Personalised Service Discovery and Composition based on Conversational Case-Based Reasoning

Author: Charlie Abela
Supervisor: Dr. Matthew Montebello
External: Dr. Kalyan Moy Gupta

Department of Computer Science & AI
University of Malta
September 2006

Submitted in fulfilment of the requirements for the degree of Master of Science
Abstract

The proliferation of Web services is fostering the need for applications to provide more personalisation during the service discovery and composition phases. An application has to cater for different types of users and seamlessly provide suitably understandable and refined replies.

In this thesis, we describe the motivating details behind PreDiCtS, a framework for personalised service discovery and composition. The underlying concept behind PreDiCtS is that, similar service composition problems could be tackled in a similar manner by reusing past composition best-practices. These have to be useful and at the same time flexible enough to allow for adaptations to new problems. For this reason we are opting to use template-based composition information.

PreDiCtS’ template-retrieval and refinement technique is based on conversational case-based reasoning (CCBR) and makes use of a core OWL ontology called CCBROnto for case representations. This mixed-initiative approach provides both a process through which the requester can express more clearly his service-related needs, and also an effective way in which templates with relevant service-composition knowledge can be reused.

The template retrieval process uses different CCBR algorithms that handle both the case similarity and the conversation-generation. The retrieved templates are personalised through an adaptation process and eventually handed over to a partial order planner (POP) planner that has the task of discovering and binding services that comply with the template. The bound services are then executed through an execution engine.

We test and evaluate PreDiCtS’ behaviour through the use of two different domains, the medical and travelling domains, for which we provide a set of related ontologies. We discuss the attained results and eventually discuss future extensions to the concepts presented in this work.
Dedication

This thesis is dedicated to
my wife, Doris,
for her continuous support, tolerance and unselfish love,
and to
my children, Cherise Ann, Andrew and Matthew,
for their patience, understanding and constant encouragement.
Acknowledgements

As with all dissertations, many people helped make this project possible, both through their endless support, help and courage. I apologize in advance to those who are overlooked here.

First of all I would like to thank my supervisor and friend, Dr. Matthew Montebello, for his continuous encouragement, support and guidance.

I would also want to thank Mr. Mike Rosner, head of the Computer Science and AI department, and Dr. Chris Staff, for always finding time to discuss issues related to this work.

During this work I have benefited from numerous discussions with authors whose papers I have cited in this thesis. Every effort has been made to obtain permission for the use of copyrighted materials or to cite them in a unified professional way. I hope that my apologies will be accepted for any errors or omissions. I would like to thank these authors and in particular Dr. Nicola Henze for her time to discuss issues related to personalisation and similarity distance and Dr. Kalyan Gupta, for finding the time to reply to my emails in relation to Taxonomic CCBR. His clarifications were always very helpful.

Last but not least I would like to thank the people contributing to the mailing lists about Web services, Semantic Web and Jena for their support in providing information and clarifications.
# Contents

Abstract ................................................................. i
Dedication .............................................................. ii
Acknowledgements .................................................. iii
Contents ................................................................. iv
List of Figures .......................................................... viii
List of Tables ........................................................... x
List of Acronyms ....................................................... xi

## Introduction
1.1 Problem Overview ................................................. 2
1.2 Solution Overview ................................................ 3
1.3 Thesis Structure ................................................... 6

## Web Service Description and Composition Languages
2.1 Introduction ......................................................... 7
2.2 Initiatives at W3C ................................................... 8
2.3 OWL-S .............................................................. 10
2.4 WS-BPEL ........................................................ 11
2.5 WSMO ............................................................. 12
2.6 WSDL-S .......................................................... 13
2.7 Comparison ........................................................ 15
2.8 Summary ........................................................... 18

## Web Service Composition Techniques
3.1 Introduction ......................................................... 20
3.2 Composition using AI Planning ................................ 21
3.2.1 STRIPS Planning ............................................. 22
3.2.2 HTN Planning ................................................ 25
3.2.3 POP Planning ................................................ 27
3.2.4 Conclusion ..................................................... 31
3.3 Abstract Templates ................................................ 32
3.3.1 Abstract Processes in Meteor-S .......................... 32
3.3.2 Templates in OWL-S ....................................... 33
3.3.3 Abstract Workflows in Grid Environments .......... 34
3.4 Summary ........................................................... 35

## Methodologies Employed: Background and Motivations
4.1 Introduction ........................................................ 37
4.2 Case-Based Reasoning .......................................... 38
7.4.3  ContextPanel and ContextBuilder Classes ......................................... 101
7.4.4  ProblemPanel and ProblemBuilder Classes......................................... 102
7.4.5  VisualComposer Class ......................................................................... 103
7.5  Case Retriever Package .............................................................................
    7.5.1  Overview .......................................................................................... 107
    7.5.2  CaseRetrieverTask ........................................................................... 110
    7.5.3  Default CCBR Methodology ............................................................ 112
    7.5.4  Taxonomic CCBR Methodology ....................................................... 113
7.6  Maintenance Package ............................................................................... 117
7.7  Summary ................................................................................................... 118

Testing and Evaluation .............................................................................. 119
8.1  Testing PreDiCtS Retrieval System ............................................................ 119
    8.1.1  Default Similarity Measure ............................................................... 120
    8.1.2  Comparison between the CCBR Similarity Measures ....................... 125
8.2  Suitability of the PreDiCtS Solution .......................................................... 132
    8.2.1  QAPair design ................................................................................. 132
    8.2.2  Solution Representation ................................................................... 133
8.3  Summary ................................................................................................... 134

Future Work and Conclusion ..................................................................... 135
9.1  Adaptation mechanism ........................................................................... 135
9.2  Integration phase ..................................................................................... 136
9.3  Feedback Mechanism ............................................................................. 137
9.4  Context Awareness ................................................................................ 138
9.5  Graph-Based Technique ......................................................................... 139
9.6  Conclusion ............................................................................................... 139

Appendix A .................................................................................................. 142
1.  Introduction ............................................................................................... 142
2.  Related Work ............................................................................................ 143
3.  CCBROnto ................................................................................................. 143
4.  Context ..................................................................................................... 143
5.  Problem ................................................................................................... 144
6.  Solution .................................................................................................... 145
7.  Conclusion ............................................................................................... 145
References .................................................................................................... 146

Appendix B .................................................................................................. 147
1.  Introduction ............................................................................................... 147
2.  Conversational Case-Based Reasoning .................................................... 149
3.  CCBROnto ................................................................................................. 149
4.  PreDiCtS: implementation issues ............................................................... 151
    4.1  Case Creation ...................................................................................... 152
    4.2  Case Retrieval .................................................................................... 154
5.  Conclusion ............................................................................................... 155
References .................................................................................................... 155

Appendix C .................................................................................................. 157
CCBROnto Ontology ..................................................................................... 157
Medical Domain ........................................................................................ 164
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travelling Domain</td>
<td>166</td>
</tr>
<tr>
<td>Complete Case Definition Example</td>
<td>168</td>
</tr>
<tr>
<td>QAPairs for the Travelling Domain</td>
<td>172</td>
</tr>
<tr>
<td>Configuration file</td>
<td>175</td>
</tr>
<tr>
<td>References</td>
<td>177</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Blocks World</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>STRIPS Regressive Algorithm (taken from Weld [77])</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Methods to accomplish some tasks (taken from Nau [47])</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>Plan Space (taken from Weld [77])</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>Regressive version of POP (taken from Weld [77])</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>Case-based reasoning (adapted from Leake [39])</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>CBR cycle (adapted from Weber [73])</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>CCBR cycle (adapted from Weber [73])</td>
<td>42</td>
</tr>
<tr>
<td>9</td>
<td>CCBR cycle in PreDiCtS</td>
<td>48</td>
</tr>
<tr>
<td>10</td>
<td>CCBR taxonomy</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>PreDiCtS structure</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>Question-Answer Pair Scoring</td>
<td>53</td>
</tr>
<tr>
<td>14</td>
<td>CBRONTo case structure (taken from Diaz-Agudo [19])</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>Example of case definition based on CBRONTo, Diaz-Agudo [19]</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>Case Definition</td>
<td>62</td>
</tr>
<tr>
<td>17</td>
<td>CCBRONTont Ontology Context structure</td>
<td>63</td>
</tr>
<tr>
<td>18</td>
<td>CCBRONTont Problem structure</td>
<td>64</td>
</tr>
<tr>
<td>19</td>
<td>QAPair example</td>
<td>65</td>
</tr>
<tr>
<td>20</td>
<td>CCBRONTont solution structure</td>
<td>66</td>
</tr>
<tr>
<td>21</td>
<td>Travelling Domain Ontology</td>
<td>68</td>
</tr>
<tr>
<td>22</td>
<td>Travelling QAPairs Taxonomy</td>
<td>70</td>
</tr>
<tr>
<td>23</td>
<td>Travelling Domain Template Representation</td>
<td>71</td>
</tr>
<tr>
<td>24</td>
<td>Part of the Medical Domain Ontology</td>
<td>72</td>
</tr>
<tr>
<td>25</td>
<td>Creating a new case for the Medical Domain</td>
<td>74</td>
</tr>
<tr>
<td>26</td>
<td>System Component Summary</td>
<td>77</td>
</tr>
<tr>
<td>27</td>
<td>Main PreDiCtS use cases</td>
<td>78</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1</td>
<td>QAPair/Concept relations</td>
<td>51</td>
</tr>
<tr>
<td>TABLE 2</td>
<td>Travelling Problem Question/Triple association</td>
<td>69</td>
</tr>
<tr>
<td>TABLE 3</td>
<td>Sample for Medical Problem Question/Triple association</td>
<td>73</td>
</tr>
<tr>
<td>TABLE 4</td>
<td>QAPairs for the Medical domain</td>
<td>120</td>
</tr>
<tr>
<td>TABLE 5</td>
<td>Cases for Medical Domain</td>
<td>122</td>
</tr>
<tr>
<td>TABLE 6</td>
<td>QAPairs for the Travelling Problem</td>
<td>125</td>
</tr>
<tr>
<td>TABLE 7</td>
<td>Cases related to the Travelling domain</td>
<td>126</td>
</tr>
</tbody>
</table>
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBR</td>
<td>Case-Based Reasoning</td>
</tr>
<tr>
<td>CBROnto</td>
<td>Case-Based Reasoning Ontology</td>
</tr>
<tr>
<td>CCBR</td>
<td>Conversational Case-Based Reasoning</td>
</tr>
<tr>
<td>CCBROnto</td>
<td>Conversational Case-Based Reasoning Ontology</td>
</tr>
<tr>
<td>HTN</td>
<td>Hierarchical Task Networks</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>OWL-S</td>
<td>Web Ontology Language for Services</td>
</tr>
<tr>
<td>PDDL</td>
<td>Planning Description Definition Language</td>
</tr>
<tr>
<td>POP</td>
<td>Partial Order Planning</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Language</td>
</tr>
<tr>
<td>SESMA</td>
<td>Semantic Service Markup</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOC</td>
<td>Service Oriented Computing</td>
</tr>
<tr>
<td>STRIPS</td>
<td>STanford Research Institute Problem Solver</td>
</tr>
<tr>
<td>SWSF</td>
<td>Semantic Web Services Framework</td>
</tr>
<tr>
<td>SWSI</td>
<td>Semantic Web Services Initiative</td>
</tr>
<tr>
<td>SWSL</td>
<td>Semantic Web Services Language</td>
</tr>
<tr>
<td>SWSO</td>
<td>Semantic Web Services Ontology</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>VHPOP</td>
<td>Versatile Heuristic Partial Order Planner</td>
</tr>
<tr>
<td>WS-BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>WS-CDL</td>
<td>Web Service Choreography Description Language</td>
</tr>
<tr>
<td>WSCI</td>
<td>Web Service Choreography Interface</td>
</tr>
<tr>
<td>WSCL</td>
<td>Web Service Conversation Language</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSDL-S</td>
<td>Web Services Semantics</td>
</tr>
<tr>
<td>WSMF</td>
<td>Web Services Modelling Framework</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Services Modelling Ontology</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

