stated.
The mashup outputs define the goal of the planning process based on the description of resources, which are concepts such as *Hotel* and *Street* as well as roles (properties) such as *longitude* and *latitude*.

8.2.1.3 Planning Problem [P2] - Company Research

This planning problem is more complex than the first one, because it requires to find intermediate web services that do not directly contribute to the goal information state.

```
<task>
  <name>research</name>
  <mashupInputs>http://proton.semanticweb.org/2005/04/protonu#City</
      mashupInputs>
  <mashupInputs>http://proton.semanticweb.org/2005/04/protonu#Company</
      mashupInputs>
  <mashupOutputs>http://127.0.0.1/ontology/minerva-portals.owl#StockQuote</
      mashupOutputs>
  <mashupOutputs>http://127.0.0.1/ontology/minerva-portals.owl#StockSymbol</
      mashupOutputs>
  <mashupOutputs>http://127.0.0.1/ontology/minerva-portals#hasWikiPage</
      mashupOutputs>
  <mashupOutputs>http://proton.semanticweb.org/2005/04/protonu#NewspaperIssue
      </mashupOutputs>
  <mashupPrecondition></mashupPrecondition>
  <mashupEffect></mashupEffect>
</task>
```

Listing 8.2:]Task Description - Planning Problem [P2]

8.2.2 Performance and Scalability Analysis

This analysis evaluates the performance and scalability of the algorithm for the planning problems [P1] and [P2]. For each problem the planner has been randomly executed to solve by the planner fifty times. The parameters of the algorithm have been the following:

- Library Size: 1000
- Population Size: 8
- Selection Size: 8
- Selection Type: Tournament Selection
- Crossover: Two-Point Crossover
- Service Mutation Probability: 0.2
- Topology Mutation Probability: 0.7
- Grow Probability: 0.3
- Shrink Probability: 0.3

Figure 8.1, 8.2 outline the results for the different problems. The diagrams shows the weighting of the best individual of the population in dependence to the number of generations. In both problems the algorithm converges for the first generations relatively fast. For the former problem the algorithm comes in all cases to a correct and complete solution. In the most runs it needed maximum 1000 generations. Some outliers need more than the 1000 generations.
The latter problem is more complex and the algorithm also needs more generations to come to a solution. For two runs the algorithm has been stopped at the maximum of 10000 generations, the curves are not shown in the diagram. The other runs solved the problem in the average in 25 seconds, which seems to be acceptable.
In 3 runs the solutions achieved not the utility of 80, because they were correct and complete, but had an additional not necessary web service.
It could be therefore concluded that the algorithm is able to solve such problems and that they could be solved in an acceptable time regarding the a web service registry of 1000 web services. Furthermore, the more complex a problem is the more generations are required by the algorithm.

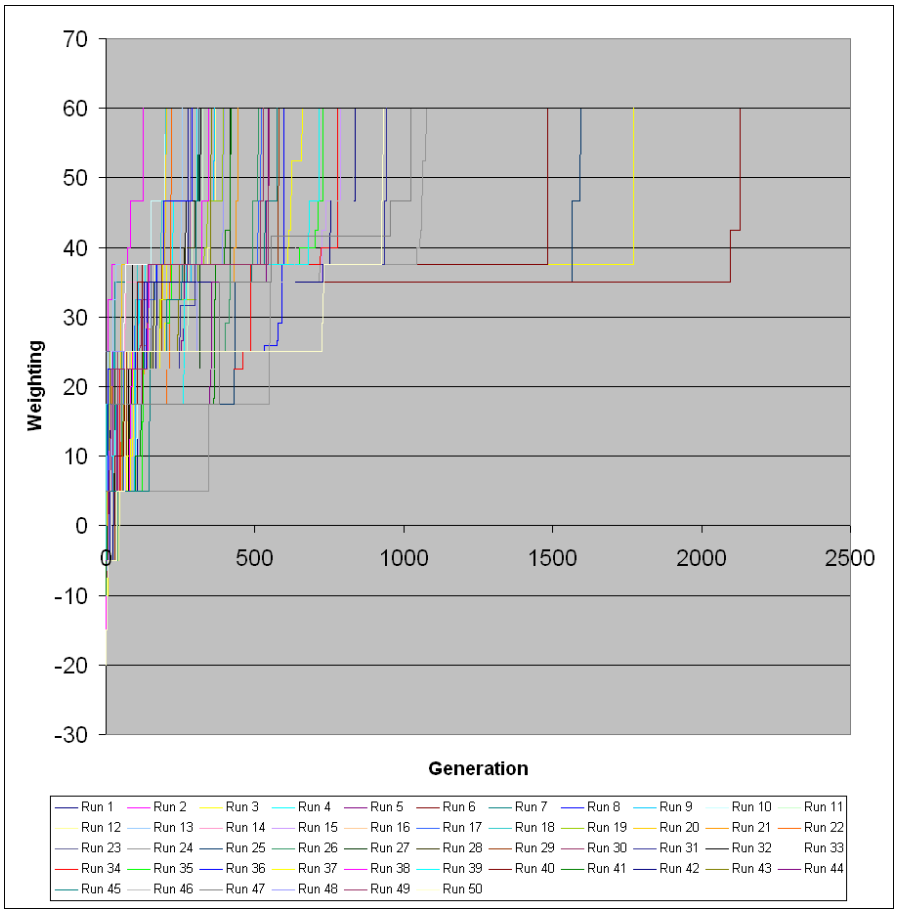

Figure 8.1: Performance Analysis of Problem [P1]

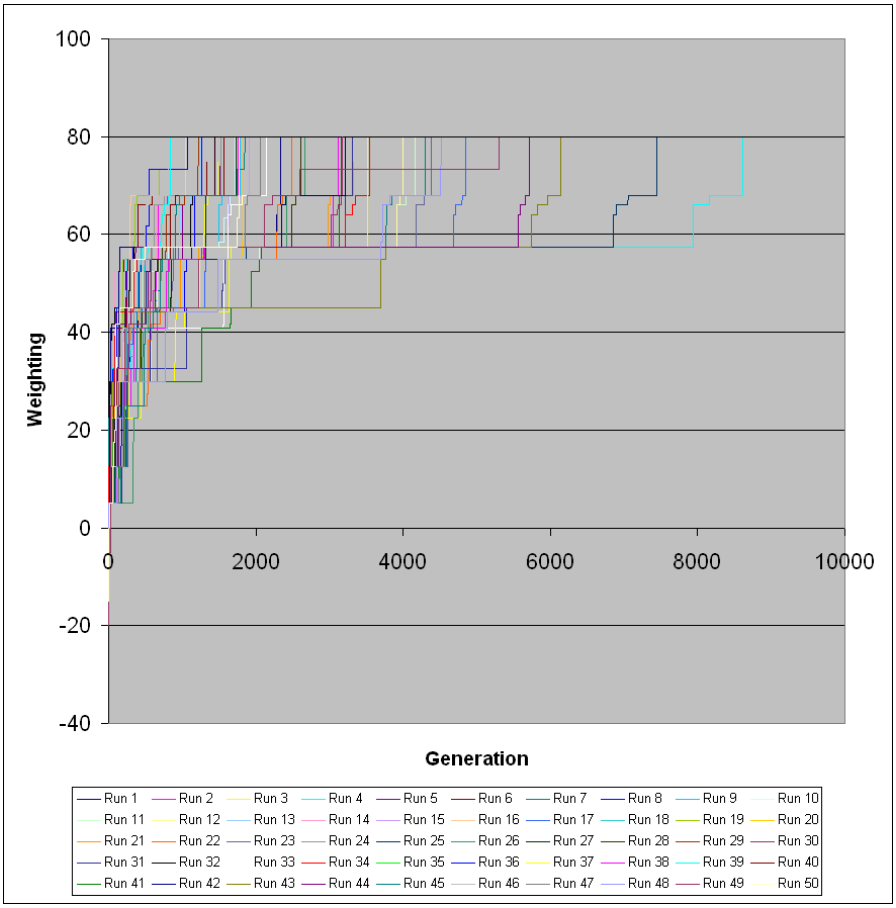

Figure 8.2: Performance Analysis of Problem [P2]

The scalability in dependence to the registry size is very important. The following analysis is based on planning problem [P2] and investigates the scalability of the agent for the registry size. The planner is executed iteratively with registry sizes between 150 and 1000 registered web service descriptions. For each registry size the planner is run for 9 different seed values of the random number generator to ensure the comparability. Figure 8.3 shows the median of the count of evaluated states in dependence to the registry size. The count of evaluated states is calculated by the number of invocations of the weighting function until a solution has been found. For instance, the weighting function could evaluate 1000 states (plans) of the states space and then have a correct and complete solution. In this case the measure is 1000. Figure 8.3 shows the median of the values of the different runs.
It could be concluded, that an increasing registry size requires the evaluation of more states. This was expected, however, it is important to note that the state space increases much more faster than the algorithm has to evaluate more states. Therefore, the algorithm has a good scalability even for this large registry sizes. For instance, for 100 web services the state space has $100! \sim 9,3e+157$ states, whereas for 1000 web services the state space has $1000! \sim 4,0e+2567$ states. At the same time, the median of required states has multiplied by round about five. Furthermore, in all cases the algorithm has found a correct and complete solution.

