
# Case-based Reasoner

## for OWL-S Web Services

**Diploma Thesis,** February, 4, 2007

Simon Lützelschwab
of Kaiseraugst AG, Switzerland

Student ID: 99-718-934
simon.l@gmail.com

Supervisor: **Michael Daenzer**

Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.unizh.ch/ddis


Dynamic and Distributed
Information Systems

## Acknowledgments

I would like to thank my supervising assistant Michael Daenzer for the numerous discussions around the topic and for coping so well with my crazy ideas. He has done a fantastic job steering this thesis in the right direction and ultimately made it what it is. I would also like to thank Prof. Abraham Bernstein for giving me the opportunity to write this thesis in the first place.

## Abstract

This thesis explores a novel approach to use techniques found in Case-based Reasoner Systems and apply them to the Semantic Web. In the context of the Web Service Ontology OWL-S, a framework following the principles of Case-based Reasoning is introduced. A suitable case structure is defined that builds the basis of the system. Furthermore, various similarity strategies are implemented to determine the appropriate selection of suitable cases based on the novel problem presented to the system. Similarity is measured using semantic, syntactic and graph measurements. Additionally, different adaption strategies are introduced to facilitate the reuse process. The framework's architecture allows for custom extension of additional similarity and adaption strategies in the future.

## Zusammenfassung

Diese Diplomarbeit versucht die Technik traditioneller Case-based Reasoner Systeme auf das Gebiet des semantischen Webs anzuwenden. Ein auf den Prinzipien von Case-based Reasoning basiertes Framework wird vorgestellt welches auf der Web Service Ontologie OWL-S aufbaut. Eine angemessene Fallstruktur wird definiert, welche als Basis des Frameworks dient. Des Weiteren werden verschiedene Ähnlichkeitsmasse eingeführt welche bei der Auswahl der geeigneten Fälle dienlich sind. Ähnlichkeit ist gemessen anhand des neuen Problems, das dem System übergeben wird und verwendet semantische und syntaktische Ansätze als auch Methoden aus der Graphentheorie. Zusätzlich werden verschiedene Adaptionsstrategien eingeführt welche die Weiterverwendung der Fälle bewerkstelligen. Die Architektur des Frameworks erlaubt individuell gefertigte Erweiterungen weiterer Ähnlichkeits- und Adaptionsstrategien.

# Table of Contents

# 1 Introduction

The Internet has gained significant momentum over the past few years and has shaped many aspects of each individual's daily life. Thanks to search engines such as Google[1], information is more easily accessible than ever before, allowing people to connect with each other in ways that otherwise would not have been possible. This new form of collaboration not only results in interesting projects such as Wikipedia[2] but it has also had a significant impact on today's science as shown by the progress made related to prosopagnosia[3]. Though the Internet known today already serves as a helpful instrument, we are far from reaching its true potential. Most of the information and services available today are aimed at human users and consumed through a web browser. The increasing popularity of Web Services and initiatives such as the Semantic Web will allow any combination of communication between humans and software agents or fully automated interaction between agents only.

*"I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize."* - Tim Berners-Lee, 1999[4]

The Semantic Web not only opens up new dimensions for scientists to collaborate with each other, exchange knowledge, but it also enables to share resources across the globe. Research fields such as Bioinformatics or BioNMR require the use of complex tools and experiment set-ups where some tasks can run for several weeks. Today, scientists researching in these areas have to invest a significant amount of effort into getting familiar with the various tools and best practices, often by consuming the time of more experienced researchers or by trial and error approaches [Daenzer 2005].

To allow scientists to focus their energy on their primary research goal and contribute to their area, various software tools have been developed to assist in the creation of experiment set-ups, and to allow them to share resources as described in [Daenzer 05] and seen in MyGrid[5] and Taverna[6].
[Sankar and Simon 2004] describes Case-based Reasoning (CBR) as "a body of concepts and techniques that touch upon some of the most basic issues relating to knowledge representation, reasoning and learning from experience." CBR represents a very simple, but powerful technique to assist in solving novel problems based on past experience. In the context discussed above, CBR can add significant value in shortening the time needed to set-up new experiments and reduce the fault rate of new experiments when orientated on what has worked in the past.

1 Google – http://www.google.com
2 Wikipedia – http://www.wikipedia.com
3 Face Blind – http://www.wired.com/wired/archive/14.11/blind.html
4 Semantic Web – http://en.wikipedia.org/wiki/Semantic_Web
5 MyGrid - http://www.mygrid.org.uk
6 Taverna - http://taverna.sourceforge.net/

## 1.1 Goal of the Thesis

The goal of this thesis is to explore a novel approach to apply the concepts and techniques known by CBR to the Semantic Web and Web Services described by OWL-S in specific. Firstly, means to build a case base which will serve as the repository of past knowledge are introduced. A new problem will then be presented to the system which will return a set of best matching past cases based on various search criteria. Following different strategies, these cases are then combined with the new problem and returned. The system is developed to serve as a framework to facilitate the implementation of different means for retrieving and adapting cases. Various strategies for both steps have been implemented to serve as a basis and a proof of concept.

## 1.2 Structure

The structure of this thesis follows the design science thesis as outlined in [Bernstein 2005]: After highlighting the motivation, followed by a detailed description, an evaluation part with a concluding discussion and some final remarks are presented.

*Chapter 2* goes into the details about the initial motivation of choosing the concept of Case-based Reasoning applied to OWL-S. It also introduces the context to which the system is intended to be applied and presents related work.

*Chapter 3* gives a more detailed introduction into the design aspects of the system. Specifically, the main components – Case, Case Base, Retrieval, Reuse - of Case-based Reasoning are outlined in the specific context of OWL-S. Furthermore, the basics for the various similarity measurements and adaption strategies are outlined.

*Chapter 4* describes the concrete implementation of the design aspects outlined in the previous chapter. Various strategies for comparing and adapting cases are discussed in the context of their actual implementation.

*Chapter 5* discusses aspects of potential test cases that may be used to evaluate the system. Different scenarios are presented along with the intended purpose of the tests and what aspects of the system are tested.

*Chapter 6* summarizes the achievements of this thesis and its accomplishments compared to the initial goal. Additionally, an outlook of future work is presented.

## 1.3 Target Audience

This thesis has as its primary audience computer scientists who are interested in the field of Semantic Web, Web Services, Matchmakers, Planners and last but not least, CBR. Specifically, it is targeted towards the interested reader who wants to explore matchmaking beyond the traditional purely semantic approaches based on past experience. Additionally, individuals interested in the field of CBR might be able to explore new applications. Last but not least, people interested in reading a few entertaining pages that expand their horizon should feel encouraged to judge my literary qualities.

# 2 Motivation

This chapter gives an overview of the initial motivation, presents NExT and also provides an introduction to the principals of Case-based Reasoning, the Semantic Web and the Web Service ontology OWL-S.

## *2.1 NExT – The NMR EXperiment Toolbox*

Scientists working in the domain of Nuclear Magnetic Resonance (NMR) are facing complex experiment set-ups, expensive and scarce resources and the lack of appropriate Software support. Michael Daenzer took up the challenge to address many of these issues and go beyond it. In [Daenzer2005], a summary of his findings as well as a detailed proposal are presented. A semantically enriched and formal NMR process builds the basis of NExT. Taking advantage of the extensibility of OWL-S, a Software framework incorporates the planning process of new experiments, their documentation methods and their execution and control. An important aspect of the framework is to support its users in the design process of new experiment set-ups, which can be achieved by different planning systems and/or problem solving systems.
In summary, NExT is addressing the following main objectives: process plan definition, user guidance, process automation and knowledge exchange.

### 2.1.1 An exemplary Scenario

[Daenzer2005] describes an exemplary scenario to visualize the purpose and functionalities of NExT which is reproduced in a shortened version below:

A spectroscopist starts his protein structure determination project by creating a new NMR case and entering specific information known to him. He proceeds to open the workflow editor which gives him the choice of using an existing template or starting from scratch. In this particular scenario, it is assumed that an existing process plan is used that needs to be adapted.
After performing the required adaption, additional data needed for the execution of the experiment is entered and the execution process is initiated. During the execution, the user receives feedback over various different media such as the user interface, email or mobile text messages. In case of an error notification such as indication of a spectrometer malfunction, the execution can be interrupted and a request to the system to provide assistance on how to resolve the error can be issued. In this case, NExT suggests various actions to be taken (e.g. to fine tune some of the spectrometers working parameters).
If at any point in the experiment the spectroscopist realizes that he has limited experience with a certain procedure, he can run a reasoner engine to let him present similar cases and documents dealing with this topic. Additional researchers can also remotely connect to NExT servers in different laboratories and are able to view, change, invoke and monitor the current projects on their own screen without a need to be physically present.
As a project evolves, the reasoners provided by NExT can provide the user with an expanding number of hints, tips, clues and ideas. For example, when coming to a new experimental problem, the system may present a list of alternative plans on how to proceed from the current stage of the project based on past experience and various rules.

After the experiment has been run successfully, the spectroscopist can deposit his case on the worldwide case base to let other NMR spectroscopists benefit from his work.

### 2.1.2 The Concepts

NExT introduces formal models for NMR data, atomic and composite process representations to achieve the four main objectives: process plan definition, user guidance, process automation and knowledge exchange.
An experiment is reflected as a specific form of a composite process. A composite process consists of any number of atomic processes. These atomic processes in turn reflect various concrete tasks that can be controlled and executed by the system. Composite and atomic processes consume and produce data that is either entered by the user, passed on from preceding processes or outputted in some form.
NExT describes processes and data in a machine readable format in the sense of a formal data and process model. OWL-S builds the basis of this model as it provides the quality of formality and many similar concepts required such as representation of composite and atomic processes and definition of data flows.
In NExT, the NMR process can therefore be represented as a OWL-S service.

### 2.1.3 User Guidance

[Daenzer2005] determined that user guidance in the context of NExT can be provided in the following means:
*Inductive support*. Building on previous experiments, the user can gain insight into how specific issues have been resolved in the past. This approach builds on techniques known by Case-based Reasoning.

*Deductive support*. Properties, mostly logical, of single steps of possible executable functions, are used to provide suggestions on possible chains and configurations. Artificial Intelligence planning takes advantage of this specific approach.

*Templates*. Providing the user with appropriate templates helps to accelerate the design process of experiments. Most modern Software provide users assistance in the form of templates.

The focus of this thesis is to explore concrete implementations of offering the user inductive support in the context provided by NExT.

## *2.2 Case-based Reasoning*

### 2.2.1 History of Case-based Reasoning

As mentioned in [Sankar and Simon 2004], the field of Case-based Reasoning (CBR) has a relatively young history and has its origin in research being done in cognitive science. The earliest contributions in this area were from Roger Shank and his colleagues at Yale University. During the period 1977 to 1993, CBR research was highly regarded as a plausible high-level model for cognitive processing. It mainly focused on problems such as how people learn a new skill and how humans generate hypotheses about new situations based on their past experiences. The objectives of these cognitive-based researches were to construct decision support systems to help people learn. Many prototype CBR systems were built during this period. Examples include Cyrus [Kolodner 1983], Mediator [Simpson 1985], Persuader [Sycara 1988], Chef [Hammond 1989], Julia [Hinrihs, 1992] and Protos [Bareiss 1989]. Various CBR workshops were organized in 1988, 1989, and 1991 by the U.S. Defense Advanced Research Projects Agency (DARPA). These events are considered to have formally created the discipline of Case-based Reasoning. In 1993, the first European workshop on Case-based Reasoning (EWCBR-93) was held in Kaiserslauten, Germany. Since then, many international workshops and conferences on CBR have been held in different parts of the world. Other major artificial intelligence conferences, such as ECAI (European Conference on Artificial Intelligence), IJCAI (International Joint Conference on Artificial Intelligence), have also had CBR workshops as part of their regular programs.

### 2.2.2 Main Components and Features of Case-based Reasoning

A typical example to describe Case-based Reasoning can be taken from the medical domain: It is assumed that physicians are using a system to perform medical diagnosis on new patients based on past experience. The system holds a set of cases to make this past experience accessible. For this system, a case can represent a person's symptoms together with the associated treatments. When a doctor then examines a new patient, he compares the person's current symptoms with those of earlier patients who had similar symptoms. The treatment that these past patients have received is then used and modified as necessary to suit the new patient.
The above example which can be extended to many different domains shows that a Case-based Reasoner solves new problems by adapting solutions to older problems. Therefore, CBR involves reasoning from prior examples: retaining a memory of previous problems and their solutions and solving new problems by references to that knowledge. Generally, a Case-based Reasoner will be presented with a problem, either by a user or by a program or system. The Case-based Reasoner then searches its memory of past cases in its case base and attempts to find a case that has the same problem specification as the case under analysis. If the reasoner cannot find an identical case in its case base, it will attempt to find a case or multiple cases that most closely match the current case.
In situations where a previous identical case is retrieved, assuming that its solution was successful, it can be offered as a solution to the current problem. In the more likely situation that the case retrieved is not identical to the current case, an adaption phase occurs. During adaption, differences between the current and retrieved cases are first

identified and then the solution associated with the case retrieved is modified, taking these differences into account. The solution returned in response to the current problem specification may then be tried in the appropriate domain setting.

The problem solving life cycle in a CBR system consist essentially of the following four parts:

1. Retrieving
   Similar previously experienced cases whose problems are considered to be similar are selected.
2. Reusing
   The cases are reused by either copying or integrating the solution from the retrieved cases.
3. Revising
   The retrieved solutions are revised or adapted to try to solve the new problem.
4. Retaining
   The new solution is stored after being confirmed and validated.

The following figure shows a graphical representation of the typical CBR cycle:

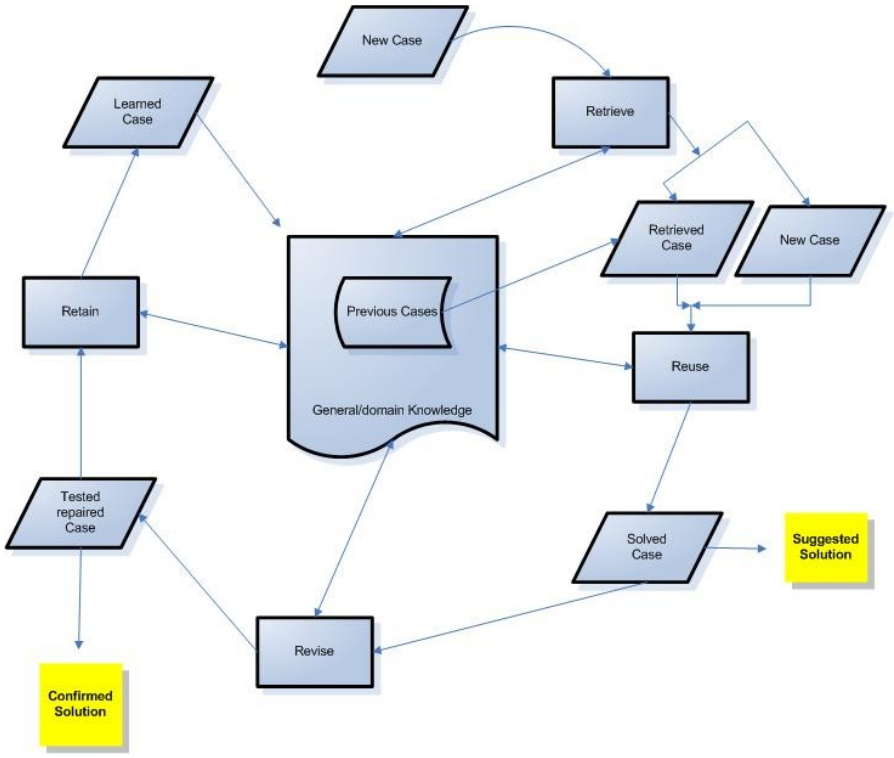


*Figure 1: CBR Cycle*

In many practical applications, the reuse and revise stages are often difficult to distinguish and therefore, are often also referred to as a single stage called adaption. The adaption process is usually a complicated process and strongly dependent on the general domain knowledge. This process can be supported by implementing rules that are applicable to the domain and therefore vary in their complexity. As an alternative approach, it could also be considered to take advantage of such systems as AI planners to complement the missing parts.

## *2.3 Semantic Web and OWL-S*

“The Semantic Web is a web of data. There is lots of data we all use every day, and its not part of the web. I can see my bank statements on the web, and my photographs, and I can see my appointments in a calendar. But can I see my photos in a calendar to see what I was doing when I took them? Can I see bank statement lines in a calendar?

Why not? Because we don't have a web of data. Because data is controlled by applications, and each application keeps it to itself.

The Semantic Web is about two things. It is about common formats for interchange of data, where on the original Web we only had interchange of documents. Also it is about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing.”[7]

### 2.3.1 The Semantic Web

To achieve the goals of the Semantic Web, the following technologies are used which are also included in the Semantic Web stack shown in figure 2[8][9].