With ever developing globalization markets, today's business organizations operate their business in a globalized economy. In order to be more competitive in such dynamic and economic environments, business organizations need to streamline their business processes. With the proliferation of the Internet and the wide acceptance of e-commerce, an increasingly number of Web services is being offered. At the same time, the discovery and composition of Web services has gained a lot of momentum as a means to support business process automation. Our research is motivated by the need to facilitate and personalise the discovery and composition processes through reusable Web service compositions.

This chapter is organized as follows. In Section 1.1, we outline the research issues that we propose to tackle in this thesis. In Section 1.2, we summarize our intended solutions and in Section 1.3 we present the structure of this thesis.
1.1 Problem Overview

Reusability and interoperability are at the core of the Web services paradigm. This technology promises seamlessly interoperable and reusable Web components that facilitate rapid application development and integration. Nonetheless, the technologies behind Web services (such as WSDL [83] and UDDI [71]), lack the semantics necessary for machines to be able to autonomously make effective use of this technology. A solution to this lack of semantics is the main goal behind the Semantic Web. The aim is to bring structure to the existent Web through semantic annotations defined through languages such as RDF [56] and OWL [48]. This will make it possible for machines and humans to collaborate in a more effective manner. We find an analogous situation in research on Semantic Web services, where a number of languages which include OWL-S [49] and WSMO [85] are being developed such that software agents can use the semantic annotations to deal in a more efficient manner with Web service discovery, composition, invocation and execution.

When referring to composition, this is usually interpreted as the integration of a number of services into a new workflow or process. A number of compositional techniques have been researched ranging from both, manual and semi-automatic solutions through the use of graphical authoring tools, discussed in Sirin [61], Scicluna [59], to automated solutions based on techniques such as AI planning, used in Peer [54], Sirin [62] and others.

The problem with most of the composition techniques mentioned above is three fold:

(i) such approaches attempt to address service composition by composing Web services from scratch, ignoring reuse or adaptation of existing compositions or parts of compositions,

(ii) it is assumed that the requester knows exactly what he wants and how to obtain it and

(iii) composing Web services by means of concrete service interfaces leads to tightly-coupled compositions in which each service involved in the chain is tied to a Web service instance.

If service reuse is based on (iii), this may lead to changes in the underlying workflow which range from slight modifications of the bindings to whole re-designing of parts of the workflow description. Therefore in our opinion, services should be interpreted at an
abstract level to facilitate their independent composition. Goderis [27] adds, “abstract workflows capture a layer of process description that abstracts away from the task and behaviour of concrete workflows”, and this allows for more generalisation and a higher level of reusability. A system can start by considering such abstractly defined workflow knowledge and work towards a concrete binding with actual services that satisfy the workflow.

The reuse of abstract workflows brings with it a set of other issues, which include the way that these workflows are generated, stored and retrieved. We therefore have to provide answers to the following questions:

- Who will create these abstract workflows?
- How will the system identify which is the best workflow to present to the user?
- Will the system allow the user to adapt (thus personalise) an abstract workflow so that this becomes more suitable to his needs?

In the next section we will present an overview of the solution we want to adopt that will provide answers for these questions. Our solution will be based on the following motivating points:

1. Reusability of compositions has the advantage of not starting from scratch whenever a new functionality is required.
2. For effective reusability, a higher level of abstraction has to be considered, which generalises service concepts and is not bound to specific service instances.
3. Personalisation of compositions can be achieved by first identifying more clearly the user’s needs and then allowing for reuse and adaptation of these past compositions based on these needs.
4. Compositions can be bound with actual services thus making them concrete.

1.2 Solution Overview

In our approach we wanted to tackle the problems highlighted above and thus our main goal is to put the user (developer or otherwise) in a situation whereby he can reuse existing past experience. In fact this approach is similar to that adopted in Rajasekaran [55], Sirin [63], Deelman [15], and Weber [74], which use pre-stored abstract workflow definitions or templates in their composition framework. However, in these approaches it is assumed that the service templates are created by developers needing a more
flexible composition. In PreDiCtS though, these templates contain service-composition knowledge which is recommended by domain experts and specifically intended for reuse.

To make this reuse process effective we initially considered CBR, which is amenable for storing, reusing and adapting past experience for current problems. Nevertheless CBR restricts the user to define a complete problem definition at the start of the case-retrieval process. Therefore a mixed-initiative technique such as conversational case-based reasoning (CCBR), Aha [7], was considered as being more appropriate since it allows for a partial definition of the problem by the user, and makes use of a refinement process to identify more clearly the user’s problem state.

In summary the goal of this work is to present, the motivation behind, and prototype of PreDiCtS, a framework which allows for personalisation of service discovery and composition through the reuse of past composition knowledge. One could say that we are trying to encode and store common practices of compositions which could then be retrieved, reused and adapted through a personalisation technique. The solution we propose in PreDiCtS has two phases, though in this work we will mainly focus on the first phase. Nevertheless, we will discuss several aspects of the second phase, where required.

For the first phase, which we call the *Similarity phase*, we have adopted a mixed-initiative technique to provide a suggestion-refinement process which is based on CCBR. This provides for the personalisation process. Given a new problem (a request for a particular service functionality), the CCBR approach provides a set of questions which the user can choose to answer. Depending on these answers, a ranked list of past, similar cases are retrieved and recommended to the requester, together with other unanswered questions. Through this dialogue process the requester can decide when to stop this iterative-filtering phase, and when to reuse or adapt a chosen case.

To be able to define CCBR cases we have developed an OWL-based ontology, which we call CCBROnto (CCBR Ontology). Each case is divided into three main components, namely the *CaseContext*, *Problem* and *Solution*.

The *CaseContext* represents knowledge related to the case creator, case history, ranking value and case provenance. In this regard, we have considered context as defined by Dey [16] and also ideas presented in Bry [11] and Maamar [41] which
discuss the importance of context in relation to Web services. In PreDiCtS, context knowledge helps to identify why a case was created and by whom, certain aspects of case usage and the case relevance to problem-solving.

The Problem description in a PreDiCtS case is a list of question-answer pairs (QAPairs) rather than a bag since these have to be ranked when they are presented to the user.

In CCBR a Solution represents an action that can be used to solve the requester’s problem. Such actions in PreDiCtS encapsulate a process model description (or a service composition template). Each template represents a workflow of generic, unbound service components. We use the OWL-S process model as the basis for these templates in this work, though, we envision that other workflow models could be utilised as well.

The case-retrieval process in PreDiCtS is handled through a number of different similarity measures. One of these measures is based on the similarity measures defined in Aha [7] and used by Weber [73] to handle workflow reuse. Another similarity measure is based on the taxonomic theory defined in Gupta [28]. Through this technique similarity considers the abstract relations between QAPairs and in particular the sub-class relation.

Once a case is refined it can be presented to the next phase of PreDiCtS, which we call the Integration phase. During this phase, a mapping is attempted, from the features in the solution of the chosen, and possibly, adapted case (this we term as the Most-Similar Case or MSC), to actual services that are found in a service registry. Whenever the Similarity phase does not return a suitable MSC the user can create a new case from scratch. Building new cases is not a trivial task, since it requires detailed knowledge about the domain. Therefore such task is best left for knowledge engineers. We considered this situation and it is envisioned that in the future PreDiCtS will allow users to add new cases by communicating with other peers across a network.