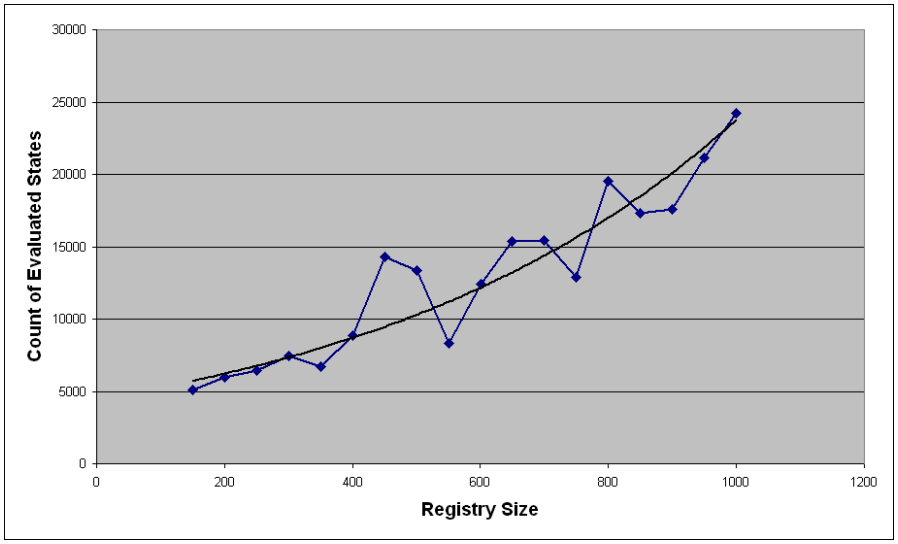

Figure 8.3: Scalability Analysis of Problem [P2]

8.2.3 Parameter Sensitivity Analysis

The parameter sensitivity analysis investigates the change of the performance of the planning algorithm in accordance to the variation of one parameter. This seems to be important to evaluate the planning process in different parameter setting.

The basic parameters of the algorithm have been the following:

- Library Size: 1000
- Population Size: 8
- Selection Size: 8
- Selection Type: Tournament Selection
- Crossover: Two-Point Crossover
- Service Mutation Probability: 0.2
- Topology Mutation Probability: 0.7
- Grow Probability: 0.3
- Shrink Probability: 0.3

The considered parameters for the sensitivity analysis are:

- Population size $pSize \in [2, 20]$
- Selection size $sSize \in [2, 20]$
- Service mutation probability $sMutProb \in [0, 1]$
- Topology mutation probability $tMutProb \in [0, 1]$

The sensitivity analysis is computed for a set of 5 different seed values of the random number generator to ensure the comparability of the results.
The mutation probability is an important parameter that should be not round about 0.2 (see Figure 8.4). If the mutation probability is to big the algorithms has to evaluate more states. For service mutation probabilities greater or equal than 0.5 the algorithm converges not to a solution state. This is acceptable, because as explained in Chapter 6, big mutations also go along with great negative impacts, and this is confirmed by this diagram.

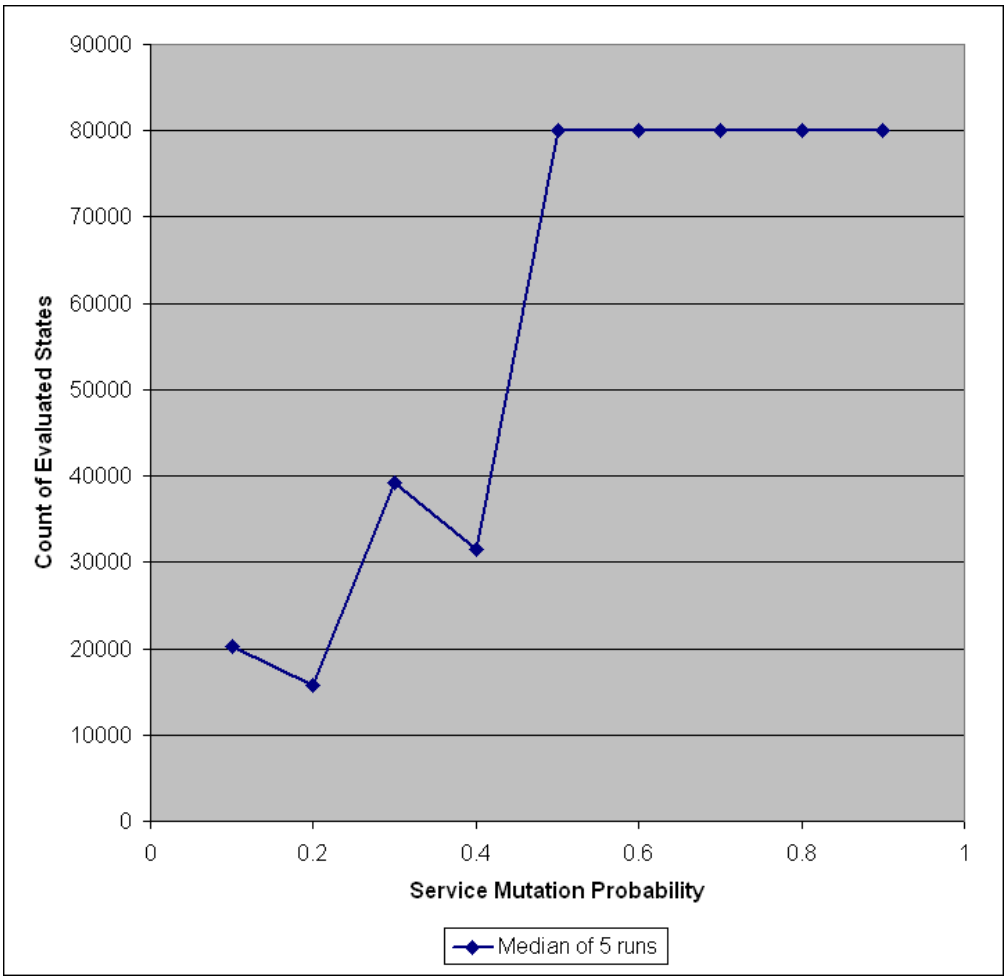

Figure 8.4: Sensitivity Analysis of Service Mutation Probability of Problem [P2]

Figure 8.5 shows the topology mutation analysis. The topology mutation should be not to small, because otherwise the algorithm comes not to a solution. This is not straightforward. However, the topology mutation does not change the services itself, it changes only their placement. If the topology mutation is to small the algorithm is not able to come quickly to different coordinations and thus such an higher probability drives the optimization better.