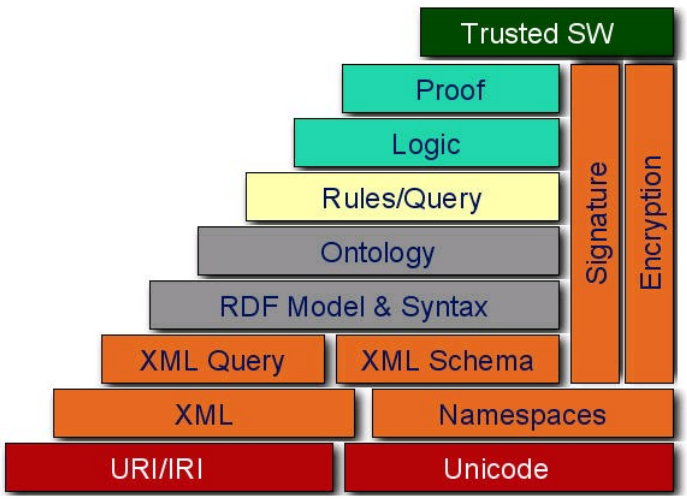


*Figure 2: W3C Semantic Web stack*

- XML
  Provides a surface syntax for structured documents but imposes no semantic constraints on the meaning of these documents.
- RDF
  Is a simple data model for referring to objects ("resources") and how they are related.
- RDF-Schema
  Is a vocabulary for describing properties and classes of RDF resources

7 Introduction to the Semantic Web on the W3C website - http://www.w3.org/2001/sw/
8 Figure from W3 website - http://www.w3.org/2004/Talks/1109-sb-gsaWebSci/slide14-0.html
9 Components description partially from Wikipedia - http://en.wikipedia.org/wiki/Semantic_Web

with a semantics for generalization-hierarchies of such properties and classes.

- OWL
  Adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

The Semantic Web has numerous advantages and is able to contribute with a number of benefits in various scenarios. The main advantage is that it provides a common framework to share data on the Web across application boundaries. This will not only allow R&D scientists to exchange knowledge in a much more efficient way but also to automate many tasks that are currently being done manually. The following figure visualizes the data integration in Life Sciences[10]:

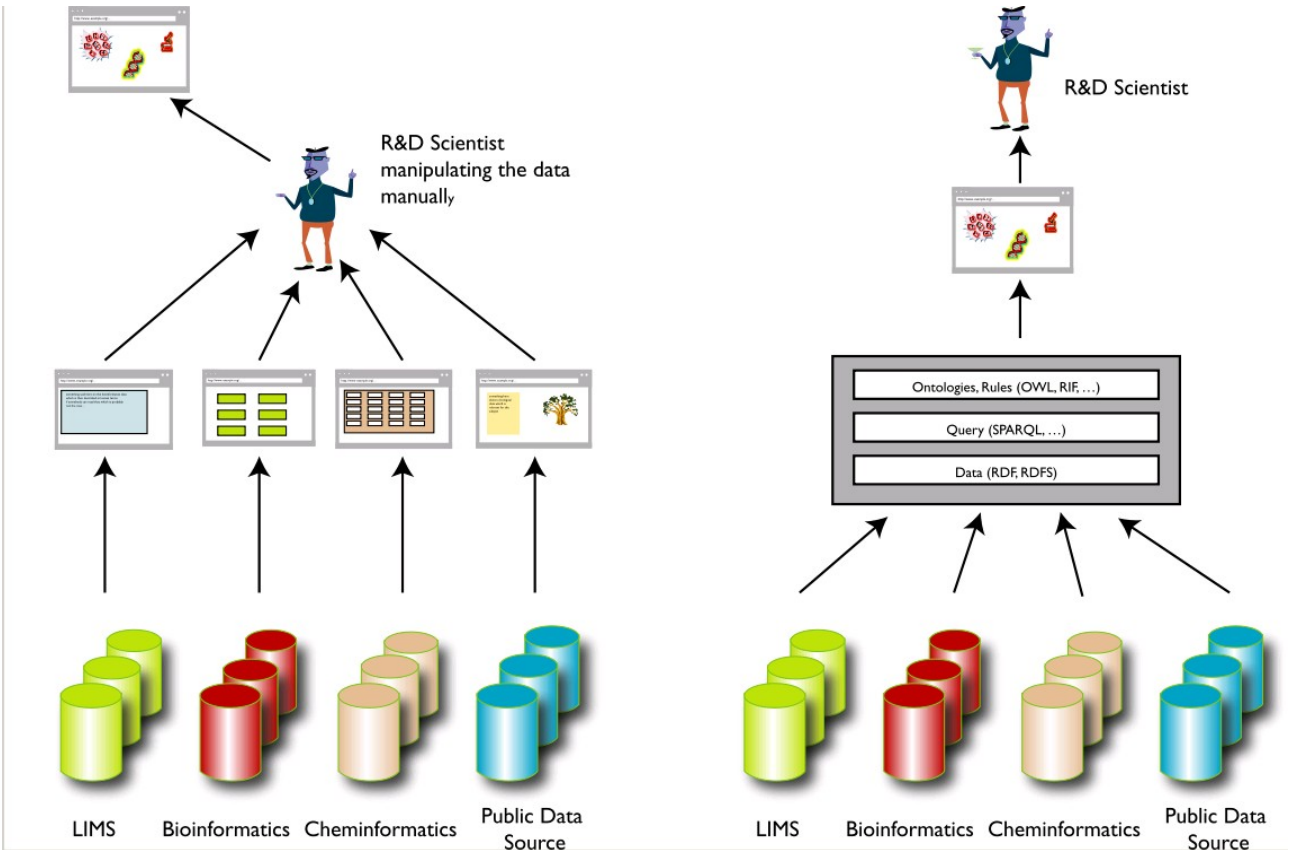


*Figure 3: Data Integration in Life Sciences*

## 2.3.2 OWL-S

"The Semantic Web should enable greater access not only to content but also to services on the Web. Users and software agents should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties, and should be able to do so with a high degree of automation if desired. Powerful tools should be enabled by service descriptions, across the Web service lifecycle. OWL-S (formerly

10 Figure taken from http://www.w3.org/2006/Talks/0927-Berlin-IH/Slides.html#(10)

DAML-S) is an ontology of services that makes these functionalities possible." [Martin et al. 2004]
Based on the progress made in the Semantic Web effort, the DARPA Agent Markup Language (DAML) Program developed a set of ontologies in the ontology language OWL to describe Web services. OWL-S is therefore "a OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form."[11] The goal is to facilitate the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation.
OWL-S therefore defines a top-level ontology Service to provide the data needed to discover and invoke, but it also allows composition and inter-operation of Web Services.
A service either consists of a single atomic process or a composite process. The process model itself is formally described in OWL. The following figure shows selected classes and properties of this model as described in [Martin et al. 2004]:

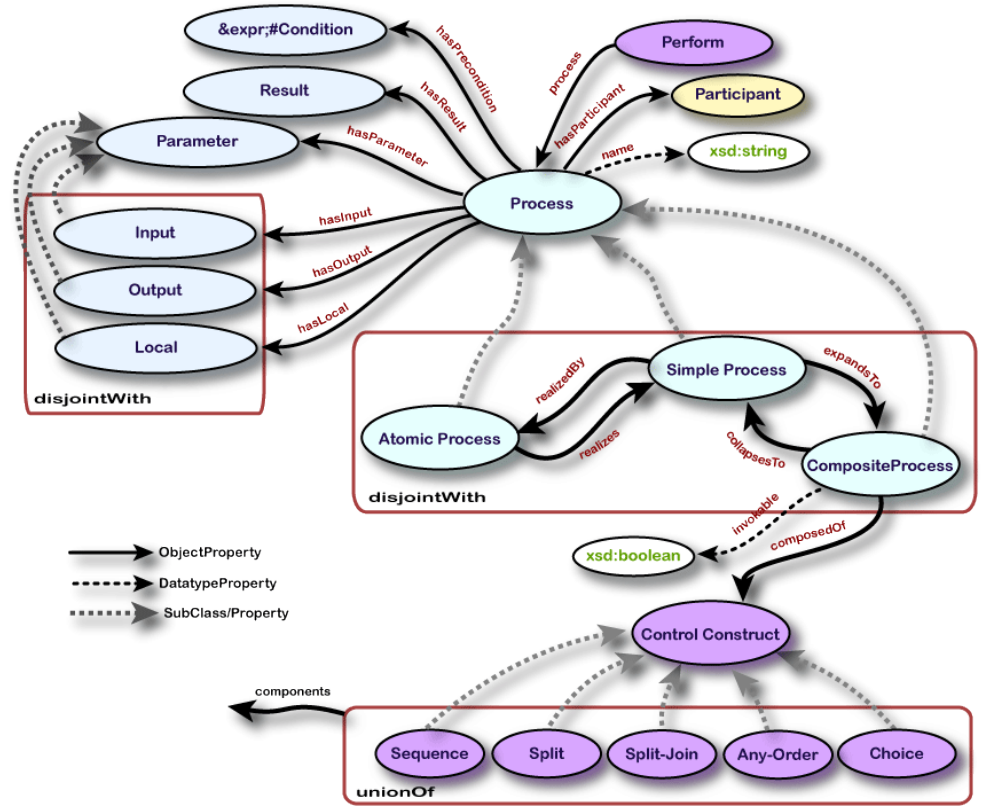


*Figure 4: Top level of the process ontology*

Each process can have parameters, preconditions and results. Parameters are either inputs or outputs in any number and associated with a certain type such as string, integer, book, ZIP or any other representation described in OWL. Preconditions are conditions that need to be satisfied in order to execute the process. There is currently no defined standard on how these conditions should be described formally. Examples of possible languages used include Swrl, KIF, and PDDL. Results describe the outcome of the successful execution of a process, in particular, the output in terms of information that is returned by the process and the effects which are changed conditions in the world.

11 http://www.daml.org/services/owl-s/

Each composite process has an appropriate control construct associated with it to describe the execution path of the processes that belong to it. These control constructs can be sequence descriptions of processes or more complex constructs such as If-Then, Repeat-While etc. to control the execution flow.

## *2.4 Related Work*

In this section, related work is presented and briefly summarized. Furthermore, the description of each project is concluded by a comparison to the initial motivation.

### 2.4.1 Case-based Reasoner: jColibri


jColibri[12] is a generic framework to build new Case-based Reasoning systems. It has been developed by the Group for Artifical Intelligence Applications (GAIA[13]) at the University of Madrid (Universidad Complutense Madrid). The framework is a proposal of a domain independent architecture named COLIBRI (Cases and Ontology Libraries Integration for Building Reasoning Infrastructures). Its purpose is to assist in the design process of knowledge intensive CBR (KI-CBR) systems. COLIBRI is based on knowledge acquisition from a library of application-independent ontologies and the use of CBROnto which is intended to be an ontology holding the common CBR terminology that guides case representation and allows the description of flexible, generic and reusable CBR Problem Solving Methods (PSMs) to solve the typical CBR tasks.
jCOLIBRI is a technological evolution of COLIBRI that incorporates into a distributed architecture a DLs engine, GUI clients for assembling a CBR system from reusable components and an object-oriented framework in Java. The design of the framework comprises a hierarchy of Java classes plus a number of XML files organized around the following elements:

- *Tasks and Methods*. XML files describe the tasks supported by the framework along with the methods for solving those tasks.
- *Case Base*. Different connectors are defined to support several types of case persistency, from the file system to a data base. Additionally, a number of possible in-memory Case Base organizations are supported.
- *Cases*. A number of interfaces and classes are included in the framework to provide an abstract representation of cases that support any type of actual case structure.
- *Problem solving methods*. The actual code that supports the methods included in the framework. jCOLIBRI is designed to easily support the construction of (different types of) CBR systems taking advantage of the task/method division paradigm described. Building a CBR system is a configuration process where the system developer selects the tasks the system must fulfill and for every task it assigns the method that will perform the action. Ideally, the system designer would find every task and method needed for the system at hand so that there would only be a need to program the representation for cases. However, in a more realistic situation a number of new methods and in some cases, new tasks, may be needed. Since jColibri is designed as an extensible framework, new elements should smoothly

12 http://gaia.fdi.ucm.es/grupo/projects/jcolibri/index.html
13 http://gaia.fdi.ucm.es/index.html

integrate with the available infrastructure when they follow the framework design.

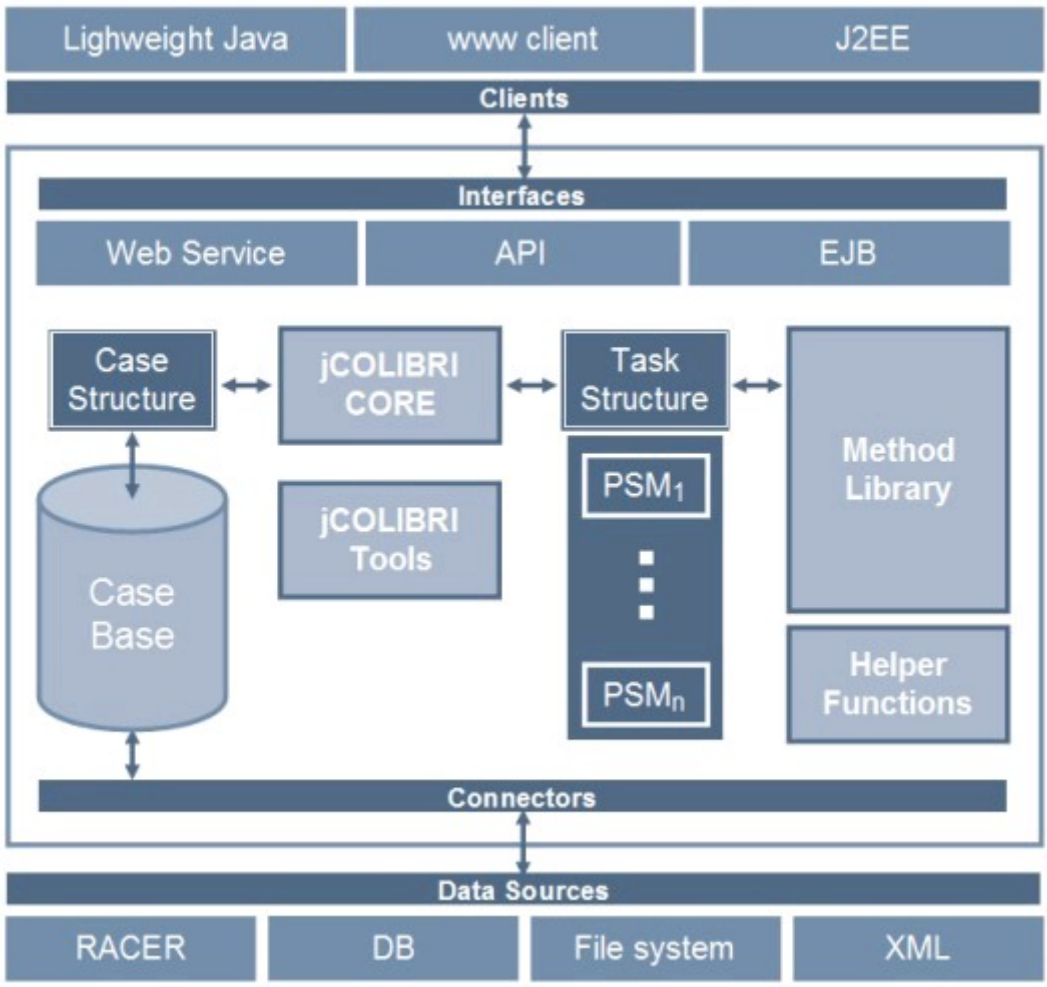


*Figure 5: jColibri Architecture*

In order to alleviate a framework instantiation effort, jColibri features a semiautomatic configuration tool that guides the instantiation process through a graphical interface. This interface is dynamically built to reflect the actual contents of the task/method ontology, relying on the XML files describing task and method constraints and profiting from reflection facilities implemented in Java. The configuration of a CBR system using this interface consists of the following processes:

- Defining the case structure, the source for cases and the case base organization.
- While the system is not complete, selection of one of the tasks without a method assigned, selection and configuration of a method for that task. At start-up the task tree has only one element, CBRTask, which is solved by a decomposition method that results in additional tasks. Task/method constraints are being tracked during the configuration process so that only applicable methods in the given context are offered to the system designer.
- Once the system is configured, the configuration code is generated so that a running CBR system is available. The configuration tool also provides a default interface for running the configured CBR system.

As jColibri presents an extensible framework and also supports using ontologies as attributes of cases, it lacks the flexibility to present cases of flexible structure. However, it will be interesting to see further progress of the framework and the author hopes it will find its justified place amongst the CBR community.

### 2.4.2 Matchmaker: OWLS-MX


OWLS-MX[14] is a hybrid OWL-S Web Service Matchmaker developed at the German Research Center for Artificial Intelligence[15] (DFKI) of Saarbruecken, Germany.
OWLS-MX is a hybrid semantic Web service matchmaker that retrieves services for a given query both written in OWL-S, and based on imported ontologies in the described in OWL. For this purpose, the OWLS-MX matchmaker performs pure profile based service IO-matching but combines logic-based semantic matching with syntactic token-based similarity metrics to achieve better performance. It uses the OWL-DL description logic reasoner Pellet for logic based filtering, and the cosine, loss-of-information, extended Jacquard, and Jensen-Shannon information divergence based similarity metrics for complementary approximate matching.
OWLS-MX computes the degree of semantic matching for a given pair of service advertisement and request by successively applying five different filters: exact, plug in, subsumes, subsumed-by and nearest-neighbor. The first three are logic based only whereas the last two are hybrid due to the required additional computation of syntactic similarity values.
The general matching algorithm used by OWL-S MX is described as follows in [Klusch et al. 2006]: The OWLS-MX matchmaker takes any OWL-S service as a query, and returns an ordered set of relevant services that match the query each of which annotated with its individual degree of matching, and syntactic similarity value. The user can specify the desired degree, and syntactic similarity threshold. OWLS-MX then first classifies the service query I/O concepts into its local service I/O concept ontology. For this purpose, it is assumed that the type of computed terminological subsumption relation determines the degree of semantic relation between pairs of input and concepts. Auxiliary information on whether an individual concept is used as an input or output concept by any registered service is attached to this concept in the ontology. The respective lists of service identifiers are used by the matchmaker to compute the set of relevant services that I/O match the given query according to its five filters. In particular, OWLS-MX does not only pairwisely determine the degree of logical match but syntactic similarity between the conjunctive I/O concept expressions in OWL Lite. These expressions are built by recursively unfolding each query and service input (output) concept in the local matchmaker ontology. As a result, the unfolded concept expressions are including primitive components of a basic shared vocabulary only. Any failure of logical concept subsumption produced by the integrated description logic reasoner of OWLS-MX will be tolerated, if and only if the degree of syntactic similarity between the respective unfolded service and request concept expressions exceeds a given similarity threshold.