The solution in the MSC is used by an AI planning component to query a service registry and to combine retrieved services according to this information. The planning algorithm could be based on the partial order planning algorithm (POP) (defined in Weld [77]) and as in Peer [54].
1.3 Thesis Structure

The rest of this thesis is structured as follows. In Chapter 2 we discuss and compare issues related to the most important Web service composition languages initiatives. We describe research related to different composition techniques in Chapter 3. In Chapter 4 we provide further motivating issues behind reuse and sharing of knowledge by providing background on techniques such as CBR and CCBR with the main focus being on case reusability and similarity. Then in Chapter 5 we will describe the design and implementation of CCBROnto, an OWL ontology which is used to define cases for our prototype. In Chapters 6 and 7 we will present the specification, design, and implementation details of the PreDiCtS prototype. We present evaluation details in Chapter 8, which is followed by the future work and conclusion sections in Chapter 9.
Chapter 2

Web Service Description and Composition Languages

2.1 Introduction

The emphasis in Application Development today has shifted from tightly coupled, monolithic, proprietary software to loosely coupled, dynamically bound service-based systems, comprising of distributed components provided by more than one vendor. The current Web service model, as defined by IBM [32], enables service discovery dynamically through the use of markup languages that describe service properties. However it does not account, for automatic integration of one service with another. Extensive work has been done in the area of service discovery and matchmaking. However, the dynamics of service composition still remains one of the most challenging aspects for researchers in academia and industry.

The minimally accepted framework for Service Oriented Computing (SOC) is the Service Oriented Architecture (SOA) [64], which consists of the following basic components: (i) A service provider, or owner of the service, (ii) a service requester, or client that searches for services to invoke depending on his required needs and (iii) the
service directory, which is a repository of services that have been advertised by service providers. When a provider wants to make his service public, he publishes its interface (which is a description of the inputs, outputs, message types and operations) with one or more service directories. A requester may consult any one of these directories to find a service that provides the necessary functionality to suite his needs and binds to a specific service provider so that service invocation can occur.

In the present scenario, configurations for Web services revolve around three main XML-based open standard technologies namely, Web Services Description Language (WSDL) [83], Simple Object Access Protocol (SOAP) [65] and Universal Description Discovery and Integration (UDDI) [71], which are respectively the description language for services, the protocol that is used to support interactions between services and the distributed repository where services are published. However these standards are still evolving and may be subject to revisions and extensions based on new requirements and possibly even changes or refinements to the vision of the SOC paradigm.

In addition to the Web service standards mentioned above a plethora of languages for modelling and specifying different facets of SOC have been proposed. In what follows we will focus our discussion on some of these languages and how descriptions expressed in these languages could be used as templates.

2.2 Initiatives at W3C

WSDL is an XML format for describing both abstract Web service functionalities and concrete Web service bindings. A WSDL description describes the abstract Web service interface through which a service consumer communicates with a service provider, as well as the specific details of how a given Web service has implemented that interface. WSDL is described as a Component Model consisting of WSDL components like messages (typed data elements) and operations (a set of input and output messages), port types (a set of operations), bindings and services, and type system components. Port types are reusable and can be bound to multiple ports. A binding component describes a binding of a port-type component and associated operations with a particular concrete message format and transmission protocol. WSDL lacks the semantics to express message exchange and ordering. It also does not support any operations for monitoring services, such as checking the availability of an operation or the status of a
submitted request. Recently, a first working draft of WSDL Version 2.0 has been published [84]. Some of the most important changes concern (i) the definition of message patterns for the exchange of messages. These can be seen as a first step towards the association of semantics to exchange between messages, and (ii) the support of inheritance (but not of overloading) between operations. It seems very likely that other constructs will be added in future working drafts, possibly those supporting composition.

Like WSDL, the Web Service Conversation Language (WSCL) [81] is another ongoing initiative at W3C. It models the conversations supported by a Web service by specifying the XML documents that are exchanged together with the allowed sequences, similar to an activity diagram representation. A WSCL specification includes four main elements: (i) document type descriptions specify the types (schemas) of XML documents the service can accept during a conversation, (ii) interactions that model the actions of the conversation as document exchanges between two participants. (iii) transitions specify the ordering relationships between interaction and (iv) conversations that list all the interactions and transitions that make up the conversation.

Evolving from WSCL we find the Web Service Choreography Interface, WSCI [80] and the Web Service Choreography Description Language, WS-CDL [79]. The latter is the latest in a series of evolutions of the choreography initiative coming from W3C. This specification describes collaborations between any type of party, with no concern about the supporting platform or programming model used on the hosting environment. Using the WS-CDL specification, a contract containing a "global" definition of the common ordering conditions and constraints under which messages are exchanged, is produced. This describes the common and complementary observable behaviour of all the parties involved from a global viewpoint. Each party can then use this global definition to build and test conforming solutions. The global specification is in turn realized by a combination of the resulting local systems, on the basis of appropriate infrastructure support. In order to facilitate collaborations through WS-CDL, services commit to mutual responsibilities by establishing relationships. Their collaboration takes place in a jointly agreed set of ordering and constraint rules, whereby information is exchanged between the parties.
2.3 OWL-S

The OWL-S coalition [49] defines a set of ontologies and a Semantic Web Services language called OWL-S (formerly called DAML-S). The ontology enables the description of service related concepts and complex relations between them. The OWL-S ontology set comprises:

- **Service Profile**: describes the properties of services necessary for automatic discovery. This includes both functional properties such as, inputs, outputs, preconditions and effects, and non-functional properties, such as name of the service, contact information, quality of the service, and additional information that may help to evaluate the service. The profile is intended to automate the service discovery process.

- **Service Model**: describes the services’ process model, i.e. control and data-flow. It presents three types of service descriptions; Atomic or primitive service, composite or complex service (made up of atomic and possibly other composite services) and simple which is an abstract representation of either an atomic or a composite service. The service model is intended to provide for autonomous service composition and execution.

- **Service Grounding**: focuses on how to access a service in terms of the communication protocol, marshalling and serialisation. It also connects the process model description to the communication protocols and message descriptions defined in WSDL.

The OWL-S ontologies are still evolving (latest version is 1.2) and some parts, such as the execution monitoring in the process model are still not defined. Recently though the consortium involved in the Semantic Web Services Initiative, SWSI [68] have launched an initial draft of the Semantic Web Services Framework (SWSF) [67] which presents some interesting work that builds over OWL-S and WSMO. In particular we are referring to the newly proposed Semantic Web Services Language, SWSL [69] and relevant Semantic Web Services Ontology, SWSO [70]. The First-order Logic Ontology for Web Services, FLOWS, is the first-order logic (FOL) axiomatization of SWSO and defines its model-theoretic semantics. The goal of FLOWS is to enable reasoning about the semantics underlying the Web Services paradigm, how they interact with each other and with the real world. It does not strive for a complete representation of Web services,
but rather for an abstract model that is faithful to the semantic aspects of service behaviour.

2.4 WS-BPEL

Web Services Business Process Execution Language or WS-BPEL (formerly BPEL4WS) [78] is the standard language, under the Organization for the Advancement of Structured Information Standards (OASIS), for specifying orchestration and coordination of services. It is an XML-based language and is intended to support the modelling of both executable and abstract processes. An abstract process is a business protocol that specifies the message exchange behaviour between different parties without revealing their internal behaviour. An executable process specifies the execution order between a number of activities that constitute the process, the partners involved in the process, the messages exchanged between these partners, and the fault and exception handling that specify the behaviour to adopt in cases of errors and exceptions. The BPEL process is a kind of flow-chart, where each element in the process is called an activity. An activity can be either primitive or structured. The set of primitive activities contains: *invoke*, which is used to invoke an operation on some Web service; *receive*, that is used to wait for a message from an external source; *reply*, which is used when replying to an external source; *wait*, when it is necessary to wait for some time; *assign*, for copying data from one place to another; *throw*, to indicate errors in the execution; *terminate*, when terminating the entire service instance; and *empty*, when the process is doing nothing. Several structured activities are defined to enable the presentation of complex structures. These are: *sequence*, which is used to define an execution order; *switch*, used for conditional routing; *while*, used for looping; *pick*, for race conditions based on timing or external triggers; *flow*, which is used for parallel routing; *scope*, for grouping activities to be treated by the same fault-handler; and *compensate*, which is used to undo the effects of already completed activities.

Structured activities can be nested and combined in arbitrary ways. Within activities that are executed in parallel the use of *links* can further control the execution order. These are sometimes also called *control links* and allow the definition of directed graphs. The graphs can be nested but must be acyclic.
WS-BPEL makes use of the complementary specifications, WS-Coordination [82] and WS-Transaction [86]. These are jointly developed by IBM and Microsoft. They provide the coordination between the dependable outcomes of both short and long running business activities. The WS-Transaction specifies a framework that allows a composed Web service to monitor the success or failure of each individual, coordinated activity. Through this specification a service can reliably cancel the monitored process if something goes wrong during execution. The WS-Coordination specification on the other hand, defines a framework which allows the developer to define the shared “coordination context” within a composed service. This context contains the information necessary to link the various activities within the service together.

2.5 WSMO

The Web Service Modelling Ontology (WSMO) [85] is a research effort conducted through a joint initiative of European researchers and is focused around Semantic Web services. Similar to OWL-S, WSMO aims at the development of an overall framework for Semantic Web services that supports automated Web service discovery, composition, and execution. WSMO defines four top-level notions related to Semantic Web services.

For management purposes, every WSMO component is described by non-functional properties based on the Dublin Core Metadata Set defined by Weibel [76]. This is defined as a generic description model for information items. In accordance to the Web Service Modelling Framework WSMF, defined in Fensel [25], WSMO applies two major design principles:

1. **the principle of maximal de-coupling**: that is, all WSMO components are specified autonomously, independent of connection or interoperability with other components.

2. **the principle of strong mediation**: stating that the connection and interplay between different components is realized by Mediators that resolve possible occurring heterogeneities between the connected components.

In its current version, the WSMO Standard specifies the following description elements and components:
1. **Ontologies**: these are the key to link conceptual real-world semantics defined and agreed upon by communities of users. They define a common agreed-upon terminology by providing concepts and relationships among the set of concepts, together with a set of axioms.

2. **Goals**: describe the user’s requests that should be resolved by invoking a service. These goals are expressed as logical expressions and reference domain ontologies. Post-conditions and effects are used to define when a goal is solved.