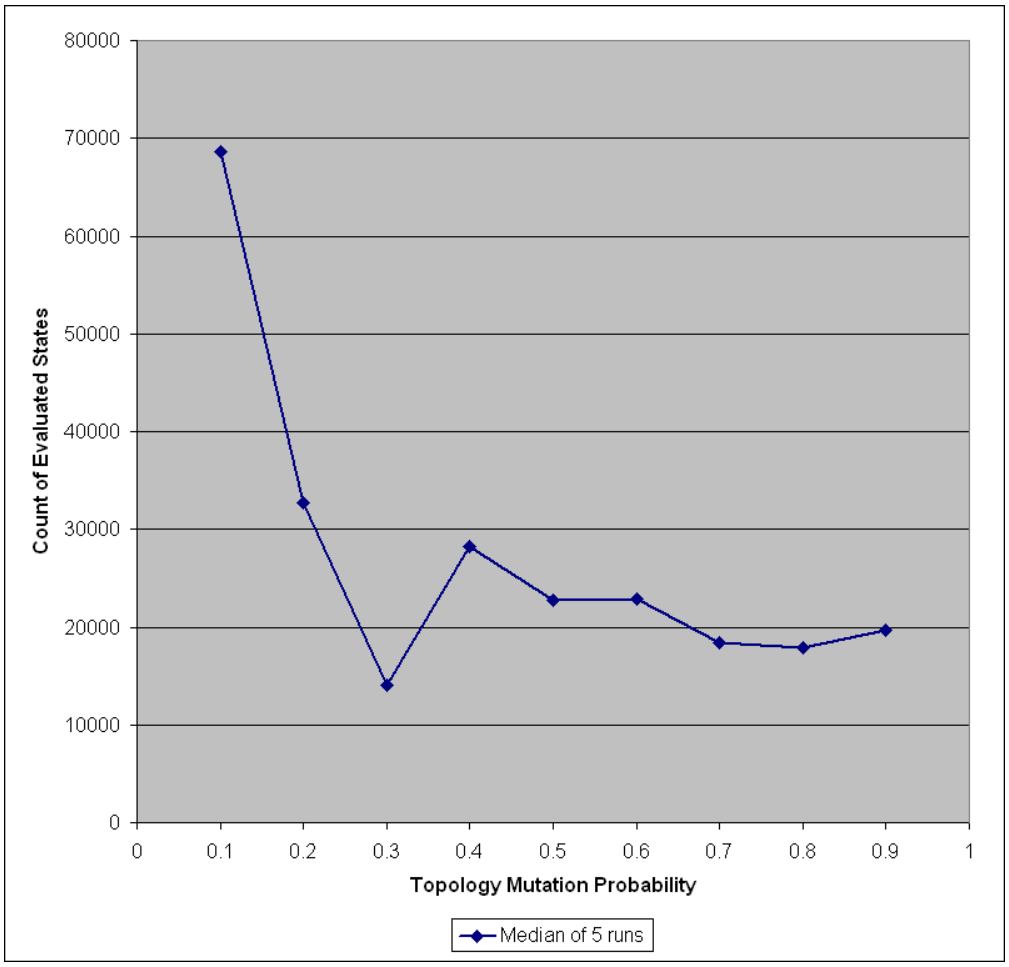

Figure 8.5: Sensitivity Analysis of Topology Mutation Probability of Problem [P2]

In Figure 8.6 and Figure 8.7, the sensitivity of the population size and the selection size are investigated. However, there could be no clear trend detected. Thus, a higher population size not necessary increases the performance of the algorithm. Therefore, there could be not clear trend stated.

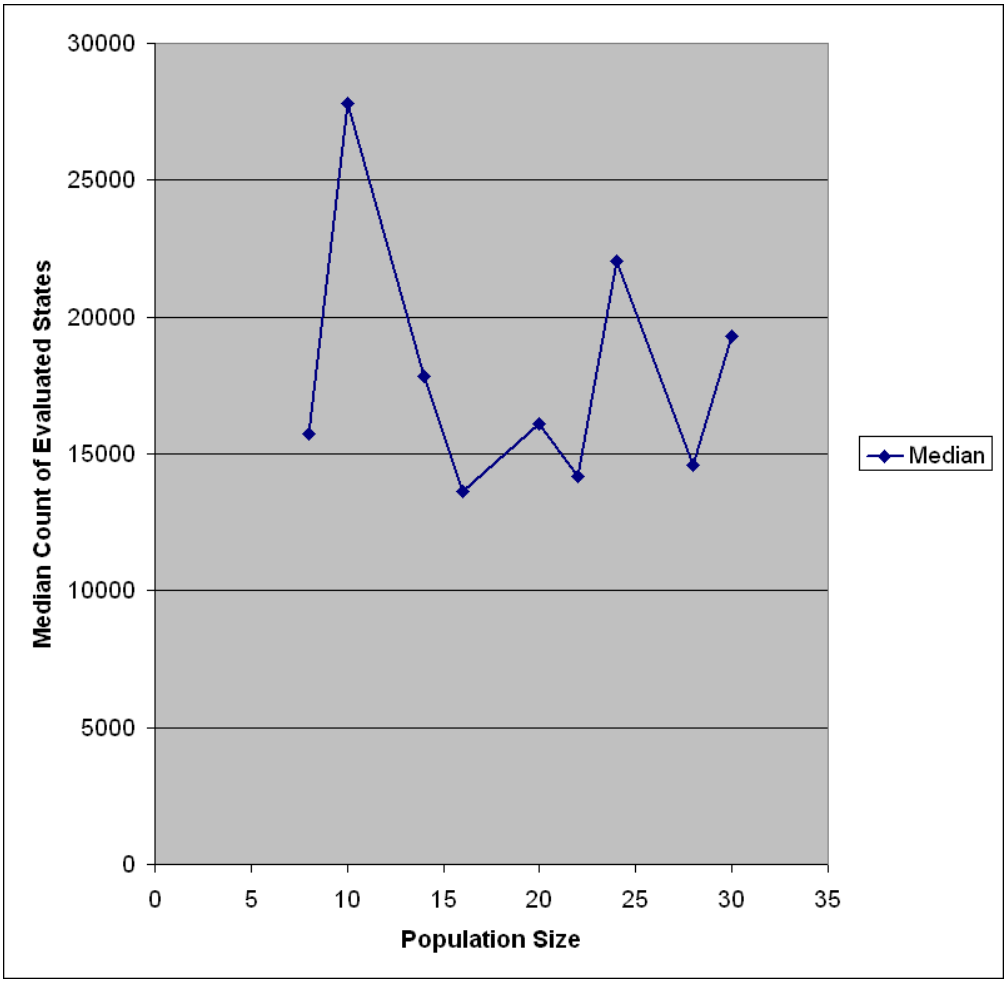

Figure 8.6: Sensitivity Analysis of Population Size of Problem [P2]

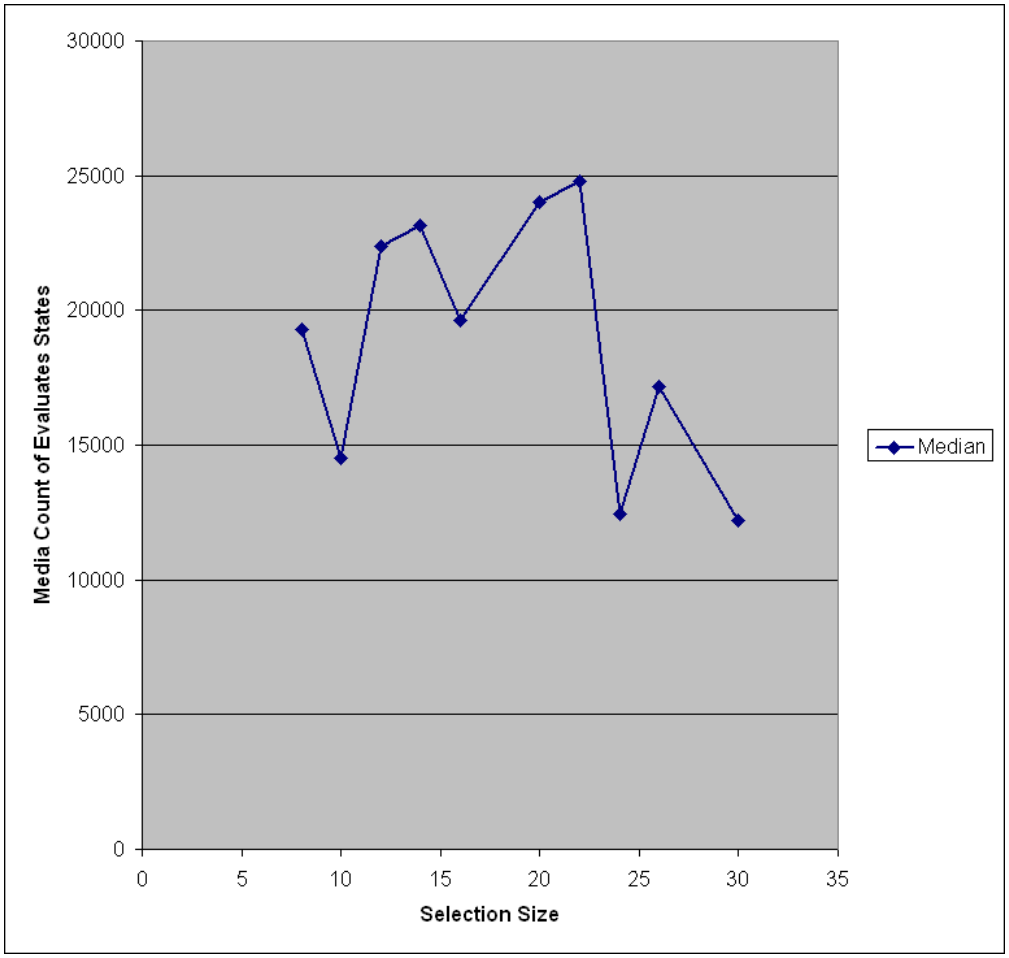

Figure 8.7: Sensitivity Analysis of Selection Size of Problem [P2]

8.2.4 Diversity Analysis

The usage of a population of solution candidates offers the possibility to search parallel at different points of the state space. This advantage is only present, if the different solution candidates (states) are **not identical**. However, the presence of several solution candidates itself makes no evidence for **diversity**.
Diversity could be measured in different ways and has to be adapted to the problem. Popular measures are the Shanon entropy or the substring diversity [Wei07, p.62-63]. For the existing problem a **substring** oriented diversity seems to be appropriate, because it adheres also the diversity of subparts of the plans. Furthermore, we compute the diversity on the partial order that is given by the web service keys of the genotype. Thus, the topological connections of the DAG are neglected. This seems to be suitable, because finally the partial ordering of the web service is the only information that is needed to invoke the services, since the execution stage is based on the SAWSDL lifting and lowering. Nevertheless, the topological information are needed in the planning process to process the **heuristic** functions.