14 http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/index.html
15 http://www.dfki.de/

OWL-S MX extends the approach of matchmaking for Web Services by building a global ontology and also by including syntactic algorithms in the matching process. It is, however, not including the service's structure or single resources it is composed of as the focus is on being a matchmaker application.

### 2.4.3 AI Planner: CASPER

CASPER[16] (Continuous Activity Scheduling Planning Execution and Replanning) is a product of the Jet Propulsion Laboratory[17] which is a part of NASA. It uses iterative repair to support continuous modification and updating of a current working plan in light of the changing operating context. It uses AI technology for Planning and Scheduling, Planning and Execution and Replanning. CASPER is being used in a range of projects including autonomous spacecraft, autonomous rovers, ground communications station automation, and uninhabited aerial vehicles.
Traditional batch oriented models of planning have shortcomings for spacecraft control. Constructing a plan from scratch can be a computationally intensive process and onboard computational resources are typically quite limited, so that it still may require considerable time to generate a new operations plan. As a data point, the planner for the Remote Agent Experiment (RAX) flying on-board the New Millennium Deep Space One mission takes approximately four hours to produce a three day operations plan. RAX is running on a 25 MHz RAD 6000 flight processor and uses roughly 25% of the CPU processing power. While this is a significant improvement over waiting for ground intervention, making the planning process even more responsive (e.g., on a time scale of seconds) to changes in the operations context would increase the overall time for which the spacecraft has a consistent plan. As long as a consistent plan exists, the spacecraft can keep busy working on the requested goals.
AI planners are powerful and flexible systems. However, they are very specific to the domain they have been developed for. A typical planner takes three inputs: a description of the initial state of the world, a description of the desired goal, and a set of possible actions, all encoded in a formal language. The planner produces a sequence of actions that lead from the initial state to a state meeting the goal.

It is often the case that the main goal needs to be separated in various intermediate steps. As AI planners are certainly able to solve complex issues, the complexity also grows exponentially the more steps are involved to achieve the goal. Initially, it is often the case that these intermediate steps are unknown to the user facing the problem. The approach of Case-based Reasoning also assists in this respect as past knowledge is reused.

### 2.4.4 Conclusion

The approaches presented in this section solve partial aspects of the problem area, but none of these cover it in its entirety. Specifically, the following qualities need to be satisfied: flexible case structure, structure of OWL-S services and uncertainty of how to achieve the specific main goal.

16 http://www-aig.jpl.nasa.gov/public/planning/casper/
17 http://www-aig.jpl.nasa.gov/

## 2.5 Personal Motivation

I possess a strong personal interest in the field of knowledge management, optimization and automation of workflows and the Internet in general. With these interests in mind, I was looking for a suitable use case and consulted the home of Dynamic and Distributed Information Systems (DDIS)[18] of the Institute of Informatics of the University of Zurich. Michael Daenzer has been working on his project NExT [Daenzer 2005] and is looking for an extension to provide such functionality in the context of his Toolbox application.
The Semantic Web represents a major milestone in the history of the Internet and has the potential to significantly change our interaction with the Net and expand its usefulness beyond what we know today. Unfortunately, one of the issues encountered today is that the Semantic Web is in its early stages and there is lots of controversy about what technology to use. Furthermore, there is still a lack of available tools for the automated creation of semantically annotated data. Therefore, the flexibility offered by Case-based Reasoning therefore seems to be a natural solution.

18 Home of DDIS - http://www.ifi.unizh.ch/ddis/

# 3 Design

As outlined in the motivational chapter, the aim of the system is to provide inductive support to the user. As the formal model chosen by NExT is based on OWL-S, it needs to be able to interact with this specific technology.
In conclusion, this chapter describes the design aspects of implementing a Case-based Reasoner system for OWL-S Web Services in the context of NExT.

## *3.1 Overview*

A Case-based Reasoner should essentially provide the four main tasks as outlined in the previous chapter: retrieval, reuse, revision and retention. In order to have access to past knowledge, the concept of a case needs to be defined as well as the collection and organization of a set of these cases. Furthermore, the means to collect a specific case need to be provided in order to be able to populate the cases with concrete content.
To retrieve and reuse cases, various strategies need to be considered. These steps are taking advantage of the information recorded in a case to be able to determine the proximity of a past case to a new one.
The relevant context of the NExT environment is built around the NMR Process as defined in [Daenzer2005]. The NMR Process can formally be described as OWL-S service.
In order to retain past knowledge, the system needs to implement methods to be made aware of such knowledge. NExT implements an execution engine to be able to execute NMR processes. Any relevant data associated with the execution of a specific service needs to be recorded and stored in a suitable fashion. These recorded executions build the basis for the case base which represents the memory of the system.
The case base itself needs to expose individual cases that represent the recorded information and present it in a fashion suitable to be compared against the new problem. It also needs to make relevant information available to be able to adapt the old case to the new case.

## *3.2 Recording Cases / Execution Trails*

NMR Processes can be described as OWL-S Web Services. In order to execute a service, data needs to be submitted to its inputs if any are required. The process associated with the service is then executed with the corresponding data submitted to its inputs. In case the process is an atomic process, the relevant data is returned in its outputs and the execution is finished. On the other hand, if the process is a composite process, the control construct associated with it determines the proceeding processes to be executed which can either be atomic or composite in nature. A control construct may also be associated with a condition that determines the execution flow.

The following control constructs are defined in OWL-S:

| Control Construct |
|---|
| Sequence |
| Split |
| Split+Join |
| Any-Order |
| Choice |
| If-Then-Else |
| Iterate |
| Repeat-While |
| Repeat-Until |

*Table 1: OWL-S Control Constructs*

Corresponding to [Martin et al. 2004], a description of each control construct is given:

***Sequence***. A list of control constructs to be done in order.

***Split***. The components of a Split process are a bag of process components to be executed concurrently. Split completes as soon as all of its component processes have been scheduled for execution.

***Split+Join***. Here the process consists of concurrent execution of a bunch of process components with barrier synchronization. That is, Split+Join completes when all of its components processes have completed. With Split and Split+Join, we can define processes that have partial synchronization (e.g., split all and join some sub-bag).

***Any-Order***. Allows the process components (specified as a bag) to be executed in some unspecified order but not concurrently. Execution and completion of all components is required. The execution of processes in an Any-Order construct cannot overlap, i.e. atomic processes cannot be executed concurrently and composite processes cannot be interleaved. All components must be executed. As with Split+Join, completion of all components is required.

***Choice***. Choice calls for the execution of a single control construct from a given bag of control constructs (given by the components property). Any of the given control constructs may be chosen for execution.

***If-Then-Else***. The If-Then-Else class is a control construct that has properties ifCondition, then and else holding different aspects of the If-Then-Else. Its semantics is intended as “Test If-condition; if True do Then, if False do Else."

***Iterate***. The Iterate construct makes no assumption about how many iterations are made or when to initiate, terminate, or resume. The initiation, termination or maintenance condition could be specified with a whileCondition or an untilCondition.

Iterate is an "abstract" class, in the sense that it's not detailed enough to be instantiated in a process model. It's defined to serve as the common superclass of Repeat-While, Repeat-Until, and potentially other specific iteration constructs that might be needed in the future.

***Repeat-While*** *and* ***Repeat-Until***. Both of these iterate until a condition becomes false or true, following the familiar programming language conventions. Repeat-While tests for the condition, exits if it is false and does the operation if the condition is true, then loops. Repeat-Until does the operation, tests for the condition, exits if it is true, and otherwise loops. Therefore, Repeat-While may never act, whereas Repeat-Until always acts at least once.

The execution is finished when all relevant processes have been executed and the appropriate output is returned.

```
populate inputs with concrete data
get process associated with service
execute:
if process atomic
      execute
else if process composite
      get control construct associated with process
      evaluate eventual conditions
      get relevant processes
      for each process {
            populate appropriate inputs
            go to execute
      }

return output
```

*Text 1: Pseudo code Execution*

The exact execution trail should be recorded including all processes used and the concrete values of their corresponding inputs and outputs. The execution trail itself always consists only of control constructs of the type sequence. When a service is executed, it's concrete values define a specific execution flow for the particular value-data pair. The following figure shows an example of a composite process using a If-Then control construct and the resulting execution trail:

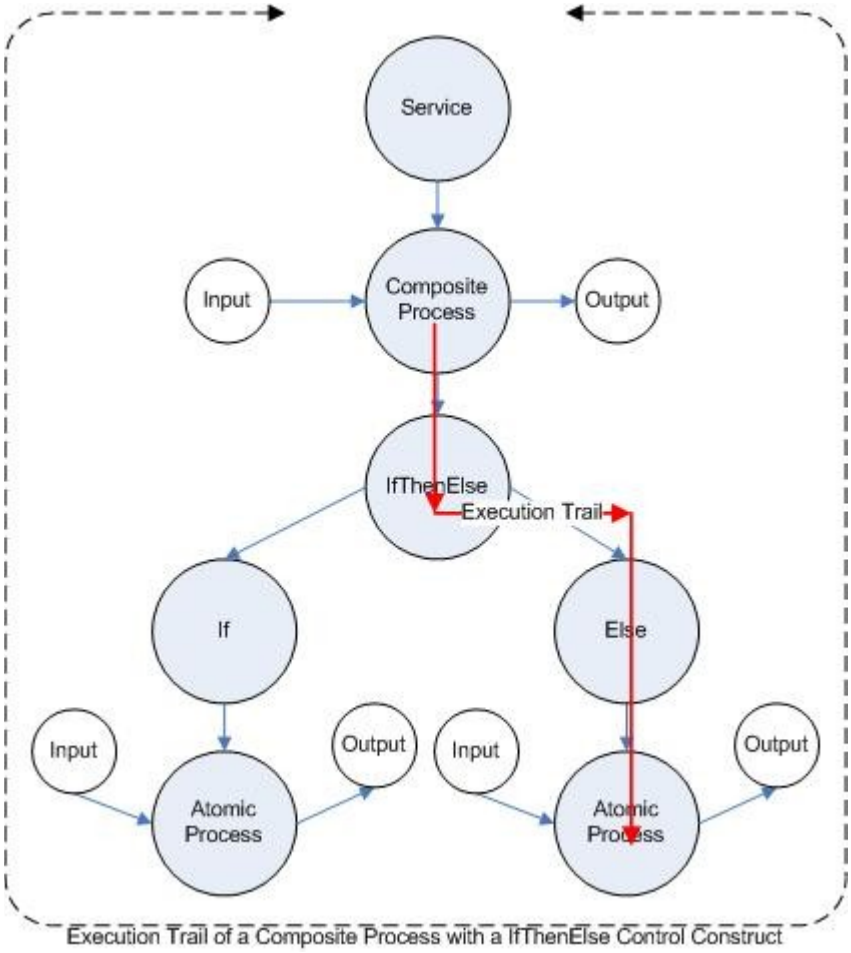


*Figure 6: Execution Trail of a Composite Process*

The resulting execution trail of the example shown in the previous figure is a composite process which consists of a sequence holding the atomic process of the Else control construct. The same logic applies to control constructs of the type Repeat-While, Repeat-Until, Split, Split+Join and Choice.

Services and processes can also hold preconditions which state under which conditions they can be executed. A process will not execute properly unless its preconditions are true. As previously noted, preconditions are represented as logical formulas in a formal language such as Swrl, KIF, or PDDL. As there is currently no standard to describe preconditions, they are currently neglected in the execution trail.

## *3.3 Case*

A case represents an actual happening of the past. It needs to make this information available in a form suitable to be reused and eventually adapted to the new problem. A case is defined in a certain structure and holds the essential information to the specific problem that has been solved with this case. In many practical Case-based Reasoning applications, cases are usually represented as two unstructured sets of attribute-value pairs that represent the problem and solution features. However, the intended purpose of a Case-based Reasoning system greatly influences what and how it is stored.

The intended purpose of this particular Case-based Reasoning system is to store executed OWL-S Web Services. These execution trails serve as the solution of a particular problem. The problem itself is implicitly encoded in the execution trail. The various inputs, outputs and processes used, but also the initial graph of the original Web Service, are ultimately defining the problem space.
Considering a Web Service takes two inputs of the type integer and returns one output of the type integer, it can be assumed that this is a potential solution to a new problem which is also using inputs of the type integer and returning an output of the type integer. Especially in the domain of NMR, the graph defined by the original Web Service can also hold useful information of the particular problem solved by this service. In NExT, a NMR process can essentially reflect a complete experiment set-up which in turn is represented by a service as defined in OWL-S. The structure of this experiment set-up can be used to determine how closely the new problem is related to a previously executed experiment. To perform a certain experiment, specific instruments need to be used which take a certain amount of inputs and return a specific amount of outputs. In case of a composite processes, the various control constructs used hold further information about the structure of the experiment set-up. The following figure shows a simple example of a potential experiment set-up that takes one input, passes this information to two subprocesses which are executed consequentially and returns one output:

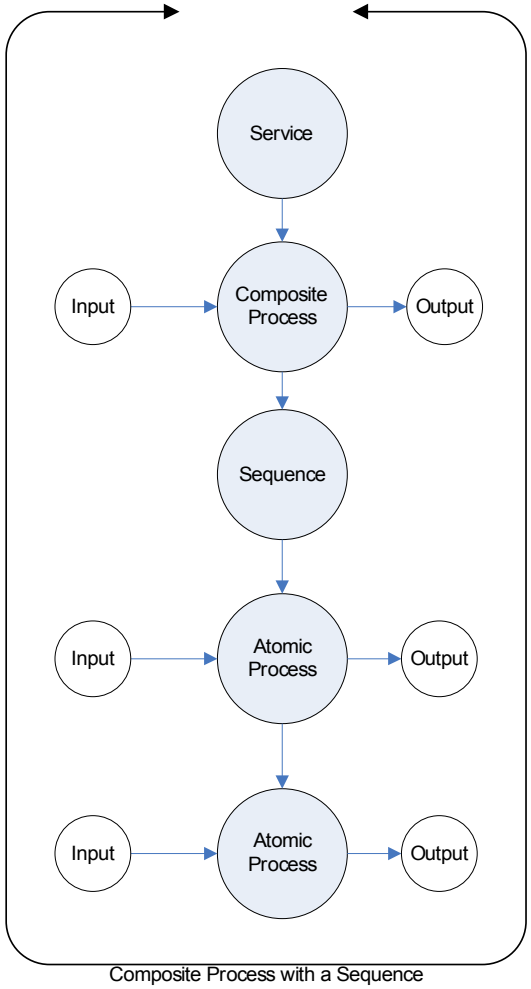


*Figure 7: Structure of an Experiment Set-Up*

OWL-S Web Services can contain any number of either atomic or composite processes which again consist of atomic processes. Furthermore, composite processes can hold various control constructs that define a specific path during execution and associate parameter bindings which define the inter-process data flows. In [Martin et al. 2004], these bindings are defined in a consumer-pull convention. This means that the data consuming process has to indicate which data item it requires. A case needs to represent the exact execution of a Web Service as happened at execution time. An additional important property of a case is that it needs to have the flexibility of representing an arbitrary number of processes of either nature and its potential bindings.

In summary, a case should hold the following information:

| Element | Information |
|---|---|
| Input | Datatype |
| Output | Datatype |
| Process | Specific process (URI) |
| Input Bindings | Inter-process data flow |
| Graph of executed service | Structural representation |

*Table 2: Case Requirements*

## *3.4 Case Base and Indexing*

The case base consists of a set of cases that represent past knowledge. Regardless of the format chosen to represent cases, the collection of cases itself also has to be structured in a way that facilitates retrieval of the appropriate case when queried. Case indexing refers to assigning indexes to cases for future retrieval and comparison. Indexes should be predictive in a useful manner. This means that indexes should reflect the important features of a case and the attributes that influence the outcome of the case, and describe the circumstances in which a case is expected to be retrieved in the future.
Past knowledge in the context of this particular system means executed Web Services described in OWL-S. The system should provide means to hold a case base and make these cases accessible in a simple manner. The structure of the case base should be simple and consistent within the OWL-S technology. Indexes should only be implemented if they are required in this context or are necessary requirements to improve performance.