3. **Web Services**: are described by a Capability (or functional description), and by an Interface that describes both its choreography and orchestration. The Capability is similar to the Service Profile in OWL-S. It defines (i) the input requested by the service in order to fulfil its functionality and pre-conditions, (ii) assumptions, which are arbitrary constraints on the world that have to hold before the Web service can be executed, (iii) the output of the service in relation to the input and post-condition, and (iv) effects which are conditions on the state of the world that result from the execution of the Web service. The Web service Interface notion serves the same purpose as the Service Model within OWL-S, but is more elaborate with regards to the conceptual model and the specification languages used.

4. **Mediators**: are the components for realizing the underlying principles of strong decoupling and mediation. Whenever WSMO components are to be connected, a Mediator is used to connect them. This mediation facilitates interoperability by resolving any occurring heterogeneities. Four types of Mediators are defined: OO Mediators which connect ontologies and import them as terminology definitions into other components, GG Mediators for connecting Goals, WG Mediators connect Goals and Web services, and WW Mediators which connect Web services.

### 2.6 WSDL-S

Another important initiative is called Web Service Semantics or WSDL-S [75] and is based on WSDL and its extensibility feature. WSDL 2.0 provides two extensibility mechanisms: an open content model, which allows XML elements and attributes from other (non-WSDL 2.0) XML namespaces to be used in a WSDL document; and
Features and Properties. Both mechanisms use URIs to identify the semantics of the extensions.

Through this extensibility feature, WSDL-S defines a mechanism to associate semantic annotations with Web services that are described using WSDL. This approach is different from those mentioned previously, as stated in Web Service Semantics [75], since:

(i) it allows users to describe, in an upwardly compatible way, both the semantics and operation level details in WSDL
(ii) it takes an agnostic approach to ontology representation languages, by externalizing the semantic domain models. This allows Web service developers to annotate their Web services with the ontology language of their choice (such as UML or OWL).
(iii) it is based on the WSDL specification, it is relatively easy to update existing tooling to accommodate this approach.

Thus WSDL-S provides a mechanism to annotate the capabilities and requirements of Web services with semantic concepts referenced from a semantic model. This is done through the mechanisms that allow annotation of the service and its inputs, outputs and operations, together with mechanisms to specify and annotate preconditions and effects of Web services. These preconditions and effects together with the semantic annotations of inputs and outputs can enable automation of the service discovery process.

WSDL-S provides five extension elements and attributes which provide hooks for attaching semantics to WSDL components. The five elements and attributes are described below:

- `modelReference` attribute; whose URI value points to a concept in the semantic model of the Web service and specifies that there is a one-to-one mapping between the owner of the attribute and the referenced concept. This attribute can be used by itself on an XML Schema element or type declaration to link directly to an ontology concept, or it can be used as part of specifying a precondition or effect.
- `schemaMapping` attribute, whose URI value points to a mapping between a schema and ontology concepts. This attribute is to be used on an XML Schema element or type declaration and can point to a transformation, for example in
XSLT, from XML data that follows this schema to ontological data, for example in RDF/OWL or in WSML.

- *precondition* and *effect* are similarly structured elements that are used on WSDL interface operations to specify conditions that must hold before and after the operation is invoked. The conditions are specified either by reference to the semantic model (reusing the *modelReference* attribute) or directly by writing the condition as the value of the *expression* attribute of the two elements.

- *category* element provides a pointer to some taxonomy category. It can be used on a WSDL interface and is intended to be used either in taxonomy-based discovery or when the WSDL interface is stored in a repository.

### 2.7 Comparison

In the previous section we have identified some of the most important initiatives related to service description and composition languages. Most initiatives are somewhat related to the WSDL standard. Nevertheless, through a WSDL description we cannot unambiguously determine what the service does. The syntax of inputs and outputs lacks meaning and does not define what changes to the environment the service makes. We do not know the meaning of the parameters nor the terms referenced in payload documents. In fact, two services can have the same syntactic definition but perform significantly different functions. Similarly, two syntactically dissimilar services can perform the same function.

In this regard OWL-S, WSMO and WSDL-S present solutions for this problem by adding semantics to the basic service descriptions, though their approaches are different. On the other hand, BPEL is considered to be a choreography language and it does not add any semantic annotations or external ontology associations.

While OWL-S has a rich and flexible semantic expressivity (and most of the other initiatives are based on this language) it suffers from some important limitations, such as those listed in, Web Service Semantics [75], (which compares OWL-S with WSDL-S):

- it is not aligned with the existing Web services standards. For example, while the grounding model in OWL-S uses WSDL bindings, the OWL-S profile model duplicates the descriptions embodied in the rest of WSDL.
it assumes that everyone uses OWL for representing ontologies which may not always be the case.

In the DIP deliverable [20] some other comparisons between OWL-S and WSMO are highlighted. In particular it states that:

(i) OWL-S does not differentiate between the service request and the service profile definitions. That is, a user has to define a profile (the more complete it is, the better is the matchmaking with concrete services) and present this as part of his service request. WSMO on the other hand, decouples these perspectives by allowing the requester to define goals and the provider to describe service functionalities as Web services.

(ii) While WSMO only allows one capability description per service, an OWL-S description allows one service to be associated with zero, one or more profiles. A WSMO Web service on the other hand, could be linked to many goals via the \textit{WGMediators}. This difference stems mainly from the fact that OWL-S enables classification-based discovery (discovery based on the classification of the Web services' profiles), while WSMO aims at logically-enhanced discovery.

(iii) OWL-S service profiles can be arranged in a hierarchy and linked to an existing taxonomy thus allowing category-based discovery. Mediators can provide such categorization based on existing taxonomies of predefined and/or generic goals. An advantage in WSMO is that these taxonomies can be entirely heterogeneous, since unlike OWL-S, in which each hierarchy is based on a predefined shared ontology of profiles, each goal or capability can use its own domain dictionary. Differences within concepts from these taxonomies can then be resolved with the help of \textit{OOMediators}.

(iv) OWL-S service profiles have associated non-functional properties that present contact information, service title and a human language description of service functionality, as well as service category. In WSMO such information is based on Dublin Core.

(v) The inputs, outputs, preconditions and effects (IOPEs) represent the functional properties in OWL-S descriptions. These are defined in the process model but they are also referenced from the profile. WSMO services
on the other hand, transform information through preconditions and post-conditions. These specify conditions over the inputs and outputs of the service. In WSMO goals, only the definition of post-conditions is reasonably considered, since the user is only interested in the results achieved by the service. In general the user is not aware what the expected inputs to a service are.

(vi) The state of the world in OWL-S is described through preconditions and effects, while in WSMO this is defined through assumptions and effects. OWL-S effects are based on conditions which are specified as expressions in a logical language, such as SWRL or KIF. On the other hand WSMO assumptions describe the necessary conditions for service execution and are not related to the input, while effects describe the state of the world after service execution.

(vii) Each WSMO service can be associated with multiple interfaces, while OWL-S allows a single Service model for each service. An OWL-S process presents an underspecified list of its participants while WSMO does not support the notion of participants at all. An Atomic process in OWL-S can be mapped to a WSMO service that does not have orchestration knowledge, while a Composite service is represented by a full-fledged WSMO service.

(viii) OWL-S and WSMO have similar approaches to the groundings of Semantic Web services. The OWL-S grounding specifies how the type of each input or output defined in the Service Model can be serialized as an XML message. It also specifies which WSDL operation corresponds to each atomic process in this model. Similarly the WSMO Grounding maps the data represented in WSMO ontologies to XML and the behaviour of service descriptions to WSDL.

As explained in section 2.5, the semantic model of a Web service in WSDL-S, is expected to contain the semantics of input and output parameters and the specifications of preconditions and effects of service operations, plus the categorization of a WSDL interface. When compared to the WSMO specification in, DIP deliverable [20], the latter provides a more detailed model through capabilities with preconditions,
assumptions, post-conditions and effects; an interface with choreography and orchestration where data is described using ontologies.

WSDL-S adds semantics to inputs and outputs by adding annotations (which link elements and types to ontology concepts) in the XML Schema used in the WSDL document. Similarly in WSMO, schema elements are linked with ontology concepts, by specifying links between choreography input and output concepts and WSDL messages.

The non-functional properties Subject and Type in a WSMO Web service can be compared to the categorization of a WSDL interface in WSDL-S. Therefore it can be assumed that a WSDL-S model covers WSMO well.

WSMO is different from WS DL-S in that it distinguishes between preconditions, assumptions, post-conditions and effects, which are modelled on the whole Web service rather than splitting it into operations. Additionally, WSMO models both the choreography and orchestration interface/s of a Web service, while this aspect is not covered by WSDL-S at all.

In the case of WS-BPEL, this is used to generate choreographies since they provide explicit and structured information about the business processes making use of a WSDL description. In WSMO on the other hand, the focus is on how to conceptually relate the business process descriptions to choreographies. A mapping between the WSMO choreography and orchestration models and WS-BPEL is suggested in, DIP deliverable [20], since there are direct correspondences between most of WS-BPEL workflow activities and ASM (Abstract State Machines) rules. Similar mappings or overlaps between BPEL and OWL-S have been found by Aslam [8], which could be used to overcome the semantic limitations of the former language. On the other hand WS-BPEL seems to be more mature then both OWL-S and WSMO, when it comes to fault handling and compensation.

2.8 Summary

In this chapter we presented a number of service description and composition languages. The basic language in each case was WSDL, but since this lacks semantics, most of the mentioned initiatives add semantic annotations so that a machine would more easily be able to understand the mechanism and functionality of the defined services. Though the approaches and level of maturity of these languages are different,
overlappings between them exist and it might be possible, in the near future to combine service descriptions defined in any of these languages into a singleton.
Chapter 3

Web Service Composition Techniques

In recent years the area of Web service composition has been researched in depth and a number of techniques ranging from AI planning to program synthesis, situation calculus, finite state machines, to name a few, have been used. Though we cannot state that any one of these prevails over the others, the work on AI planning seems to be quite suitable since Web services have been considered in Mithun [44] and Sirin [62], as synonymous to Actions in planning. Nevertheless this poses some problems when applied directly to Web service composition.