For a population P the diversity is defined as

$$Diversity_{Substring}(P) = \frac{s * \#\left(\bigcup_{1 \leq i \leq s} Substring\left(G^i.R\right)\right)}{\sum_{1 \leq i \leq s} \#Substring\left(G^i.R\right)}$$

.

The function $Substring(G.R)$ returns a set of possible substrings of the partial ordered list of web service identifiers of the genotype. If the population is converged the diversity is 1, which is the minimum of the function. The following example calculates the substring diversity for a predefined population P. Since, the substring diversity is only measured on the partial list of web services, the concrete topological information of the example population are neglected, to simplify the notation.

$P = \langle\langle 10,2,3,8\rangle, \langle 1,2,3\rangle\langle 3,8,5\rangle\rangle$
$Substring(\langle 10,2,3,8\rangle) = \{\langle 10\rangle, \langle 2\rangle, \langle 3\rangle, \langle 8\rangle, \langle 10,2\rangle, \langle 2,3\rangle, \langle 3,8\rangle, \langle 10,2,3\rangle, \langle 2,3,8\rangle, \langle 10,2,3,8\rangle\}$
$Substring(\langle 1,2,3\rangle) = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 1,2\rangle, \langle 2,3\rangle, \langle 1,2,3\rangle\}$
$Substring(\langle 3,8,5\rangle) = \{\langle 3\rangle, \langle 8\rangle, \langle 5\rangle, \langle 3,8\rangle, \langle 8,5\rangle, \langle 3,8,5\rangle\}$

$$\bigcup_{1 \leq i \leq s} Substring\left(G^i.R\right) =$$
$$\{\langle 10\rangle, \langle 2\rangle, \langle 3\rangle, \langle 8\rangle, \langle 1\rangle, \langle 5\rangle, \langle 10,2\rangle, \langle 2,3\rangle, \langle 3,8\rangle, \langle 1,2\rangle, \langle 8,5\rangle, \}$$
$$\cup \{\langle 10,2,3\rangle, \langle 2,3,8\rangle, \langle 1,2,3\rangle, \langle 3,8,5\rangle, \langle 10,2,3,8\rangle\}$$

$$Diversity_{Substring}(P) = \frac{3 * 16}{22} \sim 2,18$$

A converged population is a special population in which each solution candidate is equal. Thus the diversity measure is the smallest in this case. This is an sight for the end of the optimization or if the present optima is not global the algorithm has converged to early.
The diversity has been analysed for three random runs of the algorithm on problem [P2]. The parameters of the problem has been the following:

- Library Size: 1000
- Population Size: 14
- Selection Size: 8
- Selection Type: Tournament Selection
- Crossover: Two-Point Crossover
- Service Mutation Probability: 0.2
- Topology Mutation Probability: 0.7
- Grow Probability: 0.3
- Shrink Probability: 0.3

At the start of the algorithm the diversity is high throughout all runs, which is expectable, because the random start population should have an high diversity. Then the algorithm relatively fast converges towards a specific direction, where the best weighting of the population is round about 60. This also decreases the diversity of the population, which is expectable because the agent now has already some problem relevant web services more than one times in the population. The recombination operator leads to similar substructures in child genotypes and therefore the diversity decreases. The diversity the is always between 7 and 5. This means the population is never converged. The reason is very simple. Since the environment selection does not allow a specify genotype more than ones, the population does not converge completely. However, this is no problem and all runs come to a final complete and correct solution.
Interestingly, the diversity changes not between the generation $\sim$ 1200 and $\sim$ 2200. This

could be an evidence that the population is trapped in a local maxima where it could not escape for a longer time. This means the newly generated plans are not able to displace the existing ones. However, finally the algorithm comes to a complete and correct solution.

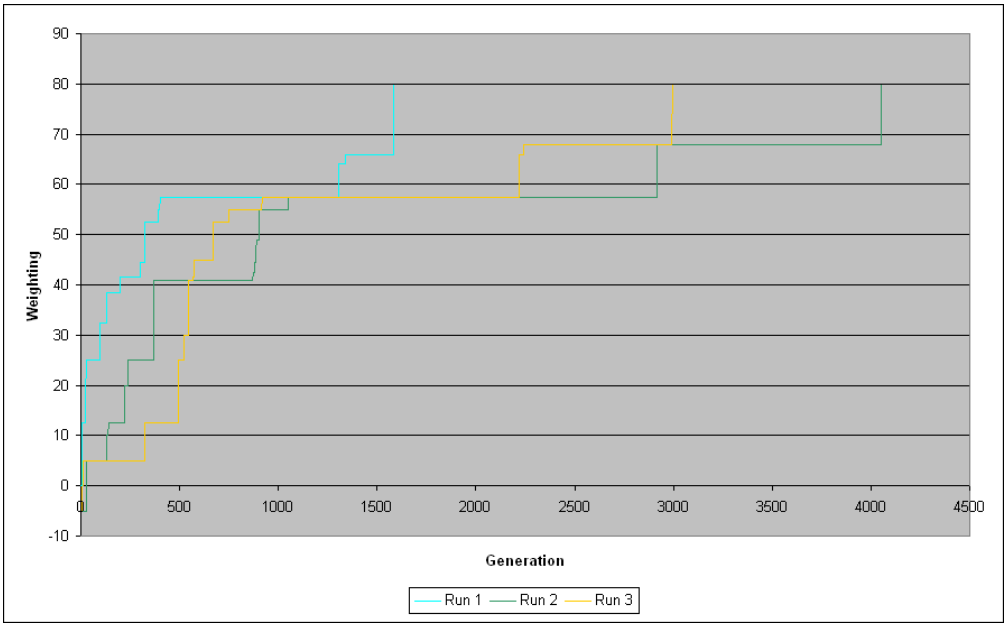

Figure 8.8: Diversity Analysis Problem [P2] - Weighting of three considered runs

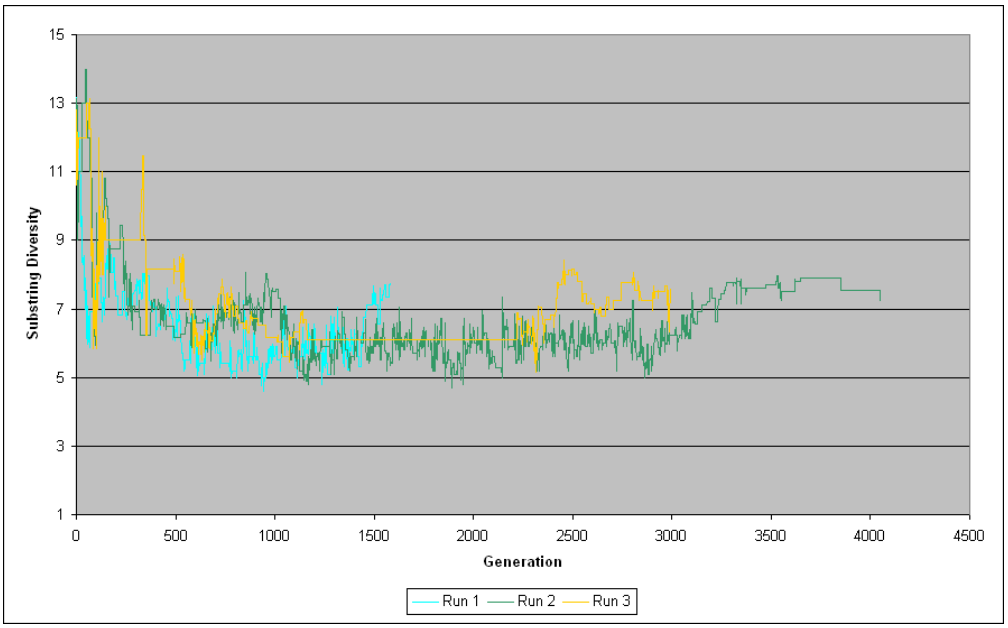

Figure 8.9: Diversity Analysis Problem [P2] - Diversity of three considered runs

8.3 Analysis of the Execution Module

This section analysis the execution of the provided plans. The content is based on a financial text that is unstructured and provided by a portlet of the web portal (see Figure 8.10). Furthermore, the task is assumed to be to find company information. The text is given to the mashup framework by the user interface and the planning process is started. The results of the planning process is a plan (see Figure 8.11)that could be invoked by the mashup agent. Figure 8.12 shows a part of the resulting mashup data that is provided to the user.