## *3.5 Retrieve*

Case retrieval is the process of finding, within a case base, those cases that are the closest to the current case. To carry out effective case retrieval, there must be selection criteria that determine how a case is judged to be appropriate for retrieval and a mechanism to control how the case base is searched. The selection criteria are necessary to determine which is the best case to retrieve, by determining how close the current case is to the cases stored.
The case selection criteria depend partly on what the case retriever is searching for in the case base. Most often the case retriever is searching for an entire case, the features of

which are compared to those of the current case. However, there are times when only a portion of a case is being sought. This situation may arise because no full case exists and a solution is being synthesized by selecting portions of a number of cases. A similar situation is when a retrieved case is being modified by adopting a portion of another case in the case base.
The selection criteria greatly influence what cases are selected and then used for adaption. Existing cases with a higher degree of similarity compared to the current case have a greater chance of being suitable for reuse. On the other hand, only considering cases that are exact matches reduces the probability of returning any case. One of the most important assumptions of Case-based Reasoning is “that similar experience can guide future reasoning, problem solving and learning” [Sankar and Simon 2004].
The structure of the case defines the features available to be used to measure similarity between the existing and new case. In particular, these attributes are the processes with their various inputs and outputs as well as the structure of the original service.
In OWL-S, parameters – inputs and outputs – are defined by a unique URI. Furthermore, every parameter has a type, specified using a URI. This is not the OWL class the parameter belongs to, but a specification of the class (or datatype) that values of the parameter belong to. The following table shows examples of possible parameter types:

| **Datatype** | **URI** |
|---|---|
| Non Negative Integer | http://www.w3.org/2001/XMLSchema#nonNegativeInteger |
| Float | http://www.w3.org/2001/XMLSchema#float |
| Price | http://www.mindswap.org/2004/owl-s/concepts.owl#Price |
| Book | http://purl.org/net/nknouf/ns/bibtex#Book |
| ZIP Code | http://www.daml.org/2001/10/html/zipcode-ont#ZipCode |

*Table 3: Example of Datatypes for Parameters*

Processes are either atomic or composite and are as well referenced using a unique URI. A process itself does not hold much other useful information.
As previously described, the graph of the service holds information about the structure of the service and eventual control constructs such as If-Then or loops.
Given the nature of the available information, appropriate similarity measures need to be implemented to select similar cases that can be considered for reuse.
Potential approaches can be found in three main areas which are described in more detail in the following subsection:

| **Type of Similarity** | **Comparison** |
|---|---|
| Semantic | Semantic Datatype |
| Syntactic | String-based URI |
| Graph | Structure of Web Service |

*Table 4: Similarity Strategies*

### 3.5.1 Similarity Strategies

The system should implement different strategies to determine similarity between existing cases and the new case. Furthermore, it should allow additional strategies to be added in the future. As a basis, the various processes used by the Web Service and its Inputs respectively Outputs should be considered.

#### *3.5.1.1 Semantic similarity*

Semantic similarities are important in the application of Web Service matchmaking. “DAML-S and its Service Profile take up the challenge of representing the functionalities of web services. This paper [Massimo et al. 2002] contributes to this challenge by describing a matching engine that allows matching of advertisements and requests on the bases of the capabilities that they describe. This is a major improvement on current technology that allows only location of services based on keyword matching. Indeed we show how the matching engine can be used to improve the functionalities of existing web service repositories such as UDDI.” [Massimo et al. 2002].

Relevant to the Case-based Reasoner application is the distinction of different matching degree. [Massimo et al. 2002] define the following matching criteria whereas outR defines the output of the requester and outA the output of the advertised service:

***Exact***. If outR=outA then outR and outA are equivalent, which is labeled as exact. Secondly, if outR subclassOf outA then the result is still exact under the assumption that by advertising outA the provider commits to provide outputs consistent with every immediate subtype of outA. This is like to say that, given the ontology fragment in the following figure, the provider, by advertising NMR resource, commits to provide probehead, spectrometer and sample. If instead it provides only spectrometer, then a better strategy would be to restrict its advertisement to the latter.

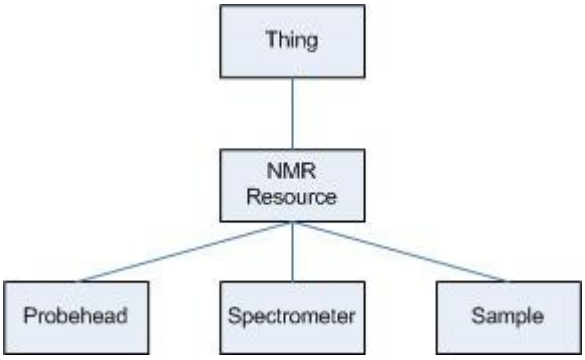


*Figure 8: Fragment of the NMR Resource Ontology*

***Plug in***. If outA subsumes outR then outA is a set that includes outR, or, in other words, outA could be plugged in place of outR[19]. For example, a service that provides - any type of - NMR resource could be of use for another service that expects spectrometer. This rule acknowledges that there is a weaker relation between outR and outA in this case, than in the exact case above: we can expect that a service that advertises an output of NMR resource provides some type of resource, but we cannot expect that it provides every type of spectrometer.

***Subsumes***. If outR subsumes outA, then the provider does not completely fulfill the request. The requester may use the provider to achieve its goals, but it likely needs to modify its plan or perform other requests to complete its task.

***Failure***. Failure occurs when no subsumption relation between advertisement and request is identified.

19 subclassOf in DAML also defines a subsumption relation, therefore the exact match defined above is also based on the subsumption relation. The rules for plug in matching apply when the concepts are not the same and no subclassOf relation holds.

Following the pseudo code of the matching algorithm as defined in [Massimo et al. 2002]:

Main control loop:

```
match(request) {
recordMatch= empty list
for all adv in advertisements do {
      if match(request, adv) then
      recordMatch.append(request, adv) }
      return sort(recordMatch)
}
```

Algorithm for output matching:

```
outputMatch(outputsRequest, outputsAdvertisement) {
globalDegreeMatch= Exact
for all outR in outputsRequest do {
      find outA in outputsAdvertisement such that
      degreeMatch= maxDegreeMatch(outR,outA)
      if (degreeMatch=fail) return fail
      if (degreeMatch<globalDegreeMatch)
      globalDegreeMatch= degreeMatch
      return sort(recordMatch)
}
```

Rules for the degree of match assignment:

```
degreeOfMatch(outR,outA):
if outA=outR then return exact
if outR subclassOf outA then return exact
if outA subsumes outR then return plugIn
if outR subsumes outA then return subsumes
otherwise fail
```

*Text 5: Pseudo Code of Matching Algorithm for Web Services*

The approach presented above can be applied in a similar fashion to determine semantic similarity in the application of the Case-based Reasoner system: the new case can be regarded as the service requester and the old case as the service advertiser. Additionally, instead of limiting the matching to the service's inputs and outputs only, the inputs and outputs of each single process can be matched using the algorithm presented above.

#### *3.5.1.2 Syntactic Similarity*

Inspired by OWL-MX[20] and the limitations imposed by the current state of the Semantic Web on relying purely on semantic matching, traditional methods of content-based information retrieval (IR) can be considered. Each resource in a OWL-S ontology is described by a unique URI. Often, these URIs are chosen by humans and hold a certain degree of useful information. The following table shows examples of URIs used in OWL-S ontologies and the resource they describe:

| URI | Description |
|---|---|
| http://www.mindswap.org/2004/owl-s/1.1/AmazonBookPrice.owl#BookPrice | Price of a book |
| http://www.mindswap.org/2004/owl-s/1.1/AmazonBookPrice.owl#BookInfo | Information about a book |
| http://www.daml.org/services/owl-s/AmazonWS/1.1/AWSProcess.owl#AmazonProcess | Process model of a composite process used in a Web Service consisting of three processes: Shopping for book process, browsing for book process and a shopping cart process |

*Table 6: URIs of Resources in Web Services*

OWL-MX takes advantage of this approach to enhance a semantic matchmaking application using implicit information: “These approaches [standard means of description logic reasoning] do not exploit semantics that are implicit, for example, in patterns or relative frequencies of terms in service descriptions as computed by techniques from data mining, linguistics, or content-based information retrieval (IR). The objective of hybrid semantic Web service matching is to improve semantic service retrieval performance by appropriately exploiting means of both crisp logic based and approximate semantic matching where each of them alone would fail.” [Klusch et al. 2006]

Resources in OWL-S are identified through a unique URI. A URI can be regarded as a string representation. Thus, different string-based measurements can be used to calculate similarity between two URIs to determine a match. Appropriate string measurements need to be selected to perform the comparison. Furthermore, the strings need to be prepared in such a fashion that only relevant information is considered for the comparison to reduce noise and improve accuracy.

#### *3.5.1.3 Graph Similarity*

An OWL-S Web Service can be described as a labeled and directed graph. This graph can be used to measure similarity between two different Web Services. In particular, the similarity of the structure can be of useful information to determine whether there is a significant relation. The representation of the OWL-S service can be reduced to emphasize the structural information. Specifically, the following information is of importance:

- Number of inputs and outputs of a process
- Composition of composite processes

20 See previous chapter - Related Work

Significant similarity between two graphs of OWL-S services can be used as an indicator of relevancy. [Baggenstos 2006] describes similarities between graphs in detail.

## *3.6 Reuse*

Reuse or case adaption is the process of transforming a solution retrieved into a solution appropriate for the current problem. A number of approaches can be taken to carry out case adaption:

- The case returned could be used as a solution to the current problem without modification.
- The steps or processes that were followed to obtain the earlier solution could be rerun without modification, or with modifications where the steps taken in the previous solution are not fully satisfactory in the current solution.
- Where more than one case has been retrieved, a solution could be derived from multiple cases or, alternatively, several alternative solutions could be presented.

The system should implement various strategies to perform the adaption process. Again, it should also be able to allow future strategies to be implemented.
The following strategies should be implemented:

| **Strategy** | **Description** |
|---|---|
| Simple | The best matching case is returned unaltered |
| Copy | The existing case is extended by adding the full old case |
| Insert | The existing case is extended by adding parts of the old case |

*Table 7: Adaption Strategies*

The major challenge connecting cases with each other is to find the appropriate entry points and the correct bindings. The bindings are clearly defined in within a case. However, once parts or single processes of a case are used and combined with an existing fragment, the appropriate bindings need to be determined. The system should attempt to make best guesses in terms what the most likely bindings are. This should be achieved by taking advantage of the various similarity strategies offered by the system.

The following table shows the pseudo code to determine bindings:

```
get outputs of previous process
get inputs of connecting process
for all outputs do {
      for all inputs do {
            determine best match input, output
      }
      if threshold met {
            add binding best match  input to output
      else {
            throw exception
      }
}
```

*Text 8: Pseudo Code Add Bindings*

If the similarity between parameters does not meet a specified threshold, the two processes can not be connected with each other and the attempt has failed.

## *3.7 Revising*

Once an appropriate solution has been generated and returned, there is some expectation that the solution will be tested in reality. To test a solution, it has to be considered both the way it may be tested and how the outcome of the test will be classified as success of failure. This means that some criteria need to be defined for the performance rating of a proposed solution. Using this real-world assessment, a Case-based Reasoning system can be updated to take into account any new information uncovered in the processing of the new solution. This information can be added to a system for two purposes: fist, the more information that is stored in a case base, the closer the match found in the case base is likely to be; second, adding information to the case base generally improves the solution that the system is able to create.

The task of revision in the context of NExT is mostly achieved through user interaction. The user ultimately decides whether the produced new solution is applicable or not. Often times, the system is also unable to perform the adaption fully automatically without user feedback. The user interaction is not within the scope of this thesis as it presents less challenges. Revising is therefore not pursued further.

## 3.8 Retaining

Learning may occur in a number of ways. The addition of a new problem, its solution, and the outcome to the case base is a common method. The addition of cases to the case base will increase the range of situations covered by the stored cases and reduce the average distance between an input vector and the closest stored vector. A second method of learning in a Case-based Reasoning system is using the solution's assessment to modify the indexes of the stored cases or to modify the criteria for a case retrieval. If a case has indexes that are not relevant to the specific context in which it should be retrieved, adjusting the indexes may increase the correlation between the occasions when a case is actually retrieved and the occasions when it ought to have been retrieved. Similarly, assessment of a solution's performance may lead to an improved understanding of the underlying causal model of the domain that can be used to improve adaption processing. If better ways can be found to modify cases with respect to the distance between the current and retrieved cases, the output solution will probably be improved.
When applying Case-based Reasoning systems for problem solving, there is always a trade-off between the number of cases to be stored in the case library and retrieval efficiency. The larger the case library, the greater the problem space covered. However, this would also downgrade system performance if the number of cases were to grow unacceptably high. Therefore, removing redundant or less useful cases to attain an acceptable error level is one of the most important tasks in maintaining Case-based Reasoning systems.
The central idea of Case-based Reasoning maintenance is to develop some measures for case competence, which is the range of problems that a Case-based Reasoning system can solve. Various properties may be useful, such as the size, distribution, and density of cases in the case base; the coverage of individual cases and the similarity and adaptation knowledge of a given system. Coverage refers to the set of problems that each case could solve, and reachability refers to the set of cases that could provide solutions to the current problem. The higher the density of cases, the greater the chances of having redundant cases. By expressing the density of cases as a function of case similarity, a suitable case deletion policy could be formulated for removing cases that are highly reachable from others.
Another reason for Case-based Reasoning maintenance is the possible existence of conflicting cases in the case library due to changes in domain knowledge or specific environments for a given task. For example, more powerful cases may exist that can contain inconsistent information, either with other parts of the same case or with original cases that are more primitive. Furthermore, if two cases are considered equivalent, or if one case subsumes another by having more feature criteria, a maintenance process may be required to remove the redundant cases.
The task of retention is dependent on the revision task as only revised solutions should be retained. Retention and case maintenance is therefore not studied in more detail for this particular application at this time.

# 4 Implementation

This chapter takes the findings outlined in the design chapter and translates them into concrete implementations. Various approaches are discussed and highlighted. The implemented system should be regarded as a framework of proof of concept, but also as a basis for future related work.

## *4.1 Overview*

The focus of the implemented system is on the following parts:

1. Recording OWL-S services
2. Developing a generic frame work for a Case-based Reasoner for OWL-S
3. Developing an appropriate structure of a case
4. Designing a case base
5. Implementation of similarity measures between cases
6. Implementation of adaption strategies

### 4.1.1 Used Technologies

The framework is implemented in Java and tries to keep as close as possible to OWL-S for any persistent information storage. It uses functionality provided by two main APIs which are briefly described in more detail below.

**Mindswap OWL-S API**
The framework is taking advantage of the functionality provided by the OWL-S API developed by MINDSWAP which is a group of people working with Semantic Web technology inside the MIND LAB at University of Maryland Institute for Advanced Computer Studies.
The OWL-S API provides a Java API for programmatic access to read, execute and write OWL-S service descriptions. The API supports to read different versions of OWL-S (OWL-S 1.1, OWL-S 1.0, OWL-S 0.9, DAML-S 0.7) descriptions. The API provides an ExecutionEngine that can invoke AtomicProcesses that has WSDL or UPnP groundings, and CompositeProcecesses that uses control constructs Sequence, Unordered, Split, If-Then-else and RepeatUntil.

**SimPack**
The Department of Informatics at the University of Zurich has developed a similarity measurement toolkit called SimPack[21]. Various measurement approaches are included to compare strings, vectors, sequences, trees and graphs. In addition, the package implements the measures from the SecondString[22], the SimMetrics[23], and the OWLS-MX[24] projects.

## *4.2 Framework*

The core of the framework is the class OWLSCaseBasedReasoner which loads the case base and implements the main methods to retrieve and reuse cases.

The following UML diagram shows an overview:

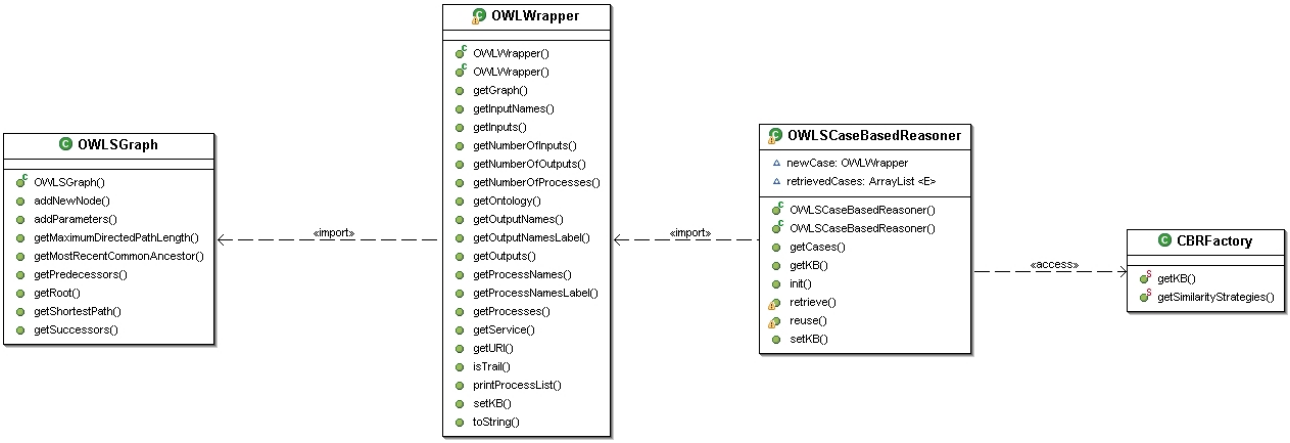


*Figure 9: UML Diagram of OWL-S Case-based Reasoner*

After the OWL-S Case-based Reasoner has been initialized, a new case in the form of an OWL ontology can be presented to the system to initiate the retrieval of appropriate cases. A subsequent call of the reuse method then attempts to apply various adaption strategies on the previously selected cases.
The OWL ontology needs to be a valid OWL-S description, but does not have to be complete in the sense that it is executable.

## *4.3 Recording Cases / Execution Trails*

The functionality of the Execution Trail is to store all relevant information that will be used to do the matching against new cases. As a means to store this information, a new standard OWL-S Service ontology is created. The main reason behind choosing this format is to be able to take advantage of the various tools that are available to interact with OWL-S ontologies. Furthermore, this allows to seamlessly build on top of the OWL-S API

21 http://www.ifi.unizh.ch/ddis/SimPack.html
22 http://secondstring.sourceforge.net/
23 http://www.dcs.shef.ac.uk/~sam/simmetrics.html
24 http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/index.html

and easily use the Pellet[25] reasoner to perform semantic matching. Potentially, this also has the advantage that the recorded cases can be executed again.
The resulting service wraps the executed service and its components. The used processes are stored in a sequence which may consist of sub-sequences depending whether the process is atomic or composite. Furthermore, the various inputs and outputs, if any, are replicated as well. These are then also associated with the specific values used in this particular execution and stored in the parameter valueData. Additionally, to maintain a reference to the original ontology used, the label property is used to store the respective URI.
The flexibility of the underlying Web Ontology Language OWL allows to extend this concept to include other custom data that may be required.