3.1 Introduction

A number of AI planning algorithms have been used to handle Web service composition Peer [54], Sirin [62], Mithun [44]. Nevertheless plan generation as described for classical planning in Russel [58], assumes a closed world and requires apriori knowledge about the planning domain. This poses a number of problems if we apply it directly to Web service composition:

1. neither the initial nor the goal states are fully known before composition starts
2. There is only partial knowledge about the services that might be involved in the composition, some might not be suitable or are not available at the time of planning.

A proposed solution involves the use of pre-canned or pre-fabricated plans. In this regard, Aberg [5], Deelman [15], Mulye [45], Sirin [63] and others refer to these pre-canned plans as workflow or composition templates that represent best practices. These (abstract) templates include generic knowledge about service compositions that can be specialised upon request. This specialisation process binds the abstract definitions with actual services.

In what follows we first present some of the initiatives that are based on AI planning, where we give background knowledge about each technique and highlight achievements and shortcomings vis-à-vis service composition. We then focus on the solution to service composition based on the use of abstract templates.

### 3.2 Composition using AI Planning

A planning problem has been described by Weld [77] as requiring the following inputs:

1. A description of the world or start state
2. A goal state
3. A set of Actions

The goal of the planning process is to identify a sequence for the set of Actions that if executed properly will lead the planner from the initial state to a state that satisfies the goal conditions.

This definition is comparable with the service composition problem, in which the goal is to create an ordering over a set of services that if executed properly, will solve the requester’s initial problem by supplying the correct information or making the correct changes to the world. The latter may refer to the situation whereby a particular action is taken as a result of the execution of the service, for example, if we consider a BookBuyingService, this may translate to the effect that the number of books in the store are decreased by one, or that a book has been packaged and will be sent to the buyer.

As a result of this similarity a number of AI planning algorithms have been used to tackle this service composition problem. In the next sections we will look in detail at some of these algorithms and their contribution to solving this problem.
3.2.1 STRIPS Planning

STRIPS stand for (STanford Research Institute Problem Solver) and it represents world models as a collection of first order predicate calculus formulas. The planner’s task is therefore that of “finding a sequence of operators in the space of world models that will transform an initial model into another model in which a given formula (goal) can be proven to be true”, Fikes [26].

The initial state of the world in a STRIPS’ representation is completely described through a set of ground literals (that is literals that are true in the world). A typical world example is the blocks world which consists of a table and three blocks, shown in Figure 1 below.

A list of true literals for this world could be:

(on A Table) (on C A) (on B Table) (clear B) (clear C)

The classical planning theory requires that the initial state description is complete prior to the start of the planning process. Therefore all atomic formulae not explicitly listed in the description are assumed to be false, as per the closed world assumption (CWA). This means that the negative literals (not(on A C)) and (not (clear A)), to list a few, have to be implicitly assumed in the initial state description.

The goal state in STRIPS is restricted to those matching a conjunction of positive literals. For example the final state for the problem shown in Figure 1 is represented by the literals, (on A B), (on B C) and (on C Table).

The third component of the planning problem is the domain theory which consists of a set of Actions. These Actions are represented with preconditions and effects. The
former are defined through a conjunction of positive literals while the latter could be a conjunction of both positive and negative literals. For example the Action `moveC(A Table)` may be defined as follows:

**Precondition:** \((\text{and} \ (\text{on} \ C \ A) \ (\text{clear} \ C))\)

**Effect:** \((\text{and} \ (\text{on} \ C \ \text{Table}) \ (\text{not} \ (\text{on} \ C \ A)) \ (\text{clear} \ A))\)

When an Action is executed it changes the world description. In STRIPS this change causes the conjunction of positive literals in the effect to be added to the present state description while all the negative literals are removed.

Weld [77] describes both a progressive and a regressive algorithm based on STRIPS. Here we look briefly at the regressive version (see Figure 2), which starts from the goal conjunction of literals and does a backward search. During each step it chooses an Action that satisfies one of the outstanding goals.

**Algorithm:** \(\text{REGWS}(\text{init-state}, \text{cur-goals}, \Lambda, \text{path})\)

1. If \(\text{init-state}\) satisfies each conjunct in \(\text{cur-goals}\),
2. Then return \(\text{path}\),
3. Else do:
   (a) Let \(\text{Act} = \text{choose}\) from \(\Lambda\) an action whose effect matches at least one conjunct in \(\text{cur-goals}\).
   (b) Let \(G = \) the result of regressing \(\text{cur-goals}\) through \(\text{Act}\).
   (c) If no choice for \(\text{Act}\) was possible or \(G\) is undefined, or \(G \ni \text{cur-goals}\),
   (d) Then return failure,
   (e) Else return \(\text{REGWS}(\text{init-state}, G, \Lambda, \text{Concatenate}(\text{Act}, \text{path}))\).

**Figure 2:** STRIPS Regressive Algorithm (taken from Weld [77])

The algorithm above is taken directly from Weld [77] and \(\text{init-state}\) is the initial state, \(\text{cur-goals}\) is the list of pending goals, \(\Lambda\) represents the set of Actions available, while \(\text{path}\) represents a path through the state-space that provides a solution. The \(\text{choose}\) function (based on a searching technique) mentioned here, nondeterministically chooses an Action that is added to the path that strives to find the initial state.

In the next section we will revisit this planning algorithm and discuss how it has been applied to solve the service composition problem.
STRIPS for Service Composition

In his thesis Mithun [44] the author describes how a regressive STRIPS’ based algorithm was used to handle service composition and invocation. The architecture includes both a planner and a service invocation engine, together with an agent that is able to interact with these components to request a service and to invoke it.

The described planner is based on Java and uses JESS (Java Expert System) [35] and Jena [34] to extract service knowledge from OWL-S service descriptions and manipulate these to produce planning operators.

A list of planning operators is synthesised from the service descriptions and implemented as Java classes. The IOPEs (Inputs, Outputs, Preconditions and Effects) of each service are represented by fields in these classes. The planning process is then triggered by the user who specifies a set of goals that he wants the system to achieve. The planner uses a backward chaining algorithm which takes these goals and tries to find services that satisfy these goals. Whenever such a service is found, its inputs and preconditions are treated as intermediate unsatisfactory goals and added to the goals list. A new searching iteration is then initiated. Planning terminates either when the planner fails to find a suitable plan or else when all the goals are satisfied.

The successful plan is then passed on to the invocation engine which fetches an operator in a forward chaining manner and uses these to generate a client stub based on Apache Axis that contains methods that handle the service invocation. During this invocation process the engine takes an operator from the plan, checks whether all preconditions hold and then searches the list of available inputs (these are specified by the user) to match with those required by the operator. Two types of queries are used for this searching process, an ExQuery and a PartQuery. These provide for different levels of input matches.

Summary

This work provides a simple but effective solution to the composition problem, though it relies on a well defined list of user goals and a finite list of services. The author does not elaborate on how these goals are obtained, and does not elaborate on how partial-goal definition is accounted for. It seems that service discovery is considered as being
totally separate from the composition process, and the retrieved services are assumed to be atomic and undecomposable. Nevertheless it is interesting how the invocation process is automatically executed and how an allowance for service execution failure is handled.

3.2.2 HTN Planning

HTN stands for Hierarchical Task Networks, and planning is viewed as the problem of decomposing a high-level activity description into a number of concrete sub-activities. This refining process is implemented through a set of action-decompositions, where each one reduces a higher-level action into a partially-ordered set of lower level actions. The action-decompositions therefore, contain the knowledge about how to implement actions. The process continues until only primitive actions remain in the plan, where primitive actions are considered as single-step actions that require no further decomposition, Russel [58].

In Erol [24] the authors analyse the complexity of HTNs. In particular they conclude that:

1. HTNs are more expressive than STRIPS-like operators (that is operators with preconditions, add and delete lists)
2. even though HTNs are undecidable, decidability can be obtained by adding some restrictions on either the non-primitive tasks or on task ordering.

The basic HTN algorithm requires the following inputs; an initial task network, a set of operators and a set of methods. The initial task network consists of the tasks that have to be carried out. These are classified as either primitive (explained above) or non-primitive, that is, these require further decomposition. Task networks may also include constraints over the tasks such as ordering information, which requires that a task has to be performed before or after another. Each operator includes a description of what has to be done to achieve a particular primitive task. These are usually based on the classical STRIPS definition and include a list of preconditions and another list of effects (which was described to include add and delete lists as above). The methods, which are usually predefined and stored in a library, define how the non-primitive tasks can be decomposed into a partially ordered set of primitive ones (see Figure 3).
The planning process starts from the initial task network and repeatedly finds a non-primitive task that can be decomposed through some available method. Each time a task is decomposed the task is replaced in the original list of tasks by its subtasks. Once no non-primitive tasks are left, a total-ordering that satisfies all the constraints, for the primitive tasks is sought. If this is found then the plan is considered as successful for the initial problem.

![Figure 3: Methods to accomplish some tasks (taken from Nau [47])](image)

In the next section we will look at how a particular planner, called SHOP2, which is based on the HTN algorithm, has been used to solve the composition of OWL-S services.

**HTN for Service Composition**

SHOP2, as described by Nau [47], is an HTN planner that generates planning steps in the same order that these will later be executed. This allows the planner to continuously know its current state during the planning process and reduces complexity by eliminating uncertainty in the world.

In Sirin [62], this planner was used to handle the composition of services described in OWL-S. The idea was that, given a set of user objectives and a set of Web services, the planner had to find a set of service requests that achieved these objectives. A direct
mapping is shown to exist between the HTN’s operators and methods, and OWL-S descriptions (some restrictions were applied).