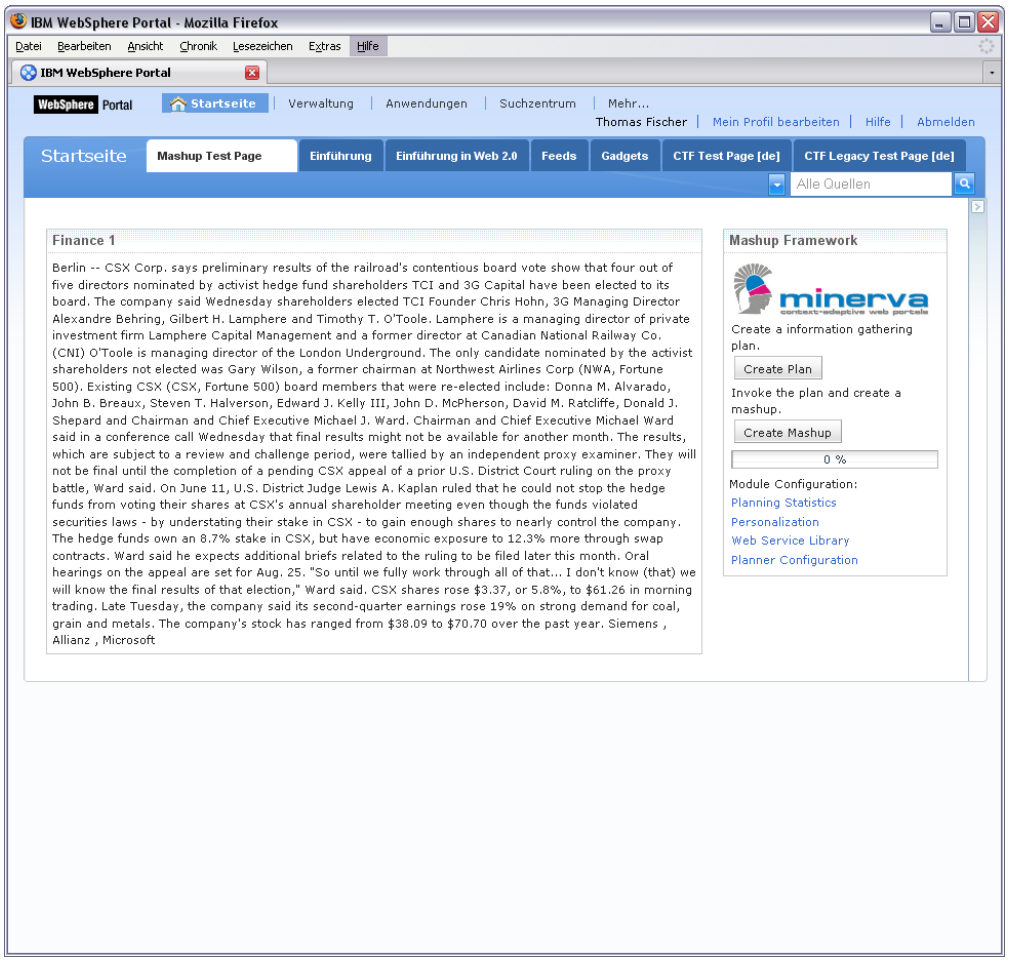

Figure 8.10: Execution Screenshot 1

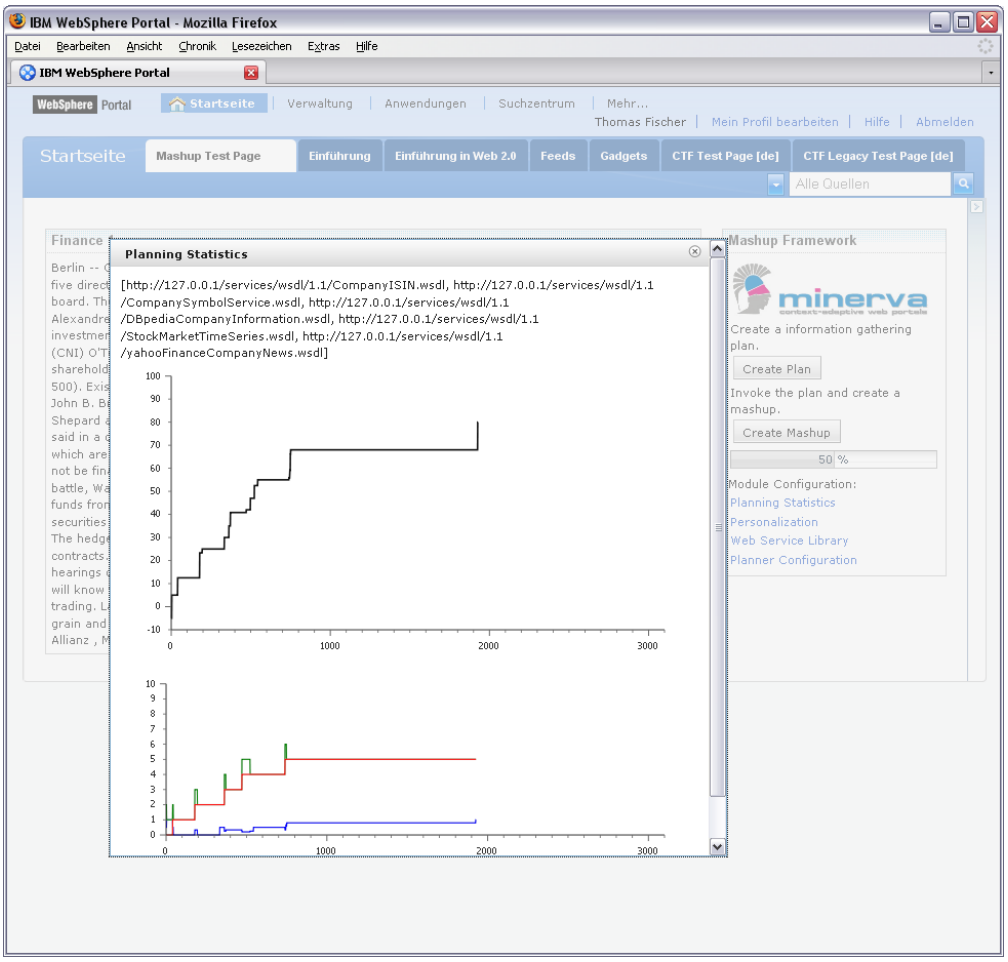

Figure 8.11: Execution Screenshot 2

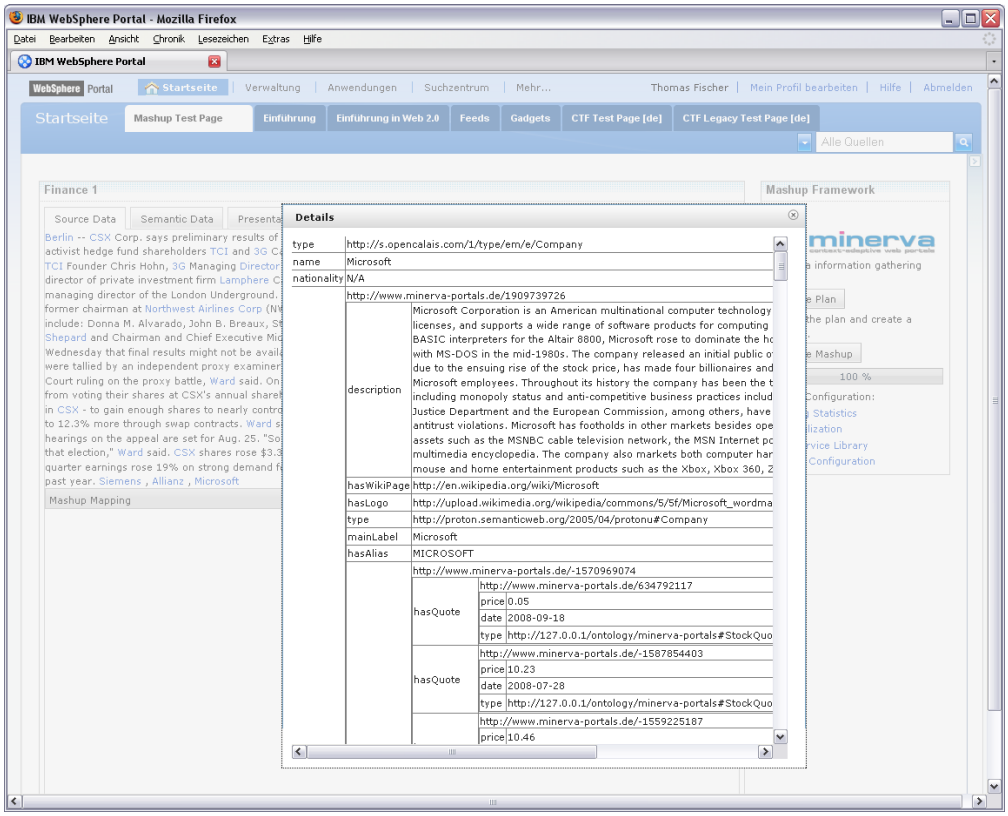

Figure 8.12: Execution Screenshot 3

This section outlined shortly the execution of the mashup framework and a possible result of the information gathering process. The amount of available information thereby depend on the ability of the web services to serve the requests. Thus, even if there is a suitable plan, it is not guaranteed that all needed information could be retrieved. Therefore, the QoS attributes are important to achieve a differentiation among the web services, which could be part of future work.

8.4 Conclusion

This chapter evaluated the planning module of the mashup framework as well as the execution of the create plans.
The planning module was evaluated by two different predefined problems. A problem related to a task of a user. The planner is able to compositions of web services. The planner could not guarantee that if finds a solution, even if there is one solution in the state space. This is based on the stochastic nature of the algorithm. However, in the evaluation the planner has solved the problems very efficiently and only in two runs of

one considered test set of 50 runs, the algorithm could not create a complete and correct plan. Furthermore, the algorithm is scalable in dependence to the amount of registered web services. In fact, this scalability was one of the main objectives for the selection of this planning approach.
The generated plans can be also invoked dynamically, which has been tested several times and was shown in this thesis by an example.

CHAPTER 9

Summary and Future Work

The objective of this thesis was to automatically generate mashups that provide background information for a given content.
The thesis has first stated several requirements for an automatic generation of mashups. The investigation about mashup patterns and existing mashup frameworks has shown that the existing mashup frameworks have fundamental limitations. These limitations let to the insight that an automatic generation of mashups is needed. Furthermore, these existing frameworks are also not able to fulfill the requirements of this thesis.
The theoretical investigation outlined the different research areas that are related to an automatic generation of mashups.
This thesis has proposed a framework for automatic generation of semantic mashups in web portals. The framework is based on an intelligent agent that is able to automatically compose different information web services. The information web services provide the background data that is utilized in the mashup.
The web service composition planning module is an important component of the architecture. This thesis proposed its own approach to web service composition planning to achieve an scalable solution for the mashupframework. The planning algorithm is based on an evolutionary algorithm that efficiently traverses through the search space of the planning problem.
The feasibility and scalability of this approach have been investigated in an evaluation of the framework. The evaluation confirmed that automatic generation of mashups by web service composition is feasible. Furthermore, the performance of the evolutionary planner tends to be good also on large sets of available web services.

In the future work the framework could be extended at different points. First, theres is a need for an user model that could be utilized to achieve the automatic adaption of the framework based on the user interests, expertise and tasks.

Furthermore, the RDF presentation has to be extended to provide richer analyse capabilities and data manipulation options.
In the planning module the incorporation of the QoS preferences into the planning process seems to be important to adhere all information criteria. Furthermore, the performance of the algorithm may could be further increased through better heuristic weighing functions for correctness and completeness of plans.

References

[AB96] J. Corera A. Bernaras, I. Laresgoiti. Building and reusing ontologies for electrical network applications. In *European Conference on Artificial Intelligence (ECAI 96) Budapest, Hungary*, page 298âĂŞ302, 1996.

[AFM+] Rama Akkiraju, Joel Farrel, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web Service Semantic - WSDL-S. Website. Available online at `http://www.w3.org/Submission/WSDL-S/`; visited on August 28th 2008.