Example of a recorded trail of the simple Web Service #DictionaryService[26] (only the service description, the profile and the associated process are shown:

```
<?xml version="1.0"?>
<rdf:RDF
  xml:base="http://www.ifi.unizh.ch/ddis/ont/owl-s/trails/ExecutionTrail-1169902906765.owl#">
  <service:Service rdf:ID="ExecutionTrailService">
    <rdfs:label>http://www.mindswap.org/2004/owl-s/1.1/Dictionary.owl#DictionaryService</rdfs:label>
    <service:presents>
      <profile:Profile rdf:ID="ExecutionTrailProfile"/>
    </service:presents>
    <service:describedBy>
      <process:AtomicProcess rdf:ID="Process1"/>
    </service:describedBy>
  </service:Service>
```

*Text 9: Example of the Service of an Execution Trail*

```
  <profile:Profile rdf:about="#ExecutionTrailProfile">
    <service:presentedBy rdf:resource="#ExecutionTrailService"/>
    <profile:hasInput>
      <process:Input rdf:ID="Input1">
        <rdfs:label>http://www.mindswap.org/2004/owl-s/1.1/Dictionary.owl#InputString</rdfs:label>
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
        >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
        <process:valueData>hello</process:valueData>
      </process:Input>
    </profile:hasInput>
    <profile:hasOutput>
      <process:Output rdf:ID="Output1">
        <rdfs:label>http://www.mindswap.org/2004/owl-s/1.1/Dictionary.owl#OutputString</rdfs:label>
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
        >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
        <process:valueData rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >1) A loud exclamation; a call to invite attention or to incite a person or an animal; a shout. 2)
To cry out; to exclaim with a loud voice; to call to a person, as by the word halloo. 3) To encourage with
shouts. 4) To chase with shouts or outcries. 5) To call or shout to; to hail. 6) An exclamation to call
attention or to encourage one.</process:valueData>
      </process:Output>
    </profile:hasOutput>
    <rdfs:label>[null]</rdfs:label>
    <profile:textDescription>[null]</profile:textDescription>
  </profile:Profile>
```

*Text 10: Example of the Profile of an Execution Trail*

25 http://pellet.owldl.com/
26 http://www.mindswap.org/2004/owl-s/1.1/Dictionary.owl#DictionaryService

```
  <process:AtomicProcess rdf:about="#Process1">
    <rdfs:label>http://www.mindswap.org/2004/owl-s/1.1/Dictionary.owl#DictionaryProcess</rdfs:label>
    <j.0:hasPerform>
      <process:Perform rdf:ID="Perform1"/>
    </j.0:hasPerform>
    <process:hasInput rdf:resource="#Input1"/>
    <process:hasOutput rdf:resource="#Output1"/>
    <service:describes rdf:resource="#ExecutionTrailService"/>
  </process:AtomicProcess>
  <process:Perform rdf:about="#Perform1">
    <process:process rdf:resource="#Process1"/>
  </process:Perform>
```

Text 11: Example of the Composite Process of an Execution Trail

The OWL-S API allows to add a listener to its execution engine to be notified when certain events happen. The following table shows the events of notification and their description:

| Event | Description |
|---|---|
| ExecutionFailed | Called when the execution fails due to an exception |
| ExecutionFinished | Called after the execution of a process finishes |
| ExecutionStarted | Called before the execution of a process starts |

*Table 12: OWL-S API Execution Engine Events*

To record the execution, a custom listener has been implemented to retain such information as executed process and specific values consumed by its inputs. The recorded trail is then on-the-fly compiled into an OWL-S ontology.

## *4.4 Case Base*

The case base constitutes of the different cases that have previously been recorded. To advertise the different cases, the case base maintains an ontology TrailList.owl which imports the ontologies associated with the different cases. This allows to continue to purely rely on OWL to store the data that is need.

```
<?xml version="1.0"?>
<rdf:RDF >
<owl:Ontology rdf:about="http://www.ifi.unizh.ch/ddis/ont/owl-s/trails/TrailList">
<owl:imports rdf:resource="http://www.ifi.unizh.ch/ddis/ont/owl-s/trails/ExecutionTrail1.owl"/>
<owl:imports rdf:resource="http://www.ifi.unizh.ch/ddis/ont/owl-s/trails/ExecutionTrail2.owl"/>
<owl:imports rdf:resource="http://www.ifi.unizh.ch/ddis/ont/owl-s/trails/ExecutionTrail3.owl"/>
<owl:imports rdf:resource="http://www.ifi.unizh.ch/ddis/ont/owl-s/trails/ExecutionTrail4.owl"/>
</owl:Ontology>
</rdf:RDF>
```

*Text 13: Example of TrailList.owl*

In order to make the cases accessible, the trail list ontology is loaded into a OWLKnowledgeBase[27] object of the OWL-S API. Once read into the knowledge base, a reference of each case is kept internally.

## *4.5 Retrieve*

The following figure shows the UML diagram as an overview of the similarity measures used and its structure:

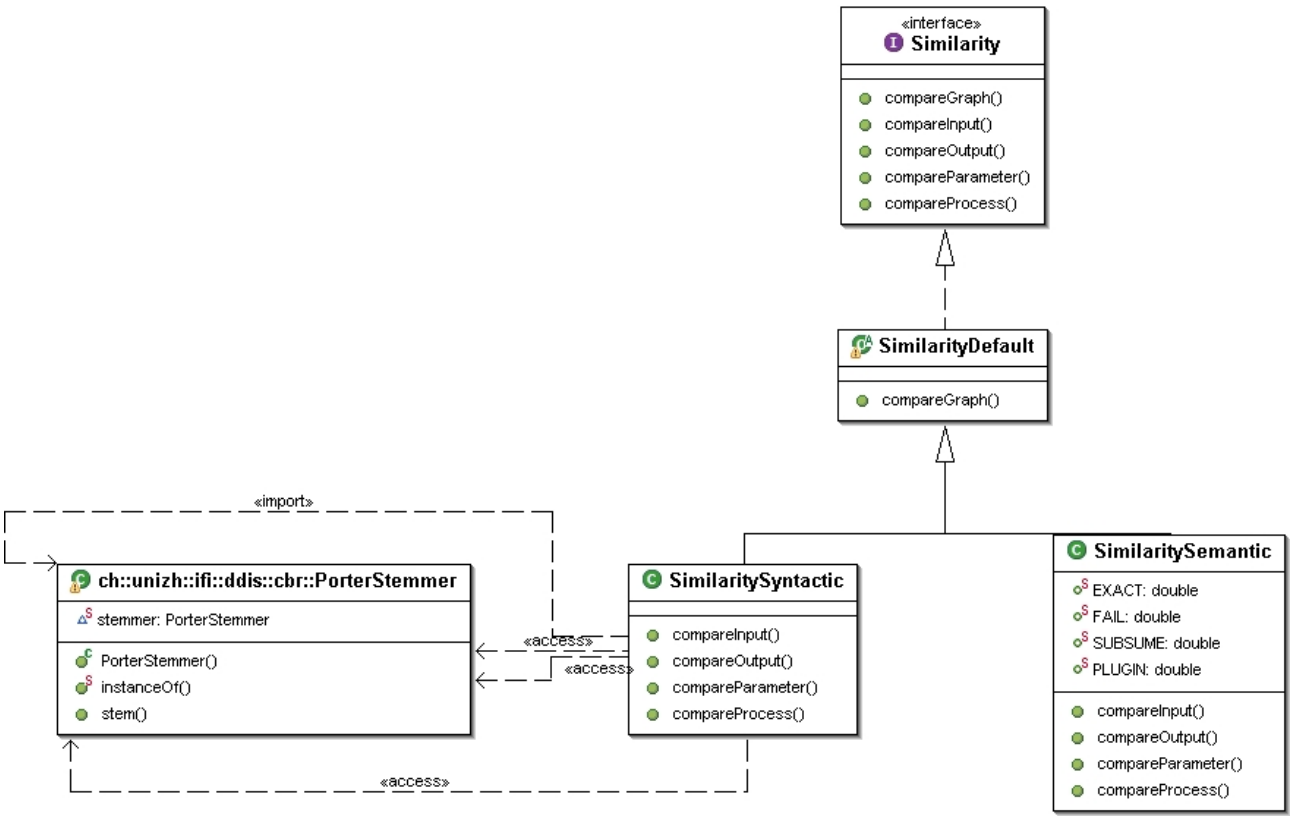


*Figure 10: UML Diagram Similarity*

### 4.5.1 OWL-S Graph

In order to take advantage of the similarity measures for graphs provided by SimPack, a graph in the form of a AbstractGraphAccessor has to be provided. SimPack is designed to be easily extended and its structure allows for custom accessors to use its various similarity functions. In SimPack, a graph is represented as set of directed and labeled nodes. See [Baggenstos 2006] for further details regarding graph representation in SimPack.
As only the structure and its elements of the graph is relevant, the OWL-S service is presented as a simple, directed graph with its elements as labels. However, SimPack would also allow to use semantics as it can hold any object to represent the label of a node. In this particular case, strings in the form of Service, AtomicProcess, CompositeProcess, Input, Output, Sequence, IfThenElse etc. are used.

27 http://www.mindswap.org/2004/owl-s/api/doc/javadoc/org/mindswap/owl/OWLKnowledgeBase.html

The following illustration shows a few examples of graph representations of various OWL-S services:

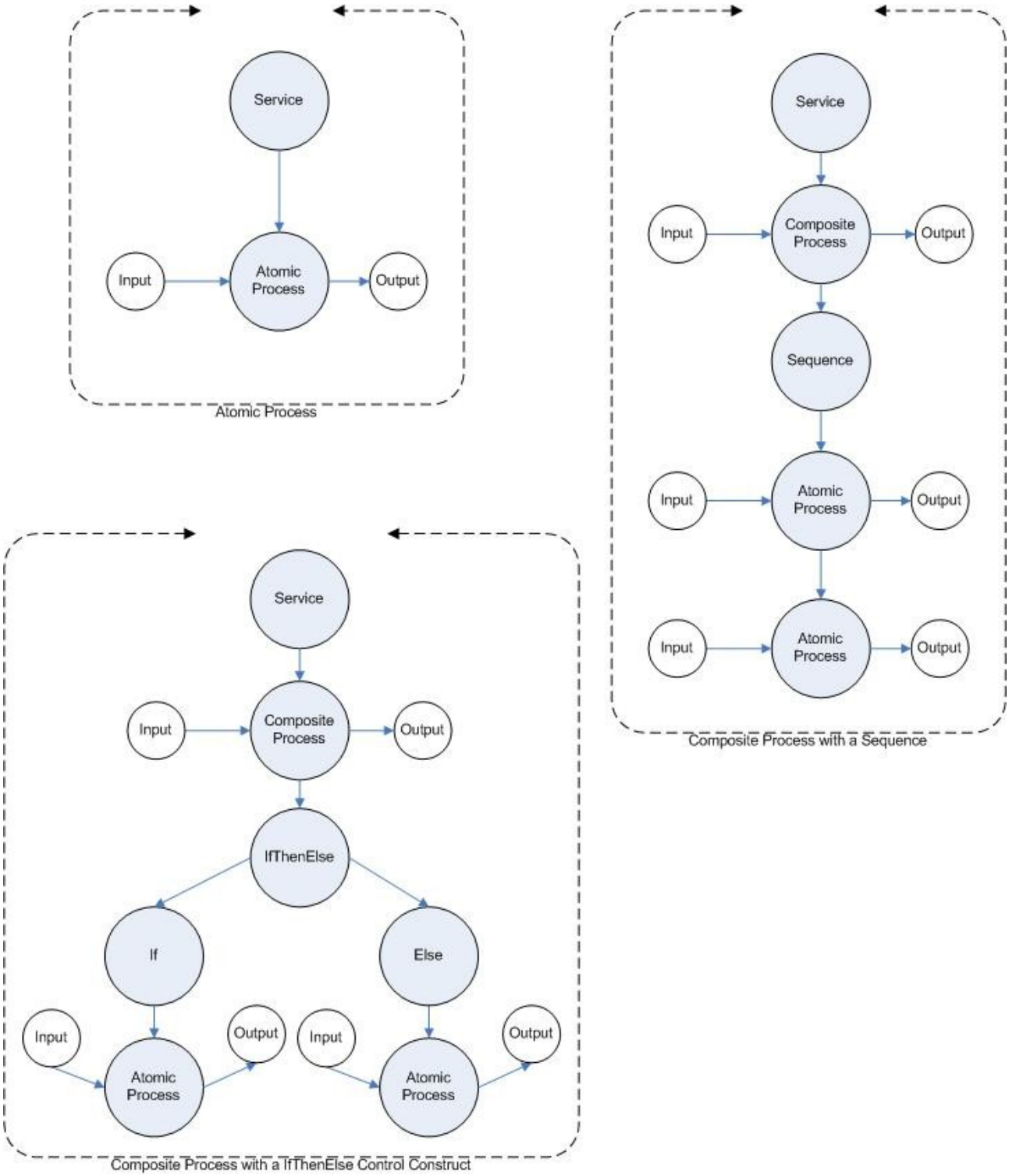


*Figure 11: Graph Representations of various OWL-S Services*

### 4.5.2 Similarity Strategies

The various similarity strategies are implemented inheriting from a super class which is implemented in the form of an interface. The following methods need to be implemented by the concrete realizations of the various similarity strategies:

```
public interface Similarity {

	// implements method to compare parameters
	public double compareParameter(Parameter oldParameter, Parameter newParameter);

	// implements method to compare inputs
	public double compareInput(Input oldInput, Input newInput);

	// implements method to compare outputs
	public double compareOutput(Output output, Output newOutput);

	// implements method to compare processes
	public double compareProcess(Process oldProcess, Process newProcess);

	// implements method to compare OWLS Graph
	public double compareGraph(OWLSGraph oldGraph, OWLSGraph newGraph);


}
```

*Text 14: Interface Similarity*

The OWL-S Case-based Reasoner uses the CBR Factory to retrieve the names of the classes of the various implemented similarity strategies. These classes are then instantiated during runtime and used to perform the comparison of the new case with the existing cases.
To perform the actual comparison of the old case and the new case, each input, output and process is compared with each other. The way of comparing is determined by the various strategies that have been implemented. The following pseudo code shows the similarity loop:

```
	For each Similarity Strategy
		get Similarity Inputs oldCase, newCase
		get Similarity Outputs oldCase, newCase
		get Similarity Processes oldCase, newCase
		get Similarity Graph oldCase, newCase
```

*Text 15: Similarity Loop*

The determination of the best matching cases is based on the assumption that the more matches are found in an existing case, the more suitable it is for reuse. For example, if the new case uses a string as its input, the more inputs of type string an existing case uses, the more suitable it will be. Therefore, the implemented similarity algorithm uses the different similarity strategies to measure the similarity between the old case and the new case and returns a weighted average of the matches relative to the total amount in the old case. For example, to compute a similarity measurement of the inputs between an old case and a new case:

Let A be the set of inputs of the old case and B the set of inputs of the new case. C then is the intersection of A and B holding any inputs that are present in both the old and new case as determined by the different similarity strategies:

$$A \in Inputs_{OldCase}, B \in Inputs_{NewCase}, C = A \cap B \tag{4.1}$$

The similarity of inputs is then calculated as the relative mean of the population of C over the population of A:

$$\mathrm{Sim}_{Inputs} = \frac{\#(C)}{\#(A)} \tag{4.2}$$

The same logic is applied to determine the similarity of outputs and processes. Inputs, outputs and processes (IOPs) are then weighted to provide an overall similarity measure for IOPs. The following table shows the current weights assigned:

| Type | Weight |
|---|---|
| Inputs | 0.6 |
| Outputs | 0.3 |
| Process | 0.2 |

*Table 16: IOP Weights*

The reasoning behind assigning different weights is based on the assumption that inputs are in most cases a given starting point. Outputs can potentially undergo an additional transformation afterwards and there might be different processes that perform the same or similar tasks. However, the weights can easily be customized if needed.

Additionally, the graph of the the old case and the new case is compared using SimPack's graph similarity method. This is described in more detail below.

If the overall similarity of the IOPs and the graph similarity reach a certain threshold, the specific case is remembered and will be reused.

The following pseudo code shows the main loop of the comparison:

```
For each old case
      get Similarity oldCase newCase
      for each similarity match {
            similarityIOPs = (similarityInputs * inputsWeight)
                        + (similarityOutputs * outputsWeight)
                        + (similarityProcesses * processesWeight);
            if(similarityGraph >= graphThreshold
                        && similarityIOPs >= IOPsThreshold) {
                  consider oldCase for reuse
                  break;
            }
      }
```

*Table 17: Main Loop Case Similarity*

#### *4.5.2.1 Semantic Matching*

Semantic matching is mainly performed on the various parameters of the old case and new case. Each input and output has a specific datatype. Therefore, based on the fact that parameters in OWL-S hold a semantic meaning, the algorithm presented in [Massimo et al. 2002] is adapted to compute a similarity measurement between two parameters.
The same matching types as used by [Massimo et al. 2002] are implemented and assigned a value so that:

$$0 < \text{sim} < 1 \tag{4.3}$$

The following table shows the type of matches and their corresponding weight assigned:

| **Type of Match** | **Weight** |
|---|---|
| Exact | 1.0 |
| Subsume | 0.75 |
| Plug In | 0.5 |
| Fail | 0.0 |

*Table 18: Type-Weight Matrix*

To perform the datatype reasoning, the Pellet reasoner is used. Pellet is an open-source OWL-DL reasoner written in Java, originally developed at the University of Maryland's Mindswap Lab. Pellet is based on the tableaux algorithms developed for expressive Description Logics (DL). It supports the full expressivity OWL-DL including reasoning about nominals (enumerated classes). In addition to OWL-DL, Pellet supports all the features proposed in OWL 1.1, with the exception of n-ary datatypes.