In fact given a set of OWL-S Web service descriptions, SHOP2 operators were generated from OWL-S atomic services, while methods were generated from simple and composite services. After this translation into the SHOP2 domain, the planner was presented with an initial task for which a plan was searched. Plans could be translated back to OWL-S through a SHOP2 to OWL-S converter. These descriptions were then passed on to a Web service execution engine which invoked the services and monitored their execution.

**Summary**

We have seen that the HTN algorithm is very suitable to handle the service composition problem. This is mainly due to its capability of decomposing complex tasks into simpler ones and then again combines these into plans that solve a particular problem. Nevertheless we have to note that this paper does not make allowances for service discovery in the classical sense (through a matchmaker), or to how the user’s request is identified more clearly. Rather it assumes that a number of services are already available for the planner to use. Thus discovery here can be considered as searching for the operators and methods during planning, though it is not clear how an efficient search is carried out when a large number of operators and methods are available.

**3.2.3 POP Planning**

The STRIPS planning discussed earlier was considered as a search through the space of world states which resulted in a path between the initial state and goal state. The planning algorithm that we will present here is a search in the plan space. This planning algorithm is called Partial Order Planning or (POP), (defined in Weld [77]) and the nodes in the search space are considered as partially-refined plans, see Figure 4. Thus the initial state is represented by what is called the null plan while the goal state is a plan that represents the solution.

This algorithm is based on the principle of least commitment, in which plans are represented as a partially-ordered sequence of actions and only the most important ordering decisions are considered.
A plan in POP is considered to be a tuple made up of $<\mathcal{A} \ O \ \mathcal{L}>$, where $\mathcal{A}$ represents a set of actions, $O$ represents a set of ordering constraints over the actions and $\mathcal{L}$ represents a set of causal links. The ordering constraints define the order in which actions have to be executed within a plan. During planning, actions may be considered that somehow may interfere with others that are already committed to the plan. These interfering actions are called threats. A threat is described as an action which although consistent with the set of ordering constraints, presents a negative effect.

Thus for example, if a partial plan contains the action $A_1 = (\text{on } A \ B)$ which is ordered to execute before $A_2$ and a new action $A_3$ is considered, the latter is said to be a threat iff, it moves $A$ off $B$ but at the same time does not conflict with the ordering constraints and can be executed between $A_1$ and $A_2$. To handle these threats the planner uses causal links which is a structure that contains a pointer to two consecutive actions, the first called a producer and the other consumer, and a proposition which is both an effect of the first action and a precondition of the second action. Whenever a threat is encountered the planner has to handle it either through demotion (i.e. execute before the producer) or through promotion (execute after the consumer).

We discuss briefly the regressive version of the POP algorithm as shown in Figure 5 (taken from Weld [77]). The algorithm starts with the null plan and an agenda which contains a conjunction of goals that have to be achieved. POP makes nondeterministic choices (between actions) until all conjuncts of every action’s precondition are
supported by causal links and all the threatened links have been protected from possible interferences. If a total order is consistent with the ordering constraints then this guarantees that the sequence of actions in the plan will solve the problem.

Algorithm: \( \text{POP}(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, \text{agenda}, \Lambda) \)

1. **Termination:** If agenda is empty, return \( \langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle \).

2. **Goal selection:** Let \( \langle Q, A_{\text{need}} \rangle \) be a pair on the agenda (by definition \( A_{\text{need}} \in \mathcal{A} \) and \( Q \) is a conjunct of the precondition of \( A_{\text{need}} \)).

3. **Action selection:** Let \( A_{\text{add}} = \text{choose} \) an action that adds \( Q \) (either a newly instantiated action from \( \Lambda \), or an action already in \( \mathcal{A} \) which can be consistently ordered prior to \( A_{\text{need}} \)). If no such action exists then return failure. Let \( \mathcal{L}' = \mathcal{L} \cup \{ A_{\text{add}} \rightarrow A_{\text{need}} \} \), and let \( \mathcal{O}' = \mathcal{O} \cup \{ A_{\text{add}} < A_{\text{need}} \} \). If \( A_{\text{add}} \) is newly instantiated, then \( \mathcal{A}' = \mathcal{A} \cup \{ A_{\text{add}} \} \) and \( \mathcal{O}' = \mathcal{O} \cup \{ A_0 < A_{\text{add}} < A_{\text{inf}} \} \) (otherwise let \( \mathcal{A}' = \mathcal{A} \)).

4. **Update goal set:** Let \( \text{agenda}' = \text{agenda} - \{ \langle Q, A_{\text{need}} \rangle \} \). If \( A_{\text{add}} \) is newly instantiated, then for each conjunct, \( Q_i \), of its precondition add \( \langle Q_i, A_{\text{add}} \rangle \) to \( \text{agenda}' \).

5. **Causal link protection:** For every action \( A_t \) that might threaten a causal link \( A_p \rightarrow A_c \in \mathcal{L}' \) choose a consistent ordering constraint, either

   (a) **Demotion:** Add \( A_t < A_p \) to \( \mathcal{O}' \), or

   (b) **Promotion:** Add \( A_c < A_t \) to \( \mathcal{O}' \).

   If neither constraint is consistent, then return failure.

6. **Recursive invocation:** \( \text{POP}(\langle \mathcal{A}', \mathcal{O}', \mathcal{L}' \rangle, \text{agenda}', \Lambda) \).

Figure 5: Regressive version of POP (taken from Weld [77])

This algorithm has been especially successful when planning in nondeterministic domains and a number of extended versions have evolved in time. UCPOP, defined in Weld [77] and VHPOP, defined in Younes [87] to name a few, both extend the basic POP algorithm. In the next section we will show how the latter, VHPOP, has been used to handle the service composition problem.
VHPOP for Service Composition

VHPOP (Versatile Heuristic Partial Order Planner) is used in Peer [54] to deal with service composition. This planner was embedded within an execution monitoring engine and it was possible to deal with both nondeterminism in the domain and the arising contingencies during planning. The service description language used is called SESMA, which is described in detail in Peer [53]. In brief, a SESMA service is defined by a quadruple $<\text{URI}, \text{Prec}, \text{Res}, \text{Var}>$ where:

1. $\text{URI}$: is a reference to the operation’s (or service) unique identifier
2. $\text{Prec}$: is a set of formulas that describe the conditions that have to be satisfied before the operation executes
3. $\text{Res}$: is the set of possible results of the operation; these may include effects and knowledge effects
4. $\text{Var}$: refers to a set of variables used in the formulas, whereby input variables are distinguished from output variables.

Having said that, the other details describing evaluations of preconditions and success conditions, the idea of conversation sets (i.e. data tokens exchanged between client and services) are not dealt with here, since it is out of scope of this review section. Although it is worth mentioning that SESMA goals are described as logical statements about the world. Goals allow variables which are treated with existential quantification. This is different from OWL-S, since there is in fact no explicit way to define goals in OWL-S.

In VHPOP, a partially ordered plan is represented by a quadruple $<S, O, B, L>$ where $S$ is a set of plan steps (or operation instances), $O$ is a set of ordering constraints, $B$ is a set of binding constraints and $L$ is a set of causal links. VHPOP uses a planning graph and a number of search heuristics to guide the planning process.

The planning problem is defined by a set of goals, an initial state, a set of SESMA annotated services and a threshold value that represents the number of planning cycles that the planner is allowed before it terminates the process. The process starts by translating both the Web service domain and the Web service composition problem into a Planning Domain Definition Language (PDDL) [52] description. This is fed to the VHPOP planner which tries to find a suitable plan. In this implementation three types of plans are allowed, (i) complete, (ii) partially ordered and (iii) plans that do not contain
avoidable causal links (causal links that violate certain restrictions). These avoidable links are used as feedback for the next planning cycle, should a plan not be found on the first pass.

Given a plan, each step is tested to learn whether or not this achieves the desired goal. The actual service execution is triggered by the creation of input messages that are sent to the service. Output messages or results, from these services are parsed and checked so that any violation is handled. The plan is successfully completed if all the operations are executed with no violation detection.

**Summary**

In this section we described the POP algorithm and how an extension, called VHPOP, was used to handle service composition. The basic algorithm is based on three main components that include a list of actions, a list of ordering constraints and a set of causal links. In the VHPOP implementation these features were fundamental when verifying whether an action fits into the plan without causing any interference with already committed actions. Both the set of services considered in a planning cycle and the goals were assumed to be available and clearly defined in SESMA.

**3.2.4 Conclusion**

In the previous section we looked into some detail at three major planning algorithms and their implementations to handle service composition. The results were quite promising though some crucial assumptions that were made, were found to be common in all three cases. These were:

1. the services taking part in the planning process are already available
2. the user’s goals were clearly defined.

In reality this information is usually never completely known at the time of composition. In fact it is realistic to presume that a user starts his request for a particular functionality without the exact knowledge of which functionality is exactly out there, possibly even with partial knowledge of what he really needs to use to solve his problem. These are issues that we have mentioned earlier and for which we provide a solution later on in this research.
3.3 Abstract Templates

In the section related to HTN planning discussed earlier, the planner used a set of pre-stored methods that were used to decompose complex tasks into less complex ones, until only primitive tasks were available. The idea behind storing these methods is synonymous to the use of canned plans or templates and has been investigated in various research works such as in Deelman [15], Sirin [63], Aggarwal [6] and Mithun [44]. The advantage of such principle is that the planner does not have to start the planning process from scratch and thus provides for more reusability. Nevertheless these templates have to encapsulate generic knowledge that can be specialised as required.

This is also in line with, Goderis [27], which discusses the benefits of using such abstract or generic templates in the e-Science domain. Other domains were templates are useful include B2B applications where business rules may be encoded using templates. Though most of the steps in a business process are static, the choice of partners during a transaction may be done at runtime depending on the specific requests. Similarly in Grid computing (where resources are distributed over multiple networks and have to be collated to solve large-scale computing problems) workflows may be used to represent the standard procedures in which a task is accomplished, though the actual service or resources that will be used for a specific task may vary with time and therefore will need to be replaced or the workflow adapted to suit the new situation.