[AKTV07] Anupriya Ankolekar, Markus Krötzsch, Thanh Tran, and Denny Vrandecic. The two cultures: mashing up web 2.0 and the semantic web. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 825–834. ACM, 2007.

[APAa] Apache Tomcat. Website. Available online at `http://tomcat.apache.org`; visited on August 28th 2008.

[APAb] Axis2. Website. Available online at `http://ws.apache.org/axis2/`; visited on August 28th 2008.

[BAB] Ben Adida and Mark Birbeck. RDFa Primer: Bridging the Human and Data Webs. Website. Available online at `http://www.w3.org/TR/xhtml-rdfa-primer/`; visited on September 2th 2008.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 1 edition, 2003.

[BCM+07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory,*

Implementation, and Applications. Cambridge University Press, 2 edition, 9 2007.

[BIM+] Mark Baker, Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Ted Wugofski, and Toshihiko Yamakam. XHTML Basic 1.1. Website. Available online at `http://www.w3.org/TR/xhtml-basic/`; visited on August 28th 2008.

[BKN07] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web: Methods and Strategies of Web Personalization (Lecture Notes in Computer Science)*. Springer, 1 edition, 6 2007.

[BL] Tim Berners-Lee. Realising the Full Potential of the Web. Website. Available online at `http://www.w3.org/1998/02/Potential.html`; visited on May 05th 2008.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, pages 29–37, May 2001. Available online at `http://www.sciam.com/article.cfm?id=the-semantic-web`; visited on August 8th 2008.

[BLP05] Chris Bizer, Ryan Lee, and Emmanuel Pietriga. Fresnel - Display Vocabulary for RDF. Website, 06 2005. Available online at `http://www.w3.org/2005/04/fresnel-info/manual/`; visited on September 2th 2008.

[BN08] Andreas Brodt and Daniela Nicklas. The telar mobile mashup platform for nokia internet tablets. In Alfons Kemper, Patrick Valduriez, Noureddine Mouaddib, Jens Teubner, Mokrane Bouzeghoub, Volker Markl, Laurent Amsaleg, and Ioana Manolescu, editors, *EDBT*, volume 261 of *ACM International Conference Proceeding Series*, pages 700–704. ACM, 2008.

[Boa] RSS Advisory Board. RDF/XML Syntax Specification (Revised). Website. Available online at `http://www.rssboard.org/rss-specification`; visited on August 28th 2008.

[Bor97] W. N. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Twente, Enschede, The Netherlands, 1997.

[BPSM+] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, FranÃğois Yergeau, and John Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). Website. Available online at `http://www.w3.org/TR/xml11`; visited on August 13th 2008.

[BR] Dan Brickley and R.V.Guha. RDF Vocabulary Description Language 1.0RDF Schema. Website. Available online at `http://www.w3.org/TR/rdf-schema/`; visited on September 2th 2008.

[BvHH+] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Website. Available online at `http://www.w3.org/TR/owl-ref/`; visited on March 28th 2008.

[BWG07] Steffen Bleul, Thomas Weise, and Kurt Geihs. Making a fast semantic service composition system faster. In *CEC/EEE*, pages 517–520. IEEE Computer Society, 2007.

[BYRN99] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[CCMW] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Description Language 1.1. Website. Available online at `http://www.w3.org/TR/wsdl`; visited on August 28th 2008.

[CFL+04] Pablo Castells, Borja Foncillas, Rubén Lara, Mariano Rico, and Juan Luis Alonso. Semantic web technologies for economic and financial information management. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *ESWS*, volume 3053 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2004.

[CFLGP03] Óscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Methodologies, tools and languages for building ontologies: Where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64, 2003.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 9 2001.

[CLV07] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer, 2nd ed. edition, 9 2007.

[CMRW] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Service Description Language 2.0 Part1: Core Language. Website. Available online at `http://www.w3.org/TR/wsdl20/`; visited on August 28th 2008.

[CNPZ08] Michael Pierre Carlson, Anne H.H. Ngu, Rodion Podorozhny, and Liangzhao Zeng. Automatic mash up of composite applications. In *Int. Conf on Service-Oriented Computing (ICSOC-08)*, 2008.

[Con] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). Website. Available online at `http://www.w3.org/TR/grddl/`; visited on September 2th 2008.