The type of a process is either atomic or composite. This information does not hold useful meaning to determine a match. Therefore, processes are compared using their unique URI to determine a match.

#### *4.5.2.2 Syntactic Matching*

The URIs of the elements to be compared are used as string representations to compute similarities. Traditional methods of content-based information retrieval have been used. SimPack contains a number of measurements implemented that are applicable to strings. Specifically, the following three measurements are used:

**Cosine Similarity**
Cosine similarity is a common vector based similarity measure. Whereby the input string is transformed into vector space so that the Euclidean cosine rule can be used to determine similarity. The cosine similarity is often paired with other approaches to limit the dimensionality of the problem. For instance with simple strings a list of stopwords are used to exclude from the dimensionality of the comparison. In theory this problem has as many dimensions as terms exist.

$$\mathrm{sim}_{\cos}(Old, New) = \frac{\vec{Old} \cdot \vec{New}}{\|\vec{Old}\|_2^2 \cdot \|\vec{New}\|_2^2} \tag{4.4}$$

with standard TFIDF term weighting scheme, and the unfolded URI of the old case Old and new case New are represented as n-dimensional weighted index term vectors $\vec{Old}$ and $\vec{New}$ respectively.

$\vec{Old} \cdot \vec{New} = \sum_{n}^{i=1} w_{i,Old} \times w_{i,New}$, $\|X\|_2 = \sqrt{\sum_{i}^{n} w_{i,x}^2}$ and $w_{i,x}$ denotes the weight of the i-th index term in vector X.

**Extended Jaccard Measure**
The binary Jaccard coefficient measures the degree of overlap between two sets and is computed as the ratio of the number of shared attributes $\vec{Old}$ AND $\vec{New}$ to the number possessed by $\vec{Old}$ OR $\vec{New}$ .

$$\mathrm{sim}_{EJ}(Old, New) = \frac{\vec{Old} \cdot \vec{New}}{\|\vec{Old}\|_2^2 + \|\vec{New}\|_2^2 - \vec{Old} \cdot \vec{New}} \tag{4.5}$$

with standard TFIDF term weighting scheme.

**Jensen Shannon Measure**
The measure is based on the information-theoretic, non-symmetrical Kullback-Leibler divergence measure. It measures the pairwise dissimilarity of conditional probability term distributions between the URI of the old case and new case.

$$\frac{1}{2\log 2} \sum_{i=1}^{n} h(p_{i,Old}) + h(p_{i,New}) - h(p_{i,Old} + p_{i,New}) \tag{4.6}$$

with probability term frequency weighting scheme: $p_{i,Old}$ denotes the probability of the i-th index term occurrence in the URI of the old case and h(x) = -x log x.

SimPack has implemented the above measurements and makes them available.

Before the URIs are used for comparison, they are unfolded and common terms are removed. The following table shows common terms that are either removed or replaced:

| *Term(s)* | *Action* |
|---|---|
| Http, www, .owl | Removed |
| [0..9] | Removed |
| /, #, ., _, - | Replaced with white space |

*Table 19: URI Term Processing*

Strings are further unfolded as much as possible. For example, words are commonly joined by using capitalization. For example, BookStore will be split into the two terms Book and Store.

Additionally, the resulting strings are stemmed using the Porter Stemming[28] algorithm. The algorithm tries to reduce terms to its stem. The algorithm is given an explicit list of suffixes, and, with each suffix, the criterion under which it may be removed from a word to leave a valid stem. The main purpose of using stemming is to improve performance of the similarity measures introduced earlier. Additionally, the implemented version of the Porter Stemming algorithm uses a list of common stop words which are removed completely.

The result of the three similarity measures are added with each other and the actual similarity returned as an average:

$$0 < \mathrm{sim}_{Cosine}, \mathrm{sim}_{Extended}\ Jaccard, \mathrm{sim}_{Jensen}\ Shannon < 1 \tag{4.7}$$

$$\mathrm{sim}_{Inputs}, \mathrm{sim}_{Outputs}, \mathrm{sim}_{Processes} = \frac{(\mathrm{sim}_{Cosine} + \mathrm{sim}_{Extended\ Jaccard} + \mathrm{sim}_{Jensen\,Shannon})}{3} \tag{4.8}$$

$$0 < \mathrm{sim}_{Inputs}, \mathrm{sim}_{Outputs}, \mathrm{sim}_{Processes} < 1 \tag{4.9}$$

The three similarity measures are expected to perform more or less equally. The main reason for taking the average of all three is to demonstrate how different measurements can be easily implemented. It has to be noted, that using Java's Reflection API, any similarity measurement provided by SimPack accepting either two strings or an IsequenceAccessor[29] object can be used.

#### *4.5.2.3 Graph Matching*

As previously introduced, the graph represents the structure of a Web Service described in OWL-S. Comparison of the structure allows to respect the number of parameters used in a Web Service, but also the structure of the various control constructs in a composite process. The graphs are compared in a manner that the smaller graph is searched in the bigger graph.
To compare the graph similarity of the old case and the new case, the Valiente's maximum

28 http://www.tartarus.org/~martin/PorterStemmer/def.txt
29 See SimPack.api.Interface ISequenceAccessor

common subgraph algorithm is used. SimPack implements this algorithm and exposes it in the class MaxCommonSubgraphIsoValiente[30]. The result is a measurement of the similarity of the two graphs:

$$0 < \mathrm{sim}_{Graph} < 1 \tag{4.10}$$

#### *4.5.2.4 Future Implementations*

As described earlier, future implementations of different similarity measurements can be added by inheriting from the super class and implementing the relevant concrete methods to perform the similarity measures. The newly added class can then be announced in the CBR Factory which will make it available to the system.

30 See SimPack.measure.graph.MaxCommonSubgraphIsoValiente

## 4.6 Reuse

The following figure shows the UML diagram of the various adaption strategies implemented:


*Figure 12: UML Diagram Adaption*

A interface declares a single method `adapt()` accepting a new and old case as its parameters. Concrete instantiations then implement the various adaption strategies that will be performed. The framework attempts to apply one strategy. If an exception occurs, the next strategy is tried.

Currently, the following strategies are implemented in the order shown in the following table:

| Order | Strategy | Description |
|---|---|---|
| 1 | Insert | The existing case is extended by adding parts of the old case |
| 2 | Copy | The existing case is extended by adding the full old case |
| 3 | Simple | The best matching case is returned unaltered |

*Table 20: Overview Adaption Strategy*

As seen in the table above, the default strategy is to return the old case unaltered if all other strategies fail.

### 4.6.1 Adaption Strategy Simple

This strategy returns the old case unaltered. The basic assumption of Case-based Reasoning is that past knowledge can be used to solve new problems. In this most basic case, it is assumed that the old case is presented to a user which may get inspiration of the case presented to solve his existing problem. Due to the fact that this case has previously been selected during the retrieve process, the system determined a certain relevance to the new case.

### 4.6.2 Adaption Strategy Copy

This strategy tries to append the selected old case to the new case as a whole. It takes the first process of the old case and tries to allocate correct bindings to the last process of the new case. To determine matching Output-Input bindings, the various implemented similarity strategies are leveraged. In case no matching binding can be established, an exception is thrown and the attempt to append the old case to the new one is considered as failed.

The following figure illustrates the strategy described above:

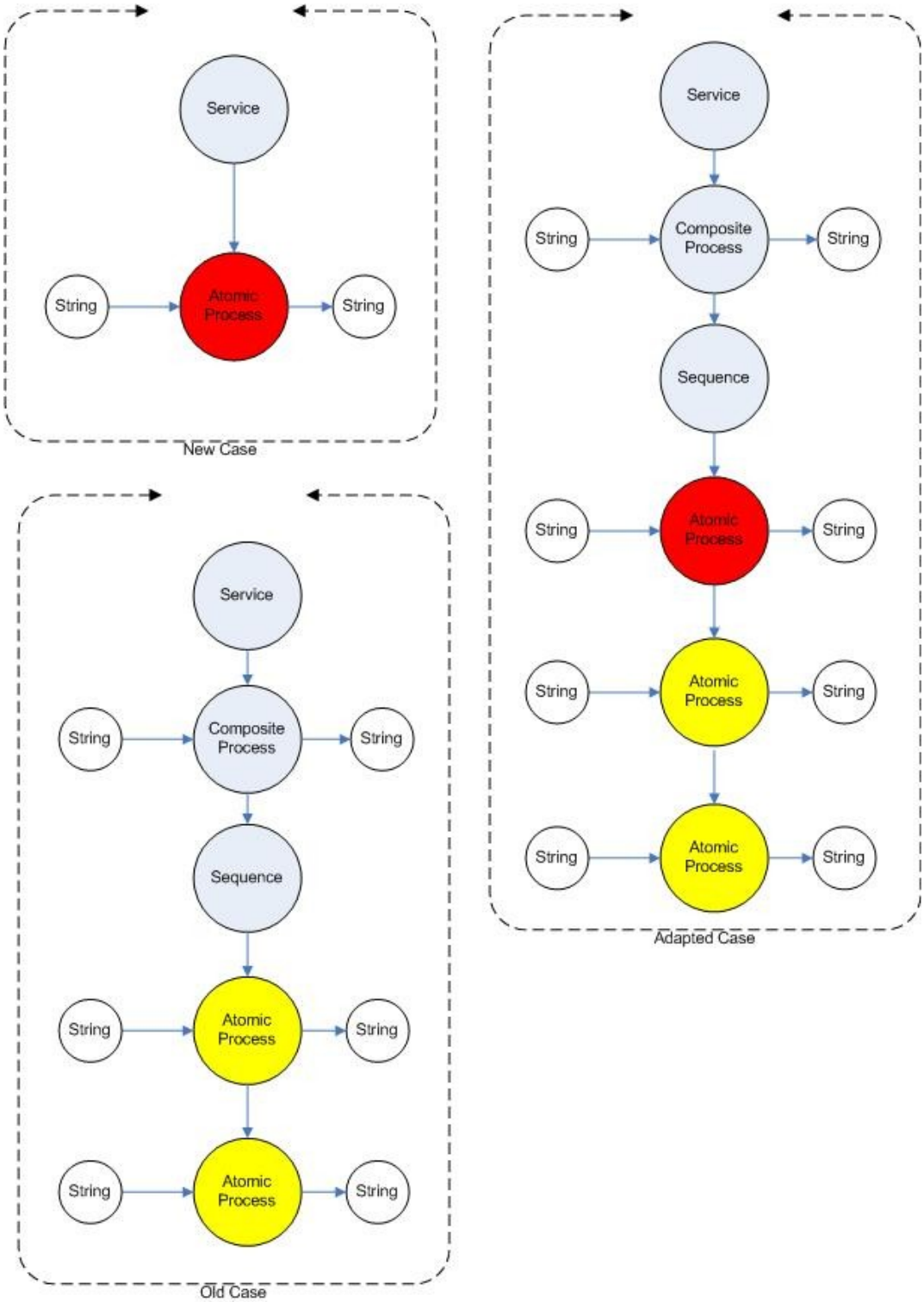


*Figure 13: Adaption Strategy Copy*

### 4.6.3 Adaption Strategy Insert

The approach in this strategy is to search for the first occurrence of the last process of the new case in the old case. If a match has been found, the remaining processes of the old case are appended to the new case. In the same fashion, the various similarity strategies implemented are used to determine a match. In case no match is found, an exception is thrown and the process aborted.
The following figure illustrates the insertion strategy:

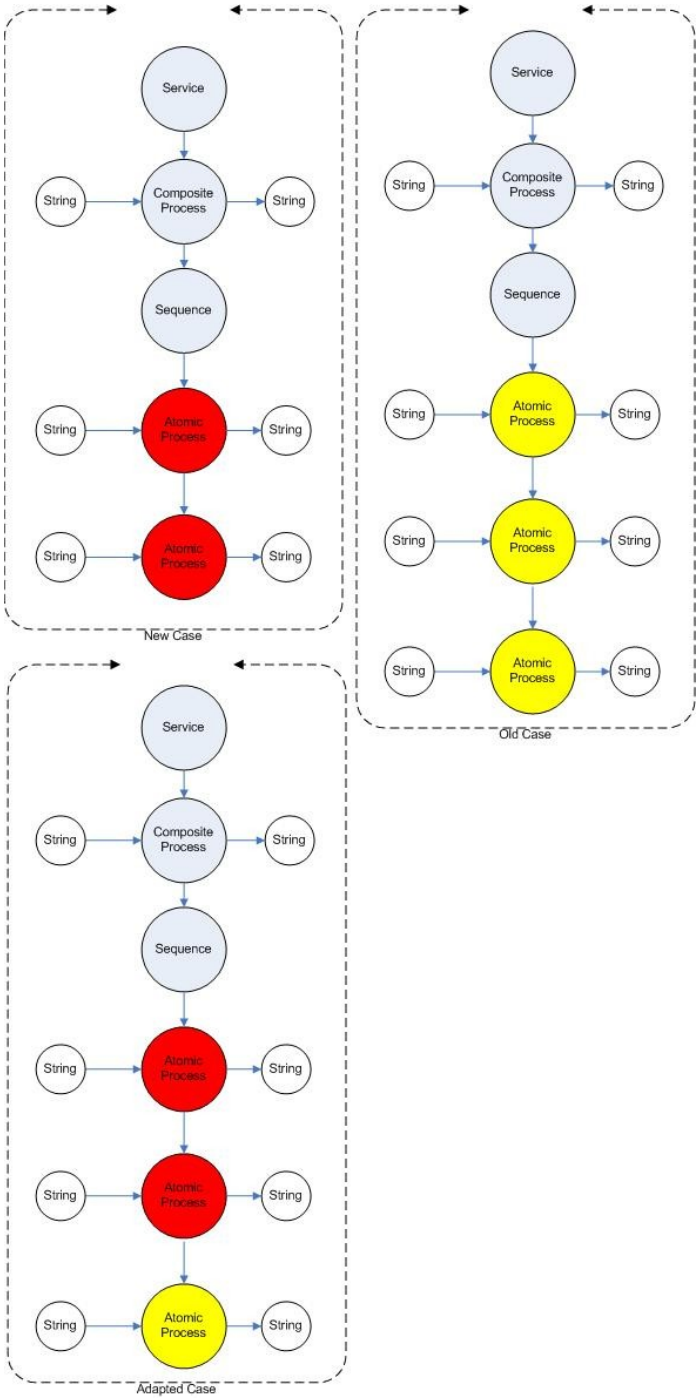


*Figure 15: Adaption Strategy Insert*

## 4.7 Revising

In the context of NExT, the user is taking the task of validating the proposed solution and this functionality is therefore not implemented.

## 4.8 Retaining

This aspect of the Case-based Reasoning system is less interesting and due to its dependency of the revising task not implemented.

# 5 Evaluation

This chapter describes a possible evaluation approach to test the performance of the implemented system.

## *5.1 Setting*

**Case Base**
A suitable case base should be built. The cases should be representative of a specific domain. In addition, the cases should consist of various degrees of complexity in terms of the amount of parameters, processes, and control constructs, used. The case base should then be filled with additional random cases to add noise to the system.