In the following sections we will consider research related to the use of such templates in service composition, with the scope of identifying how we can define (our own) and utilise them as part of our solution.

3.3.1 Abstract Processes in Meteor-S

Meteor (Managing End-To-End OpeRations) is focused on the management of transactional workflows. Meteor-S is an extension to this tool and is specifically focused on workflow management in Web services.

Aggarwal [6] makes reference to this framework but focuses mainly on the backend which consists of an abstract process designer, a discovery engine, a constraint analyser and a service binder. The abstract process designer is used to create templates, referred to as Service Templates (ST), in WS-BPEL. The discovery engine uses these templates...
and formulates queries to a service registry in which it tries to find suitable services that
match the template. It also handles any transformations required to make a service
advertisement match with the template. The constraint analyser filters services from the
list returned by the discovery engine by choosing those that satisfy all the service
selection constraints. Thus the approach is based on constraint satisfaction. The binder
is used to bind the abstract process defined in an ST to actual services after which an
executable WS-BPEL process is generated and executed through BPWS4J (a java
implementation of the WS-BPEL execution environment) [10].

The abstract processes generated by the designer involve three main tasks:

1. creation of the process flow through the control-flow constructs provided by
WS-BPEL
2. create service templates for the services required in the flow; the designer can
either bind these to existing services or else specify a Semantic Web service
description
3. add process constraints required for optimisation

The abstract process flow may include STs for specific services that can be bound to
actual services during execution time. In these STs the designer can specify the service
requirements by defining a set of operations with their inputs and outputs. These
operations together with the input/output messages are annotated by concepts defined in
different ontologies. It is the job of the discovery engine to use these templates together
with the abstract WS-BPEL definitions to find services that can be bound with these
templates.

3.3.2 Templates in OWL-S

In Sirin [63], the authors provide the motivation behind the use of workflow templates
and discuss how these can be created using OWL-S. The idea behind templates is
discussed through an example where a conference organiser provides a workflow
template that describes the necessary steps for making travelling arrangements to attend
a conference. Though the template may contain information about how to register for
the conference and how to arrange for transportation and make hotel reservation, it may
not specify the actual travelling agent or hotel booking services.
A workflow template is defined as a specification of the activities (and their ordering within the workflow) that need to be done in order for a task to be completed. It should represent a generalisation of the workflow by allowing activities to be represented by abstract definitions. These abstract activities can later on be bound to actual services during the discovery process. The template also provides for the specification of user preferences. These can be used during the binding process to further restrict the results returned during service discovery.

To be able to define these templates, OWL-S is extended with a new construct called AbstractProcess, which is added to the existing AtomicProcess, CompositeProcess and SimpleProcess service constructs (please refer to the OWL-S specification [49]). The idea is that partially-abstract templates can be produced that allow for both concrete and abstract services to be combined in the same service definition.

The authors also explain how user preferences can be added to the Perform construct of an AbstractProcess, through extensions of the ServiceParameter construct. Preferences specify some restrictions over the service profile and are used during the matchmaking process to generate service ranking.

3.3.3 Abstract Workflows in Grid Environments

The idea of abstract workflows has been researched in a number of papers, though here we want to focus on the Grid environment, Deelman [15]. The idea behind Grid computing is to provide users with the ability to harness the power of large numbers of heterogeneous, distributed resources, such as computing resources, data storage systems and instruments.

Nevertheless due to the complexity of such environments, the user has to struggle with the task of composing workflows by manually discovering and scheduling jobs directly on the Grid. Deelman [15] provides a solution through which the generation of these workflows is automated as much as possible. The abstract workflows are defined to include specifications of application components together with input data. On the other hand concrete workflows are defined as the specialisation of these abstract workflows by bindings to actual executables, data files and specific resources. The generation of concrete workflows is handled, in the presented solution, either by a concrete workflow generator (CWG) or by an abstract & concrete workflow generator.
(ACWG). The former uses a simple mapping technique while the latter uses AI planning.

In brief, the CWG starts from an abstract workflow and maps this directly to a concrete workflow (both workflows are described as directed acyclic graphs) by adding data and resources as required. A number of optimisations were considered to provide for an efficient search in the resource space especially when the number of these resources is large.

The ACWG on the other hand is used to handle both the generation of abstract workflows and the mapping to concrete workflows, through the use of AI planning. In fact the output from the planner is considered to be a concrete workflow.

3.4 Summary

The names abstract processes or workflows, and templates may be considered as being synonymous and are used interchangeably in the previous sections. The main concept is to design an abstract entity (template) that represents generic knowledge about the domain, which is then taken through a specialisation process that plugs in concrete components that transforms the abstract entity into a concrete one. The level of abstraction in these templates is important since it determines the level of reusability of the templates.

Given that an application makes use of such templates (assuming that these are tucked away until needed), it is logical to think of a way in which these templates are efficiently created and retrieved. Typical techniques that we can consider at this point are related to Case Based Reasoning (CBR), since this allows for the creation of cases which can then be retrieved, reused, revised and retained as needed. Each case can encapsulate a template that represents an abstract description of a specific composition.

It is also important to consider the design and definition language for these templates. In the previous sections we saw how OWL-S was used for the underlying definitions of both the services and workflows. Thus we can consider this as the basis of our solution.

Given a template with generic knowledge, it is also important to map this knowledge onto concrete solutions. We saw that AI planning, with its diversity of algorithms, handles well the concretisation of such templates. Though, instead of generating new
plans each time a request is made for a particular service, we saw that starting from a stored, existing plans has its benefits. This couples well with the idea of presenting the requester with cases that represent specific, service compositions as solutions. These cases can be fed to the planner which can then use a discovery engine to bind the abstractions defined in the template with correct services.

In the next chapters we will go into the details of both the CBR-related technology and the design and definition language used to develop the cases that encapsulate these templates.
Chapter 4

Methodologies Employed: Background and Motivations

4.1 Introduction

In the previous chapter we discussed in detail a number of techniques used for service composition. We saw that the ideas behind composition based on service templates are very relevant to the work that we present in this thesis. Nevertheless searching and retrieving relevant templates is still a daunting task and thus we have to identify ways and means which help the service requester by facilitating this process.

One way to do this is to use past experience as a starting point and then be able to adapt the results as required. If the template-retrieval system allows the requester to identify already used templates and possibly allow him to adapt these templates for his present needs, then this limits the number of templates that would need to be created from scratch. When one considers all these issues and tries to map to existing techniques that would help in achieving this goal, one cannot but notice the resemblance to the Case-Based Reasoning (CBR) cycle. In fact CBR provides the user with cases,
that represent past experience, and that are similar to the problem at hand. It also allows the retrieved cases to be adapted to the user’s requirements.

In the rest of this chapter we will look in detail at CBR and other evolving techniques such as Conversational Case-Based Reasoning (CCBR) which is different from CBR in that it allows for the requester’s query to be partially defined. We will discuss several application domains where CCBR is used and also describe yet a special kind of CCBR, which is called Taxonomic CCBR, or TCCBR. We will consider how various CCBR similarity metrics can be used in the underlying template-retrieval techniques in our Personalised Discovery and Composition of Services (PreDiCtS) framework, albeit with some adaptations to the original theories.

4.2 Case-Based Reasoning

Case-based reasoning is an approach to problem solving and learning, where new problems are solved by using past experiences and adapting them to new problem situations Kolonder [38].

Reasoning by using past experiences is a powerful and frequently applied way to solve problems by humans, Aamodt [1] and Kolonder [37]. For example a doctor remembers previous cases to determine the disease of a new patient. A student tries to solve a new problem by remembering how a precedent one was solved. A Web service developer working on a solution that requires a number of service components to be composed will remember how a similar solution was earlier developed by his department.

In a case-based reasoning system experiences are stored as cases in a case-base or case library. The cases in case-base contain the set of problem descriptions and a set of relevant solution descriptions. A case is a contextualized piece of knowledge representing an experience Kolonder [38], Watson [72]. Such an experience, is usually referred to as past case, previous case, stored case, or retained case, while the description of a new problem to be solved, Aamodt [1], is referred to as a new case or unsolved case.

Though CBR could be regarded as an artificial intelligence technique, in many aspects it is fundamentally different from other major approaches. While other AI approaches rely on general knowledge, CBR utilizes specific knowledge of past
experiences to solve new problems. This approach allows for incrementally, sustained learning, since a new experience is retained each time a problem has been solved. This makes it immediately available for solving future problems, Aamodt [1].

The case-based reasoning process is illustrated in Figure 6. When confronted with a new problem, the case-based reasoning system searches for a similar problem description solved previously. Then, the solution of the most similar problem is used as a starting point for generating the solution for the new problem. Finally, the solution is adapted to meet new situations.

Case-based reasoning can be described by the CBR-cycle, as proposed by Aamodt and Plaza (see Figure 7) [1]. Problems are solved by reusing past experiences. Thus, similar previous cases are retrieved from the case library. Then the suggested solution of the retrieved case can be reused. If necessary, the solution is revised and a confirmed solution is built. Finally the confirmed solution is retained in the case-base.

Typically, a case contains a problem description and a stored solution and it is usually described as a set of attribute-value pairs. In case-retrieval, previous cases are retrieved from the case-base in order to solve a new problem. To perform case retrieval, there must be a set of criteria to determine which cases are most suitable for solving the new problem. CBR is based on the underlying assumptions that similar problems have
similar solutions or that the solution of a similar problem is at least a good starting point for solving the current problem.

Therefore the most similar case is retrieved by using appropriate similarity metrics. The nearest neighbour technique is the most widely used technique to calculate similarity in CBR.

To get the nearest neighbours of the current problem, the similarity is calculated for each case in the case-base which is then ranked by their similarity. The similarity of a case from the case-base and the problem case is calculated by adding up all weighted attribute similarities. Similarities are usually normalized into a range from zero to one (zero is completely dissimilar and one is totally similar).