[dBBD+05] Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta KÃűnig-Ries, Jacek Kopecky, RubÃl'n Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web Service Modeling Ontology (WSMO). Website, 06 2005. Available online at `http://www.w3.org/Submission/WSMO/`; visited on September 2th 2008.

[DM88] Barry A. Devlin and Paul T. Murphy. An architecture for a business and information system. *IBM Systems Journal*, 27(1):60–80, 1988.

[DSS08] Rafal Drezewski, Jan Sepielak, and Leszek Siwik. Generating robust investment strategies with agent-based co-evolutionary system. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *ICCS (3)*, volume 5103 of *Lecture Notes in Computer Science*, pages 664–673. Springer, 2008.

[EBG+07] Rob Ennals, Eric Brewer, Minos Garofalakis, Michael Shadle, and Prashant Gandhi. Intel mash maker: join the web. *SIGMOD Rec.*, 36(4):27–33, 2007.

[EG07a] Robert Ennals and Minos N. Garofalakis. Mashmaker: mashups for the masses. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *SIGMOD Conference*, pages 1116–1118. ACM, 2007.

[EG07b] Robert Ennals and David Gay. User-friendly functional programming for web mashups. In Ralf Hinze and Norman Ramsey, editors, *ICFP*, pages 223–234. ACM, 2007.

[Fab98] Frank J. Fabozzi. *Investment Management (2nd Edition)*. Pearson Education, 2 edition, 10 1998.

[FB98] Xavier Franch and Pere Botella. Putting non-functional requirements into software architecture. In *In, 9th International Workshop on Software Specification and Design*, pages 60–67, 1998.

[FB02] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[FHM05] Michael J. Franklin, Alon Y. Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[FJ95] Donald E. Fischer and Ronald J. Jordan. *Security Analysis and Portfolio Management*. Prentice Hall, 6 fac sub edition, 1 1995.

[FL07] Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema. Website, 08 2007. Available online at `http://www.w3.org/TR/sawsdl/`; visited on September 2th 2008.

[Fou08] Apache Software Foundation. UIMA Overview & SDK Setup. PDF, 04 2008. Available online at `http://incubator.apache.org/uima/downloads/releaseDocs/2.2.2-incubating/docs/pdf/overview_and_setup.pdf`; visited on March 28th 2008.

[Gru93] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[Gua95] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *Int. J. Hum.-Comput. Stud.*, 43(5-6):625–640, 1995.

[Hep08] Stefan Hepper. *Java Portlet Specification Version 2.0*, 01 2008. Available online at `http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html`; visited on August 16th 2008.

[HF08] Volker Hoyer and Marco Fischer. Market overview of enterprise mashup tools. In *Int. Conf on Service-Oriented Computing (ICSOC-08)*, 2008.

[HMK07] David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: Data mash-up tool for casual users. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber,

and Philippe Cudré-Mauroux, editors, *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 2007.

[Hor07] Ian Horrocks. Semantic web: the story so far. In *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 120–125, New York, NY, USA, 2007. ACM.

[Inc07] SERENA Software Inc. MICROS SYSTEMS SEES MACRO QUALITY IMPROVEMENT WITH SERENA. Website, 10 2007. Available online at `http://www.serena.com/docs/repository/products/successes/success-story-MICROS-Systems.pdf`; visited on March 28th 2008.

[ITG08] Cobit 4.1 Excerpt. Website, 2008. Available online at `http://www.isaca.org/AMTemplate.cfm?Section=Downloads&Template=/ContentManagement/ContentDisplay.cfm&ContentID=34172`; visited on August 13th 2008.

[JAX] JAXB. Website. Available online at `https://jaxb.dev.java.net/`; visited on August 28th 2008.

[JEN] Jena - A Semantic Web Framework for Java. Website. Available online at `http://jena.sourceforge.net/`; visited on August 28th 2008.

[KBS+08] Frederik De Keukelaere, Sumeer Bhola, Michael Steiner, Suresh Chari, and Sachiko Yoshihama. Smash: secure component model for cross-domain mashups on unmodified browsers. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *WWW*, pages 535–544. ACM, 2008.

[KC] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Website. Available online at `http://www.w3.org/TR/rdf-concepts/`; visited on August 28th 2008.

[Khi08] Alex Khizhnyak. Apatar Connector Guide. Website, 2008. Available online at `http://www.apatarforge.org/wiki/display/GUI/Apatar+Connector+Guides`; visited on August 8th 2008.

[KKR04] Michael Klein and Birgitta König-Ries. Integrating preferences into service requests to automate service usage. In *First AKT Workshop on Semantic Web Services*, Milton Keynes, UK, Dezember 2004.

[KKR08] Ulrich Küster and Birgitta König-Ries. On the empirical evaluation of semantic web service approaches: Towards common SWS test collections. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC2008)*, Santa Clara, CA, USA, August 2008.

[KKRK06] Ulrich Küster, Birgitta König-Ries, and Michael Klein. Discovery and mediation using diane service descriptions. In *First Workshop of the Semantic Web Service Challenge 2006 - Challenge on Automating Web Services Mediation, Choreography and Discovery*, Palo Alto, California, USA, March 2006.

[KKRK08] Ulrich Küster, Birgitta König-Ries, and Andreas Krug. OPOSSum - an online portal to collect and share sws descriptions. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC2008), Demo Session*, Santa Clara, CA, USA, August 2008.

[KKRM05] Michael Klein, Birgitta König-Ries, and Michael Müssig. What is needed for semantic service descriptions - a proposal for suitable language constructs. *International Journal on Web and Grid Services (IJWGS)*, 1(3/4):328–364, 2005.

[KMP98] Jasna Kuljis, Robert D Macredie, and Ray J Paul. Information gathering problems in multinational banking (uk). In *HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 6*, page 622, Washington, DC, USA, 1998. IEEE Computer Society.

[KSKR05] Ulrich Küster, Mirco Stern, and Birgitta König-Ries. A classification of issures and approaches in service composition. In *First International Workshop on Engineering Service Compositions (WESC05)*, pages 00–00, Amsterdam, Netherlands, December 2005.

[KVG08] Jacek KopeckÃ¡, Thomas Vitvar, and Karthik Gomadam. Microwsmo. PDF, 03 2008. Available online at `http://cms-wg.sti2.org/TR/d12/v0.1/20080724/d12_v01_20080724.pdf`; visited on September 28th 2008.

[Lew05] Dirk Lewandowski. Web Information Retrieval: Technologien zur Informationssuche im Internet. DGI-Schriften zur Informationswissenschaft 7, Frankfurt am Main, 2005. Available online at `http://www.durchdenken.de/lewandowski/web-ir/`; visited on May 05th 2008.

[LGS07] Jon Lathem, Karthik Gomadam, and Amit P. Sheth. SA-REST and

(S)mashups : Adding Semantics to RESTful Services. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 469–476, 2007.

[Lip07] William E. Lipner. The future of online market research. *Journal of Advertising Research*, 47:142–146, 06 2007.

[LQH+06] Fen Lin, Lirong Qiu, He Huang, Qing Yu, and Zhongzhi Shi. Description logic based composition of web services. In Zhong-Zhi Shi and Ramakoti Sadananda, editors, *PRIMA*, volume 4088 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2006.

[Lus01] Markus Lusti. *Data Warehousing und Data Mining: Eine Einführung in entscheidungsunterstützende Systeme (Springer-Lehrbuch)*. Springer, 2. überarb. u. erw. aufl. edition, 10 2001.

[LYGW06] Yingjie Li, Xueli Yu, Lili Geng, and Li Wang. Research on reasoning of the dynamic semantic web services composition. In *Web Intelligence*, pages 435–441. IEEE Computer Society, 2006.

[MB00] Harry Mucksch and Wolfgang Behme. *Das Data Warehouse Konzept. Architektur - Datenmodelle - Anwendungen.* Dr. Th. Gabler Verlag, 9 2000.

[MBD+] David Martin, Mark Burstein, Grit Denker, Daniel Elenius, Joseph Giampapa, Drew McDermott, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S 1.2 Pre-Release. Website. Available online at `http://www.ai.sri.com/daml/services/owl-s/1.2/`; visited on September 13th 2008.

[MBH+04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. Website, 11 2004. Available online at `http://www.w3.org/Submission/OWL-S/`; visited on August 13th 2008.

[McK] McKinsey. How businesses are using Web 2.0: A McKinsey Global Survey. Website. Available online at `http://www.mckinseyquarterly.com/How_businesses_are_using_Web_20_A_McKinsey_Global_Survey_1913_abstract`; visited on May 19th 2008.