The following cases should be recorded:

*Recorded Case #1*

- Atomic process with one input and one output

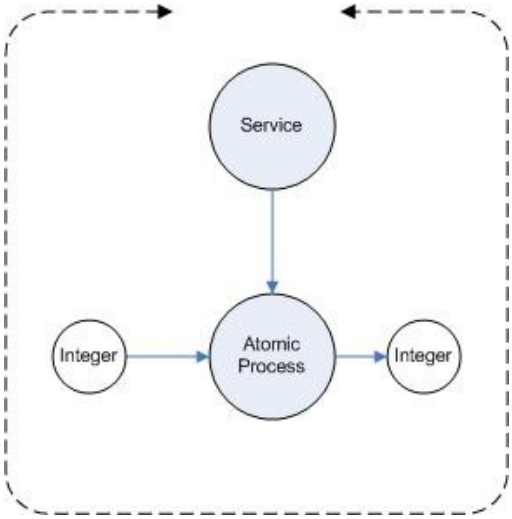


*Figure 16: Recorded Case #1*

*Recorded Case #2*

- Atomic process with two inputs and one output

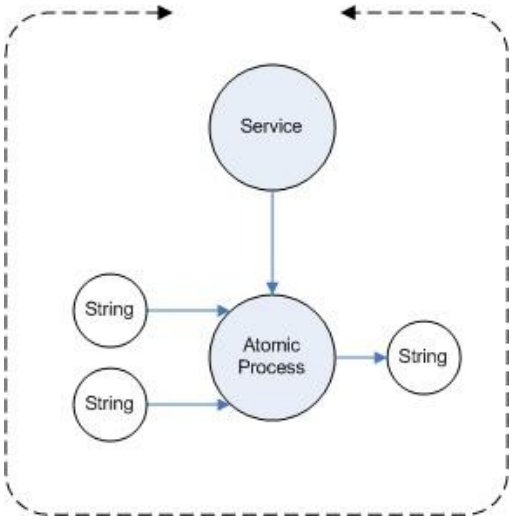


*Figure 17: Recorded Case #2*

*Recorded Case #3*

- Composite process with one input and one output, using a sequence of three processes

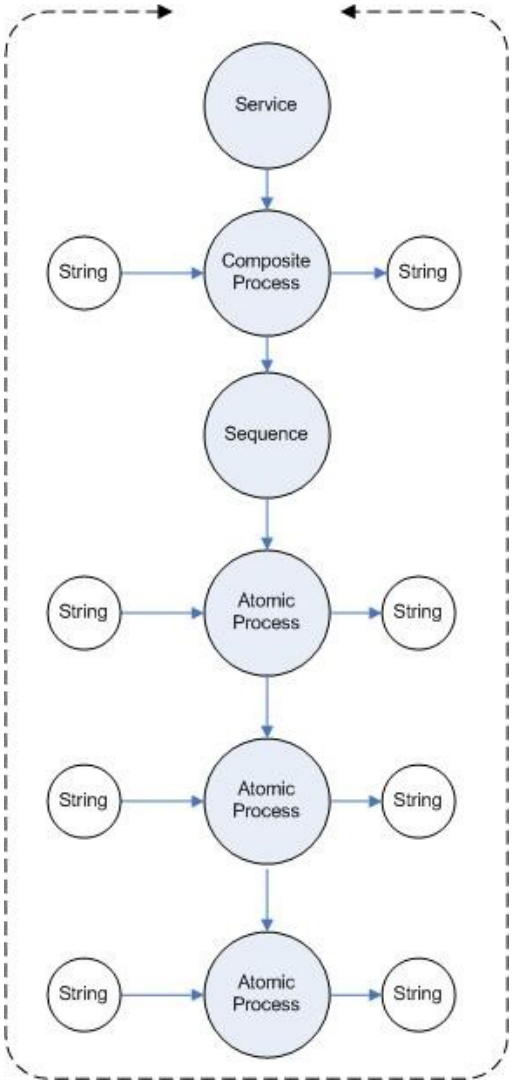


*Figure 18: Recorded Case #3*

*Recorded Case #4*

- Composite process with one input and one output, using an If-Then-Else control construct with a subsequent composite process, consisting of two atomic processes

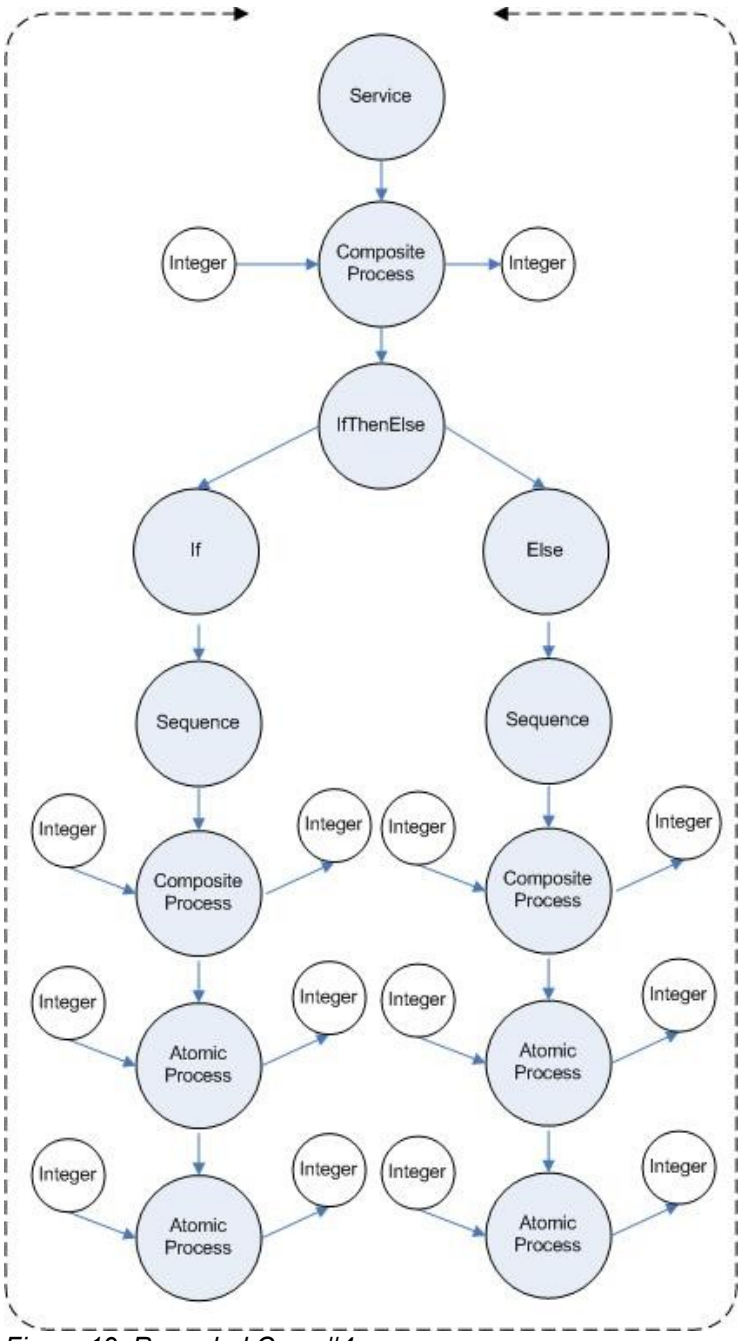


*Figure 19: Recorded Case #4*

**New Cases**
Suitable new cases should be designed that will be presented to the system. These cases should be related to the cases that are held in the case base. Following the recorded cases, the new cases below should be created:

*New Case #1*

- Atomic process with one input and one output

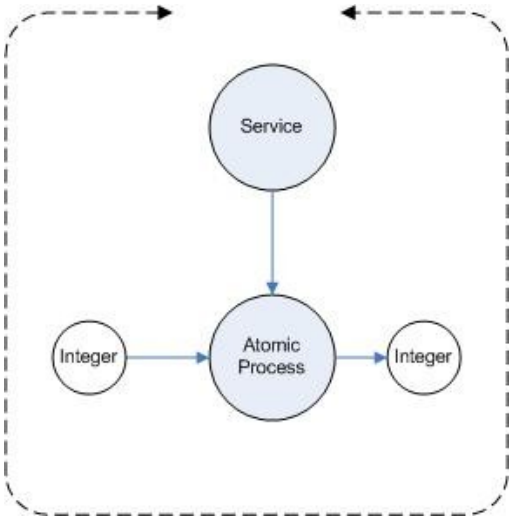


*Figure 20: New Case #1*

*New Case #2*

- Atomic process with two inputs and one output

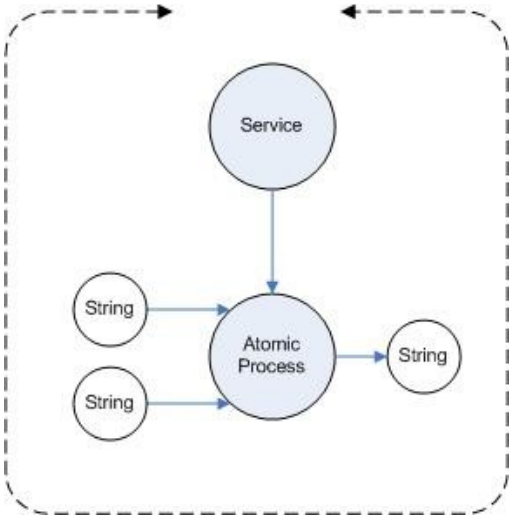


*Figure 21: New Case #2*

*New Case #3*

- Composite process with one input and one output, using a sequence of two processes

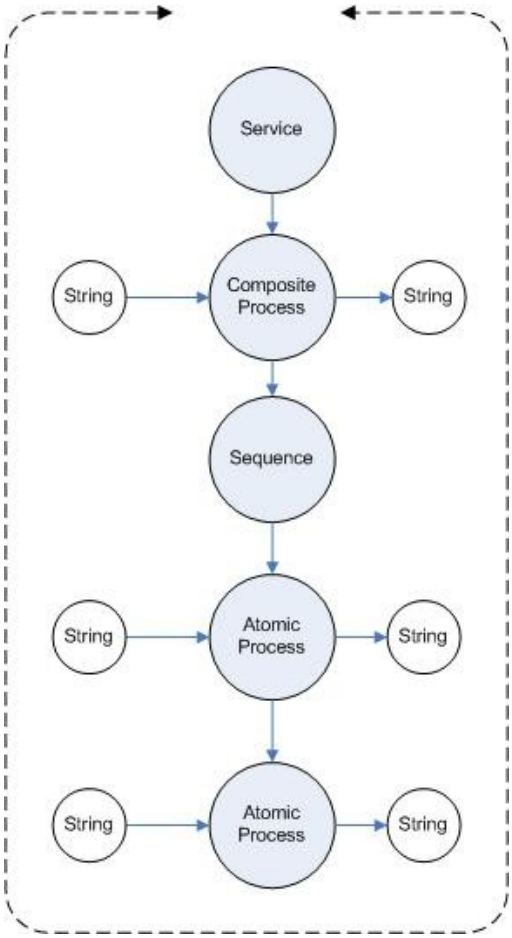


*Figure 22: New Case #3*

*New Case #4*

- Composite process with one input and one output, using an If-Then-Else control construct

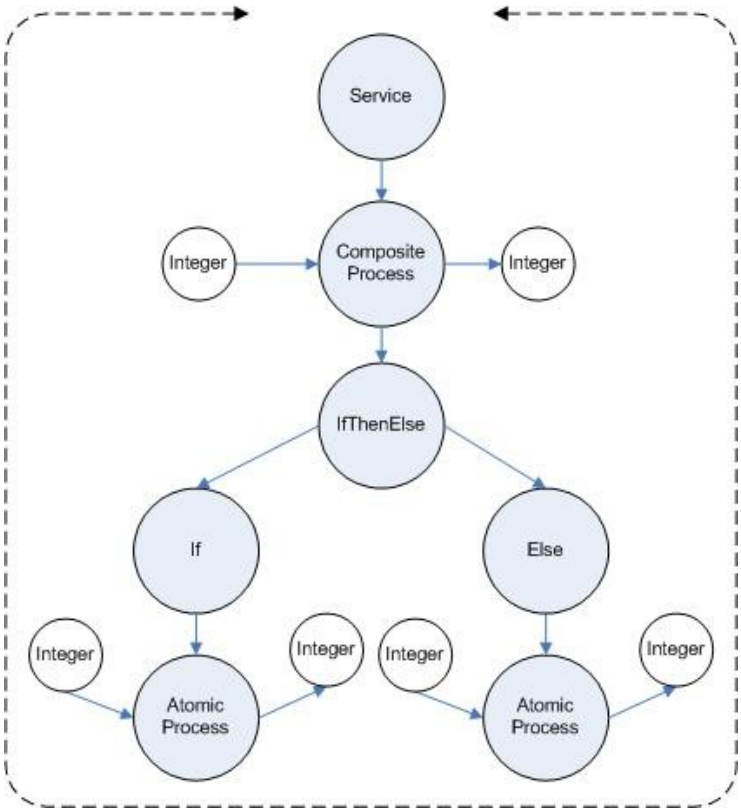


*Figure 23: New Case #4*

## *5.2 Approach*

In one scenario, the evaluation should test that the system is selecting the appropriate cases, and the implemented similarity measures are performing as expected. As a result, the system will be presented with various new cases and is expected to return the recorded case most similar to the new case.
In another scenario, the system's ability to adapt cases should be tested. This is achieved by evaluating the combination of the selected case and the new case.

## 5.3 Test Cases

The following scenarios should be tested:

**Test Case Similarity**
The purpose of this test case is to evaluate that the system is selecting the appropriate cases from the case base.
Each new case should be presented to the system individually, and the set of selected cases should be evaluated against the expected selection. The following table shows the various new cases and the expected outcome:

| Presented Case | Selected Cases |
|---|---|
| New case #1 | Recorded case #1 |
| New case #2 | Recorded case #2 |
| New case #3 | Recorded case #3 |
| New case #4 | Recorded case #4 |

*Table 21: Evaluation Table Test Case Similarity*

**Test Case Adaption**
The purpose of this test case is to evaluate the system's ability to adapt cases from its case base.
Assuming the system performed as expected in the previous test case, the outcome of adaption should be as in the following table:

| Presented Case | Performed Adaption |
|---|---|
| New case #1 | Return recorded case #1 unaltered |
| New case #2 | Return recorded case #2 unaltered |
| New case #3 | Return recorded case #3 unaltered |
| New case #4 | Return recorded case #4 unaltered |

*Table 22: Evaluation Table Test Case Adaption*

# 6 Conclusion

## *6.1 Summary*

The goal of this thesis is to apply the techniques and concepts known by Case-based Reasoning, to the Semantic Web and Web Services described in OWL-S.
A framework has been introduced that follows the typical design of any Case-based Reasoner, with a focus on the more challenging tasks retrieval and reuse of cases.

A summary of the main components and achievements of this thesis follows:

**Case and Case Base**
Firstly, a suitable case structure that is able to capture executed OWL-S Web Services has been developed. This serves as the basis of the memory of the Case-based Reasoner. To take advantage of the expressiveness of the Web Language OWL and to remain compatible with OWL-S, the specific cases themselves are maintained as OWL-S Web Services. This allows for the design of a flexible case structure and has the advantage that past cases could be executed again as they have taken place in the past.
Furthermore, a case base which holds the relevant cases has been developed. The case base itself is again compatible with OWL-S and allows for a seamless build on the same technology.

**Retrieve**
Various similarity strategies have been implemented to retrieve similar cases from the case base. These have been measured against a novel potentially unknown problem to the system. Different approaches have been explored to enhance the system's capability to measure similarity. Specifically, the following areas have been taken advantage of:

*Semantic Web*
The Semantic Web and and the Web Language OWL allow powerful reasoning capabilities to make statements about relationships between resources.

*Information Retrieval*
The area of information retrieval provides powerful algorithms for automatic text analysis that can complement a purely semantic approach. Implicit information present in the URI of resources, can be exploited by applying techniques from content-based information retrieval.

*Graph theory*
In addition to focusing on single resources only, the information held by the overall structure of the Web Service represented. This is a labeled, directed graph which allows for the creation of relevant statements in terms of similarity. Techniques such as subgraph isomorphism can be applied to determine whether there exists a structural relation between two graphs.

In addition to the similarity approaches defined above, the system allows for the implementation of custom made similarity measures. These can easily be plugged into the existing framework.

**Reuse**
Selected cases in the retrieval task are further processed, applying different adaption strategies in an effort to complement the novel case with existing ones. Various degrees of complex strategies are applied in a descending order:

*Insert*
Parts of each previously selected case are extracted so that they can be added to the new case. The approach is to complement the new case with a relevant part of the existing one.

*Copy*
This strategy tries to complement the new case with the existing case as a whole. The existing case then attempts to be added to the end of the new case.

*Simple*
In case the previously mentioned strategies fail, the selected existing case is returned unaltered.

Similar to the approach of similarity strategies, the system is built to allow for additional strategies implemented in the future.

The feasibility of applying powerful techniques and concepts of Case-based Reasoning to the Semantic Web and OWL-S Web Services has been demonstrated. As the Semantic Web is more widely accepted and more applications take advantage of its advantages such as NExT, Case-based Reasoning can add significant value. Case-based Reasoning is based on the principle that past knowledge is useful to help solve novel situations. This approach is closely related to how humans often solve problems, and therefore a process that can be adopted naturally. It will be interesting to see further progress being made in this respect as more applications evolve and the Semantic Web is more of an everyday reality.

## *6.2 Future Work*

This thesis hopefully helps foster the progress of the Semantic Web in general and the application of Case-based Reasoning to the Semantic Web specifically. The framework introduced should serve as a basis and proof of the concept that Case-based Reasoning techniques can be applied to the Semantic Web. The remainder of this chapter highlights particular aspects where the system can be improved and enhanced. It also provides an outlook of how it can be used in a future where the Semantic Web is widely adopted.

### 6.2.1 Case and Case Base

Holding a case in OWL-S allows for taking full advantage of the functionality provided by it. At present, various features have not been taken advantage of. Specifically, preconditions of processes are not recorded due to a lack of uniformity. However, once settled on a standard, the addition of preconditions will be beneficial for the accuracy of the system in order for it to produce more robust results.
In terms of retrieving appropriate cases, the case base's performance could suffer when it holds a substantial amount of cases. Means of building an appropriate index based on specific features, may have to be introduced to accelerate the retrieval process at a certain size to allow realistic performance.

### 6.2.2 Evaluation

It has been demonstrated that basic principles of Case-based Reasoning can be applied to Web Services described in OWL-S. Due to today's lack of wide availability of various Web Services described in OWL-S, an extensive evaluation of the specific similarity measurements and adaption strategies has not been performed. As briefly described in Chapter 5, a suitable case base needs to be built and systematically tested against new cases to be able to make significant statements about its performance.

### 6.2.3 Similarity Measures

In addition to the implemented similarity measurements, alternative ways should be explored. Once a substantial test case base has been developed, an overall ontology could be built following the approach of OWL-MX to perform a combination of semantic and syntactic matching based on different thresholds. SimPack also provides a number of additional string based similarity measurements, that have not been explored in the current implementation. Undergoing a thorough evaluation, these additional measurements may prove superior to the currently chosen ones under certain circumstances, or may act complementary. Additionally, graph-based measurement can be enhanced to respect not only structural similarity, but also consider specific labels for each resource. This also implies that not only recorded Web Services should serve as a basis for the case base, but the original version of the Web Service.

### 6.2.4 Adaption Strategies

Additional complexity can be added to the currently implemented strategies. Specifically, parts of different cases should be able to contribute to more sophisticated combinations instead of limiting the adaption process to a single case. Furthermore, more complex rules can be added to cover a wider variation of adaption. Another interesting approach is to use such concepts found in AI planners to perform parts of the adaption process. Such systems should be highly suitable to bridge the gap between two incompatible resources on various levels.

### 6.2.5 Revision and Retention

An integral part of a Case-based Reasoner should be the revision and retention of cases. In the context of NExT, user feedback should determine the quality and usability of the produced result by the Case-based Reasoner. Depending on the outcome, such newly generated cases should be added to the existing repository of cases to serve in the future. This allows the system to evolve over time and improve its competence, coverage and reachability. However, to not over specialize the system, it should also undergo regular maintenance in which cases may be generalized and duplicates removed.

### 6.2.6 Future Applications

Today's reality reflects a sparse application of the Semantic Web in the vast space of the Internet. Certainly the Internet itself is considered a young and emerging technology, which has already impacted many different aspects of our everyday lives. In the past, many regarded the Semantic Web as the next version of the Net. Reality proved to be slower, and today Web Applications using techniques such as AJAX are considered to be the Web 2.0. As the Internet evolves to Web 3.0, the Semantic Web should be an integral part of it. In such an environment, automated agents will be able to interact seamlessly with human agents and almost contribute equally. Applications will be exposed and made accessible over the Internet by either humans or automatic agents. An application, similar to the one introduced in this thesis, can advertise its Case-based Reasoning capabilities as a service to help solve new problems based on existing Web Services advertised by other parties.

# Index of Figures

# Index of Tables

# 7 References


[Baggenstos 2006] Daniel Baggenstos, Implementation and Evaluation of Graph Isomorphism Algorithms for RDF-Graphs, University of Zurich, 2006

[Bareiss 1989] R. Bareiss, Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning, Academic Press, San Diego, CA, 1989

[Bernstein 2005] So what is a (Diploma) Thesis? A few thoughts for first-timers. (by A. Bernstein), 2005

[Colucci et al. 2004] S. Colucci, T. D. Noia, E. D. Sciascio, F. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. In Proc. 6th Int Conference on Electronic Commerce (ICEC 2004). ACM Press, 2004.

[Daenzer 2005] Michael Daenzer, NExT – The NMR EXperiment Toolbox, University of Zurich, 2005

[Estlin et al. 2000] T. Estlin, G. Rabideau, D. Mutz, S. Chien. Using Continuous Planning Techniques to Coordinate Multiple Rovers, Electronic Transactions on Artificial Intelligence, 2000

[Hammond 1989] K. J. Hammond, Case-Based Planning, Academic Press, San Diego, CA, 1989

[Hinrihs, 1992] T. R. Hinrihs, Problem Solving in Open Worlds, Lawrence Erlbaum Associates, Hillsdale, NJ, 1992

[Li and Horrock, 2003] L. Li and I. Horrock. A software framework for matchmaking based on semantic web technology. In Proc. 12th Int World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW 2003), 2003.

[Keller et al. 2005] U. Keller, R. Lara, H. Lausen, A. Polleres, D. Fensel. Automatic Location of Services. In Proceedings of the 2nd European Semantic Web Conference, LNCS 3532, 2005.

[Klusch et al. 2006] Matthias Klusch, Benedikt Fries, Katia Sycara: Automated Semantic Web Service Discovery with OWLS-MX, International Conference on Autonomous Agents, Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, 2006

[Kolodner 1983] J. L. Kolodner, Maintaining organization in a dynamic long term memory, Cognitive Science, vol. 7, 1983

[Kolodner 1992] J. L. Kolodner, An introduction to Case-based Reasoning,

Artificial Intelligence Review, vol. 6, no. 1, pp. 3-34, 1992

[Kolodner and Mark 1992] J. L. Kolodner, W. Mark, Case-based Reasoning, IEEE Expert, vol. 7, no. 5, pp. 5-6, 1992

[Kolodner 1993] J. L. Kolodner, Case-Based Reasoning, Morgan Kaufmann, San Francisco, 1993

[Martin et al. 2004] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/1.1/overview/ - last time validated on February, 1, 2007.

[Massimo et al. 2002] Massimo Paolucci, Takahiro Kawamura, Terry Payne, Katia Sycara: Semantic Matching of Web Services Capabilities. Springer Berlin / Heidelberg, 2002

[Sankar and Simon 2004] Sankar K. Pal, Simon C. K. Shiu , Foundations of Soft Case-Based Reasoning, Wiley Series on Intelligent Systems, ISBN: 978-0-471-08635-2, April 2004

[Simpson 1985] R. L. Simpson, A computer model of Case-based Reasoning in problem solving: an investigation in the domain of dispute mediation, Ph.D. Dissertation, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, 1985

[Sycara 1988] K. Sycara, Using Case-based Reasoning for plan adaption and repair, in Proceedings of the Case-Based Reasoning Workshop, DARPA, Clearwater Beach, FL, Morgan Kaufmann, San Francisco, 1988

[Sycara et al. 2003] K. Sycara, M. Paolucci, A. Anolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. Web Semantics, Elsevier, 2003.

# A Appendix – Case-based Reasoning

## *A.A Case-based Reasoning Criteria*

Although the methodology of Case-based Reasoning can be applied to a number of domains, it is not always suitable. The following characteristics should be met in order to determine whether it is applicable:[31]

1. Does the domain have an underlying model?
   If the domain is impossible to understand completely or if the factors leading to the success of failure of a solution can't be modeled explicitly, CBR allows us to work on pas experience without a complete understanding of the underlying mechanism.
2. Are there exceptions and novel cases?
   Domains without novel or exceptional cases may be modeled better with rules, which could be determined inductively from past data. However, in a situation where new experiences and exceptions are encountered frequently, it would be difficult to maintain consistency among the rules in the system. In that case the incremental case learning characteristics of CBR systems makes it a possible alternative to rule-based systems.
3. Do cases recur?
   If the experience of a case is not likely to be used for a new problem, because of a lack of similarity, there is little value in storing the case. In other words, when experiences are not similar enough to be compared and adapted successfully, it might be better to build a model of the domain to derive the solution.
4. Is there significant benefit in adapting the pas solutions?
   One should consider whether there is a significant benefit, in therms of resources, to crating a solution through modifying a similar solution rather than creating a solution to a problem from scratch.
5. Are relevant previous cases obtainable?
   Is it possible to obtain data that record the necessary characteristics of pas cases? Do the recoded cases contain the relevant features of the problem and its context that influenced the outcome of the solution? Is the solution recoded in sufficient detail to allow it to be adapted in the future? These questions allow one to go for the CBR framework.

31 See [Kolodner 1992], [Kolodner and Mark 1992], [Kolodner 1993]

## *A.B Advantages of Case-based Reasoning*

Following a summary of some of the major advantages of CBR:[32]

1. Reducing the knowledge acquisition task
   By eliminating the need to extract of a model or a set of rules, as is necessary in model/rule-based systems, the knowledge acquisition tasks of CBR consist primarily of the collection of relevant existing experiences/cases and their representation and storage.
2. Avoiding repeating mistakes made in the past
   In systems that record failures as well as successes, and perhaps the reason for those failures, information about what caused failures in the past can be used to predict potential failures in the future.
3. Providing flexibility in knowledge modeling
   Due to their rigidity in problem formulation and modeling, model-based systems sometimes can't solve a problem that is on the boundary of their knowledge or scope or when there is missing or incomplete data. In contrast, case-based systems use past experience as the domain knowledge and can often provide a reasonable solution, through appropriate adaption, to these types of problems.
4. Reasoning in domains that have not been fully understood, defined, or modeled
   In a situation where insufficient knowledge exists to build a causal model of a domain or to derive a set of heuristics for it, a Case-based Reasoner can still be developed using only a small set of cases from the domain. The underlying theory of domain knowledge does not have to be quantified ore understood entirely for a Case-based Reasoner to function.
5. Making predictions for the probable success of a proffered solution
   When information is stored regarding the level of success of past solutions, the Case-based Reasoner may be able to predict the success of the solution suggested for a current problem. This is done by referring to the stored solutions, the level of success of these solutions and the differences between the previous and current contexts of applying these solutions.
6. Learning over time
   As CBR systems are sued, they encounter more problem situations and create more solutions. If solution cases are tested subsequently in the real world and a level of success is determined for those solutions, these cases can be added to the case base and used to help in solving future problems. As cases are added, a CBR system should be able to reason in a wider variety of situations and with a higher degree of refinement and success.
7. Reasoning in a domain with a small body of knowledge
   While in a problem domain for which only a few cases are available, a Case-based Reasoner can start with these few known cases and build its knowledge incrementally as cases are added. The addition of new cases will cause the system to expand in directions that are determined by the cases encounters in its problem-solving endeavors.
8. Reasoning with incomplete or imprecise data concepts
   As cases are retrieved, they many not be identical to the current case. Nevertheless, when they are within some defined measure of similarity to the

32 See [Sankar and Simon 2004], pp. 9-11

present case, any incompleteness and imprecision can be dealt with by a Case-based Reasoner. Although these factors may cause a slight degradation in performance, due to the increase disparity between the current and retrieved cases, reasoning can continue.

9. Avoiding repeating all the steps that need to be taken to arrive at a solution
   In problem domains that require significant processes to create a solution from scratch, the alternative approach of modifying an earlier solution can reduce this processing requirement significantly. In addition, reusing a previous solution also allows the actual steps taken to reach that solution to be reused for solving other problems.
10. Providing a means of explanation
    Case-based Reasoning systems can supply a previous case and its solution to help convince a suer of, or to justify, a proposed solution to the current problem. In most domains there will be occasions when a user wishes to be reassured about the quality of the solution provided bu a system. By explaining how a previous case was successful in a situation, using the similarities between the cases and the reasoning involved in adaption, a CBR system can explain its solution to a user. Even for a hybrid system, one that may be using multiple methods to find a solution, this proposed explanation mechanism can augment the causal explanation given to a user.
11. Extending to many different purposes
    The number of ways in which a CBR system can be implemented is almost unlimited. It can be used for many purposes, such as creating a plan, making a diagnosis, and arguing a point of view. Therefore, the data dealt with by a CBR system are able to take many forms, and the retrieval and adaption methods will also vary. Whenever stored past cases are being retrieved and adapted, Case-based Reasoning is said to be taking place.
12. Extending to a broad range of domains
    CBR can also be applied to extremely diverse application domains. This is due to the seemingly limitless number of ways of representing, indexing, retrieving and adapting cases.
13. Reflecting human reasoning
    As there are many situations where we, as humans, use a form of Case-based Reasoning, it is not difficult to convince implementers, users, and managers of the validity of the paradigm. Similarly, humans can understand a CBR system's reasoning and explanations and are able to be convinced of the validity of the solutions they receive from a system. If a human user is wary of the validity of an earlier solution, they are less likely to use this solution. The more critical the domain, the lower the chance that a past solution will be used and the greater the required level of a suer's understanding and credulity.

# B Appendix - OWL-S

Based on the progress made in the Semantic Web effort, the DARPA Agent Markup Language (DAML) Program developed a set of ontologies in the ontology language OWL to describe Web services. OWL-S is therefore “a OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form.”[33] The goal is to facilitate the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation.

OWL-S therefore defines a top level ontology Service to provide the data needed to discover and invoke, but also to allow composition and inter operation of Web Services. OWL-S associates three basic entities to a Service as shown in the following figure[34]:

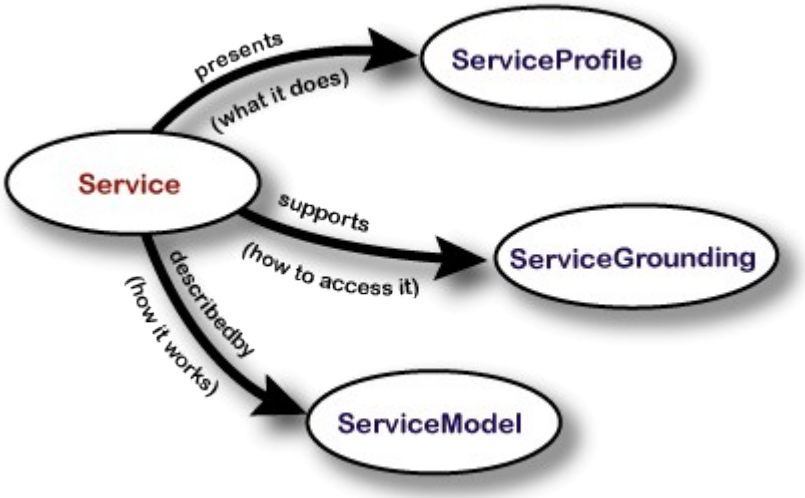


*Figure 24: Top level of the service ontology*

The service profile tells "what the service does", in a way that is suitable for a service-seeking agent (or matchmaking agent acting on behalf of a service-seeking agent) to determine whether the service meets its needs. This form of representation includes a description of what is accomplished by the service, limitations on service applicability and quality of service, and requirements that the service requester must satisfy to use the service successfully.

The service model tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. That is, it describes how to ask for the service and what happens when the service is carried out. For nontrivial services (those composed of several steps over time), this description may be used by a service-seeking agent in at least four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from

33 http://www.daml.org/services/owl-s/
34 Figure from white paper about OWL-S - http://www.daml.org/services/owl-s/1.1/overview/

multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate the activities of the different participants; and (4) to monitor the execution of the service.

A service grounding ("grounding" for short) specifies the details of how an agent can access a service. Typically a grounding will specify a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each semantic type of input or output specified in the ServiceModel, an unambiguous way of exchanging data elements of that type with the service (that is, the serialization techniques employed).

The upper ontology for services specifies only two cardinality constraints: a service can be described by at most one service model, and a grounding must be associated with exactly one service. The upper ontology deliberately does not specify any minimum cardinality for the properties presents or describedBy. (Although, in principle, a service needs all three properties to be fully characterized, it is easy to imagine situations in which a partial characterization could be useful.) Nor does the upper ontology specify any maximum cardinality for presents or supports. (It will be extremely useful for some services to offer multiple profiles and/or multiple groundings.)

Finally, it must be noted that while there is one particular upper ontology for profiles, one for service models, and one for groundings, nevertheless OWL-S allows for the construction of alternative approaches in each case.

# C Tools and Libraries

## C.A Eclipse

Eclipse 3.1 was used as an IDE to write Java code. To be able to collaborate and store the work properly, Subversive[35] was used. Subversive is an Eclipse Team Provider for the Subversion version control system. Google Code was used as a Subversion repository. The code is available in its entirety under http://code.google.com/p/cbr-owl-s/. UML 2.1 by Omondo[36] was used to generate the UML diagrams.

## C.B Mindswap OWL-S API

"OWL-S API provides a Java API for programmatic access to read, execute and write OWL-S (formerly known as DAML-S) service descriptions. The API supports to read different versions of OWL-S (OWL-S 1.1, OWL-S 1.0, OWL-S 0.9, DAML-S 0.7) descriptions. The API provides an ExecutionEngine that can invoke AtomicProcesses that has WSDL or UPnP groundings, and CompositeProcecesses that uses control constructs Sequence, Unordered, Split, If-Then-else and RepeatUntil."[37]

The OWL-S API has the ability to execute OWL-S web services that are grounded in WSDL or UPnP. As of recently, the API has been extended by Michael Daenzer to support Java Groundings. However, the extensibility of the OWL-S architecture allows future implementations of theoretically any technology.

## C.C Simpack

The Department of Informatics at the University of Zurich has developed a similarity measurement toolkit called SimPack[38]. Until now, it supports the following measurement approaches:

Currently, similarity measures from the following categories have been implemented:

Feature vectors
- Alignment, Cosine, Dice, Euclidean, Jaccard, Manhattan, Overlap, Pearson

Strings or sequences of strings (text)
- Averaged String Matching, Jaro, TFIDF

Sets
- Jaccard, Loss of Information, Resembalance

35 http://www.polarion.org/index.php?page=overview&project=subversive
36 http://www.omondo.com/
37 See http://www.mindswap.org/2004/owl-s/api/
38 http://www.ifi.unizh.ch/ddis/simpack.html

Sequences
- Levensthein Edit Distance

Trees
- Bottom-up/Top-down Maximum Common Subtree, Tree Edit Distance

Graphs
- Conceptual Similarity, Graph Isomorphism, Subgraph Isomorphism, Maximum Common Subgraph Isomorphism, Graph Isomorphism Covering, Shortest Path

Information theory
- Jiang & Conrath, Lin, Resnik

In addition, the package implements the measures from the SecondString[39], the SimMetrics[40], and the OWLS-MX[41] projects.

39 http://secondstring.sourceforge.net/
40 http://www.dcs.shef.ac.uk/~sam/simmetrics.html
41 http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/index.html

# Glossary

Case - A case is a past experience that actually happened. In this context, we refer to OWL-S ontologies that have been executed using the OWL-S API and recorded in an OWL-S ontology itself.

Execution Trail – Synonym for Case

Case Base - The case base is a set of cases.

Knowledge Base – Synonym for Case Base.

CBR – Case-based Reasoning

NMR - Nuclear Magnetic Resonance

DL – Description Logic

GUI – Graphical User Interface

XML – Extensible Markup Language

OWL – Ontology Web Language

OWL-S – OWL Web Service Ontology

Pellet – OWL-DL reasoner

I/O – Input/Output

IOP – Inputs, Outputs, Processes

AI – Artificial Intelligence

UDDI – Universal description, discovery and integration