[Mer] Duane Merrill. Mashups: The new breed of Web app. Website. Available online at http://www.ibm.com/developerworks/library/x-mashups.html; visited on March 28th 2008.

[MJC+07] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin Dong, David Ko, Cong Yu, Alon Halevy, and Google Inc. Web-scale data integration: You can only afford to pay as you go. In *In Proc. of CIDR-07*, 2007.

[MKB06] Brad A. Myers, Andrew Jensen Ko, and Margaret M. Burnett. Invited research overview: end-user programming. In Gary M. Olson and Robin Jeffries, editors, *CHI Extended Abstracts*, pages 75–80. ACM, 2006.

[MLA08] Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. Building mashups for the enterprise with sabre. In *Int. Conf on Service-Oriented Computing (ICSOC-08)*, 2008.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Website, 02 2004. Available online at http://www.w3.org/TR/owl-features/; visited on March 28th 2008.

[O'R] Tim O'Reilly. What is Web 2.0. Website. Available online at http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html; visited on May 19th 2008.

[PEL] Pellet. Website. Available online at http://pellet.owldl.com; visited on August 28th 2008.

[PS] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Website. Available online at http://www.w3.org/TR/rdf-sparql-query/; visited on September 13th 2008.

[Rad97] Robert C. Radcliffe. *Investment: Concepts, Analysis, Strategy (5th Edition)*. Addison Wesley, 5 edition, 1 1997.

[RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. Website, 09 1999. Available online at http://www.w3.org/TR/html4; visited on March 28th 2008.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 2 edition, 2003.

[RS98] D. Fensel R. Studer, V.R. Benjamins. Knowledge engineering: principles and methods. *Data and Knowledge Engineering*, (25):161–197, 1998.

[RSS07] RSSBus. An RSS feed of favorite photos from each of your Flickr contacts. Website, 08 2007. Available online at `http://blog.rssbus.com/AnRSSFeedOfFavoritePhotosFromEachOfYourFlickrContacts.aspx`; visited on Aug 26th 2008.

[RSS08] Rssbus description. Website, 2008. Available online at `http://www.rssbus.com/docs/rssbus.pdf`; visited on August 10th 2008.

[RTA07] Erhard Rahm, Andreas Thor, and David Aumueller. Dynamic fusion of web data. In *XSym*, pages 14–16, 2007.

[SAM+08] David E. Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: data mashups for intranet applications. In Jason Tsong-Li Wang, editor, *SIGMOD Conference*, pages 1171–1182. ACM, 2008.

[Sav07] Reijo Savolainen. Filtering and withdrawing: strategies for coping with information overload in everyday contexts. *Journal of Information Science*, 33(5):611–621, 2007.

[SAW] SAWSDL4J. Website. Available online at `http://lsdis.cs.uga.edu/projects/meteor-s/opensource/sawsdl4j/`; visited on August 28th 2008.

[SBD03] Eric Schwarzkopf, Mathias Bauer, and Dietmar Dengler. Towards intuitive interaction for end-user programming. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 287–289, New York, NY, USA, 2003. ACM.

[Sch04] Armin Scholl. *Planung und Entscheidung*. Vahlen Franz GmbH, 6 2004.

[SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley and Sons, 1 edition, 2005.

[SHS08] Michael Schumacher, Heikki Helin, and Heiko Schuldt, editors. *CASCOM: Intelligent Service Coordination in the Semantic Web (Whitestein Series in Software Agent Technologies and Autonomic Computing)*. Birkhäuser Basel, 1 edition, 9 2008.

[SHSG07] Marwan Sabbouh, Jeff Higginson, Salim Semy, and Danny Gagne. Web mashup scripting language. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1305–1306, New York, NY, USA, 2007. ACM.

[SK98] Atsushi Sugiura and Yoshiyuki Koseki. Internet scrapbook: Automating web browsing tasks by demonstration. In *ACM Symposium on User Interface Software and Technology*, pages 9–18, 1998.

[SOA] SOAP Version 1.2. Website. Available online at `http://www.w3.org/TR/soap12-part0/`; visited on August 28th 2008.

[SRKR97] B. Swartout, P. Ramesh, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies, 1997. AAAI Symposium on Ontological Engineering, Stanford.

[ST08] Nathalie Steinmetz and Ioan Toma. WSML Language Reference. Website, 08 2008. Available online at `http://www.wsmo.org/TR/d16/d16.1/v1.0/`; visited on August 28th 2008.

[SZM+02] Monica M. C. Schraefel, Yuxiang Zhu, David Modjeska, Daniel Wigdor, and Shengdong Zhao. Hunter gatherer: interaction support for the creation and management of within-web-page collections. In *WWW*, pages 172–181, 2002.

[TAR07] Andreas Thor, David Aumueller, and Erhard Rahm. Data Integration Support for Mashups Sixth International Workshop on Information Integration on the Web, iiWeb, Vancouver, Canada, 2007.

[TG02] Chiu-Che Tseng and Piotr J. Gmytrasiewicz. Real time decision support system for portfolio management. In *HICSS*, page 79, 2002.

[TSK08] Rattapoom Tuchinda, Pedro A. Szekely, and Craig A. Knoblock. Building mashups by example. In Jeffrey M. Bradshaw, Henry Lieberman, and Steffen Staab, editors, *Intelligent User Interfaces*, pages 139–148. ACM, 2008.

[TSP+07] Junichi Tatemura, Arsany Sawires, Oliver Po, Songting Chen, K. Selcuk Candan, Diviyakant Agrawal, and Maria Goveas. Mashup feeds: continuous queries over web services. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1128–1130, New York, NY, USA, 2007. ACM.

[UJ99] M. Uschold and R. Jasper. A Framework for Understanding and Classifying ontology applications, 1999.

[VAC+05] Oliver Vogel, Ingo Arnold, Arif Chughtai, Edmund Ihler, Uwe Mehlig, Thomas Neumann, Markus Vi£¡lter, and Uwe Zdun. *Software-Architektur:*

Grundlagen - Konzepte - Praxis. Spektrum Akademischer Verlag, 1 edition, 9 2005.

[VKF] Thomas Vitvar, Jacek Kopecký, and Dieter Fensel. WSMO-Lite: Lightweight Semantic Descriptions for Service on the Web. PDF. Available online at `http://wsmo.org/TR/d11/v0.2/20080304/d11v02_20080304.pdf`; visited on September 2th 2008.

[vRDW08] M. Birna van Riemsdijk, Mehdi Dastani, and Michael Winikoff. Goals in agent systems: a unifying framework. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 713–720, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[WBG07] Thomas Weise, Steffen Bleul, and Kurt Geihs. Web service composition systems for the web service challenge - a detailed review. Technical Report 34-2007111919638, November 2007. Permanent Identifier: urn:nbn:de:hebis:34-2007111919638.

[WC08] Janice Warner and Soon Ae Chun. A citizen privacy protection model for e-government mashup services. In Soon Ae Chun, Marijn Janssen, and José Ramón Gil-García, editors, *DG.O*, volume 289 of *ACM International Conference Proceeding Series*, pages 188–196. Digital Government Research Center, 2008.

[Web] Web Portal Software from WebSphere. Website. Available online at `http://www.ibm.com/websphere/portal`; visited on August 28th 2008.

[Wei07] Karsten Weicker. *Evolutionäre Algorithmen (2. Auflage)*. Teubner, Stuttgart, 2007.

[WH07] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In Mary Beth Rosson and David J. Gilmore, editors, *CHI*, pages 1435–1444. ACM, 2007.

[WH08] Jeffrey Wong and Jason Hong. What do we "mashup" when we make mashups? In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering*, pages 35–39, New York, NY, USA, 2008. ACM.

[WSD] WSDL4J. Website. Available online at `http://sourceforge.net/projects/wsdl4j`; visited on August 28th 2008.

[XAL] Xalan-Java. Website. Available online at `http://xml.apache.org/xalan-j/`; visited on August 28th 2008.

[Yee08] Raymond Yee. *Pro Web 2.0 Mashups: Remixing Data and Web Services (Expert's Voice in Web Development)*. Apress, 2 2008.

[YSLH03] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 2003. ACM.

[ZL96] S. Zilberstein and V. Lesser. Intelligent Information Gathering Using Decision Models, 1996.

[ZRN08] Nan Zang, Mary Beth Rosson, and Vincent Nasser. Mashups: who? what? why? In Mary Czerwinski, Arnold M. Lund, and Desney S. Tan, editors, *CHI Extended Abstracts*, pages 3171–3176. ACM, 2008.

List of Figures

List of Tables

List of Algorithms