When suitable cases could be retrieved, the knowledge of the retrieved case can be reused to solve the new problem. The solution can then either be directly applied without any adaptation or else after an adaptation process is performed. After the generation of a solution in the reuse phase, case revision follows, i.e. the results of
applying the solution in the real world are evaluated. This step is usually done outside the CBR system, as it requires feedback from the user. When a solution is incorrect, an opportunity to learn from the failure is available and the case may be repaired by detecting the errors of the current solution and retrieving or generating explanations for them. The solution is modified so that the failures do not occur again. A successful solution is retained in the case-base and provides the case-base with additional knowledge, while an unsuccessful solution is removed from the case-base.

4.3 Conversational Case-Based Reasoning

Conversational Case-Based Reasoning tries to fill in the gaps that CBR presents. In CBR, the user is required to fill-in a complete problem specification. The detail presented in such a specification depends upon a number of factors, such as expertise in the area and experience in using the system. On the other hand CCBR allows for interaction with the user so that a more complete problem specification is acquired.

CCBR is based on the ideas behind customer support and recommendation systems and they exhibit the ability to incrementally and interactively acquire the information which more fully describes the customer’s problem. A CCBR system therefore needs only to guide the user through a set of previously determined questions. The result of this conversation is a set of suitable cases whose problem definition is very similar to the user’s query and whose associated solutions will thus be the most suitable that the system can return. In the situation where the user is not satisfied with the returned solutions, a CCBR system provides the facility to adapt or create a new case.

A case as defined by Aha [7] includes the following representation:

(i) **Problem** state $C_p = C_d + C_{qa}$ ($C_d$ represents a case description and $C_{qa}$ a set of question-answer pairs) which encodes a problem that could be solved by the solution $C_s$.

(ii) **Solution** $C_s$ which is a set of actions that could solve the problem $C_p$. Where

$$C_s = \{C_{a1}, C_{a2}, \ldots, C_{aj}\}$$

A CCBR process starts with the user presenting a textual query, representing his problem (see Figure 8). The system then has to compute the similarity between this textual description and the problem descriptions of the cases stored in the case-base.
This yields an initial case similarity ranking whose solutions are displayed for assessment by the user. The system also displays a set of ranked, unanswered, related questions. The user can decide either to use one of the returned solutions, or else, continue with the conversation process by answering some of these questions. The CCBR system adds the answered questions to the query problem specification and updates the displays of both the ranked cases and unanswered questions. As more questions are answered, the problem specification and case-associated solutions become more accurate (that is the system reflects more closely what the user requires).

![Figure 8: CCBR cycle (adapted from Weber [73])](image)

To compute case similarity and the questions to present after each CCBR cycle we can resort to a number of different similarity measures and techniques. In Aha [7] case similarity is based on the difference between the sharing and conflicting QAPairs in both user query and case problem definition. The following similarity function is used:

\[
\text{sim}_1(Q, C) = \frac{\text{same}(Q_{qa}, C_{qa}) - \text{diff}(Q_{qa}, C_{qa})}{|C_{qa}|}
\]
where $|C_{qi}|$ represents the number of QAPairs in the case problem definition. This function is also adopted by Weber [73] but a normalisation function is also included to keep the similarity value between $[0, 1]$. The normalisation function is the following:

$$sim_2(Q, C) = \frac{1}{2} \times (sim_1(Q, C) + 1)$$

The technique used to present and rank new questions during a conversation is based on their frequency in the considered cases. Though, other different techniques could be utilised as defined by Aamodt [3], Gupta [28] and others. In the next section we will look more closely at the applicability of CCBR in different scenarios and explain some of these considerations related to similarity and rankings.

### 4.3.1 Uses of the CCBR technology

CCBR is mostly associated with customer-support systems though its benefits have been tested in various other fields such as:

- (i) Business Process and Workflow Management
- (ii) Software Component Retrieval
- (iii) In connection with Recommendation systems

In what follows we will consider the above scenarios in more detail.

**Business Process and Workflow Management**

Weber in her thesis Weber [73] combines CCBR with rules and presents a solution for business process management. The created prototype is called CBRFlow and allows for more flexibility and adaptability in the management of workflows. The adopted hybrid approach takes the best of rule-based and CBR systems, though the rule-based component is allowed to have some precedence over the case-based component. Rules are generated from domain knowledge while CBR is used when no rules are available or updates to a rule exist in the form of cases.

The CCBR component uses the same case-similarity metric as that supplied with Aha [7]. This similarity is computed by finding the difference between the number of the shared and conflicting observations, and then dividing the result by the total number of observations in a case. A normalisation function is used to set the result within the interval $[0, 1]$. 

43
In Weber [74], CBRFlow is used together with a process management system called ADEPT, Reichert [57], to make process-aware information systems more flexible when deviations from the pre-modelled process schemas are needed and to allow for adaptations to the business process itself.

The approach presented in this work is similar to how we intend to use CCBR in connection with Web service discovery and composition. The process schemas mentioned above will take the form of service templates which include generic service and compositional knowledge. The goal is that of identifying more clearly which type of composition template (case solution) the user requires via this mixed initiative process and presenting the most suitable from those stored within the case base.

**Software Component Retrieval**

In, Aamodt [2], the CCBR technology is used to solve the problem of software component retrieval, especially when the number of components involved is large. The proposed solution is called Conversational Component Retrieval Model or CCRM. A case is the representative of a component and a knowledge-intensive CBR methodology is used to explore the context-based similarities between the user’s query and the stored components.

A frame-based knowledge representation and reasoning system called CREEK is used to unify the component-specific cases and the general domain knowledge. Thus a case is defined by a set of concepts and the relations between them. A knowledge-intensive similarity calculation is used to determine which knowledge in the knowledge base is relevant to the retrieval process and to calculate the similarity between a new case and the stored cases.

The question-answer interaction during the conversation is based on two factors. First it is important that a QAPair is easily understood and secondly only the most informative and discriminating QAPairs should be presented to the user during a conversation. As regards the former, this is catered for by specifying a set of predefined questions together with possible answers for each slot (i.e. for each relation between two concepts) and as regards the latter, an information gain metric algorithm is used to quantitatively measure the information that each slot can provide.
An important aspect which is most relevant to our work is the way in which knowledge is stored in CREEK. Each component (or case) is represented by a frame structure which includes slots (or relations) and slot values (or facets). Thus a frame represents both the component and domain knowledge. In our work we intend to resort to such frame structures through the use of OWL ontologies, though our approach is somewhat different. We want to use an ontology of CCBR related concepts to define cases whose solutions are service templates. These templates will be defined through a process definition language, such as OWL-S, though it is possible to use other languages. For example WS-BPEL could be used, but this will require that each workflow description be mapped to an OWL-S description as in Aslam [8].

**CCBR and Recommendation Systems**

Leake [40] presents a web-based CCBR solution which was able to recommend solutions to scientists seeking resources (such as codes and data) related to an Earthquake Simulation Grid provided by the ServoGrid project [60].

A number of grid related ontologies were developed in RDF and these were used to represent case descriptions. Thus a case is considered to be a set of RDF triples. A domain independent CBR engine based on the Indiana University Case-Based Reasoning Framework (IUCBRF) [33] is used.

The implemented prototype uses the RDF ontologies to present questions about the desired resource characteristics and, typically to the CCBR process, which ranks cases based on the chosen answers. During each iteration, the system provides discriminating questions in a ranked order so that the irrelevant cases are incrementally filtered out.

Each case definition contains the problem and solution descriptions together with bookkeeping information such as the time of case creation, the contexts in which the case applies and also source or provenance information. Both the problem and solution are represented by a set of predefined features, where each feature is an RDF triple. During case-base initialisation, all possible \(<\text{predicate} - \text{predicate value}>\) pairs are extracted from the ontology and presented as features. The case retrieval mechanism is based on a threshold method which compares the set of features present in both user and case problem definitions. Cases are ranked based on the number of common features.
whose values are consistent. Cases with unknown features or having inconsistent feature values are eliminated from the process.

The way in which cases are defined through RDF is consistent with how we envision our own solution. Though in this case, all generated triples are equally considered as possible QAPairs. Furthermore, it seems that no reasoning was done on the RDF data, thus no advantage was taken from this when QAPairs were presented to the user. In our solution we want to be able to exploit as much as possible the logic behind the concepts and relations within a case description by using an OWL reasoner. For example given that a question related to some particular concept has already been presented to the user, it is superfluous to present another question whose concept is more generic than the one associated with the previous question. In this regard it was interesting to look at the work done by Gupta [28] on Taxonomic CCBR, the details of which we will present in the next section.

4.4 Taxonomic CCBR: A special kind of CCBR

Taxonomic CCBR (TCCBR) tries to tackle the pervasive issue of expressing case contents and features at different levels of abstraction. The solution is based on the ability to make use of feature taxonomies.

The motivation behind the use of TCCBR is highlighted by three sources of abstraction:

1. the different levels of domain expertise between users and developers
2. the variations in information availability and the cost of acquiring it
3. the variations in decision-making needs

Ignoring abstraction presents problems such as unwanted correlation between features, redundancy in the number of questions presented to the user during conversation and inconsistencies in the case representations when new features are added. TCCBR is defined to include:

i. A set of questions which are used for indexing the cases. Each question is associated with a set of answers

ii. A set of taxonomies each one representing a set of QAPairs which are related through either an is-a-type-of or is-a-part-of relation

iii. A set of cases each having a problem definition in the form of a set of QAPairs and a solution
Furthermore, in TCCBR two important rules have to be applied to the set of QAPairs in a defined case:

i. Only one QAPair from a particular taxonomy can be included in each case

ii. The most specific available and applicable QAPair is used to represent the case

The process of TCCBR as explained by Gupta [28] is divided into three main tasks (the third is optional though):

i. Case retrieval

ii. Conversation

iii. Case Creation

Case retrieval is subdivided into three main steps referred to as searching, matching, ranking and selecting. During this phase, cases are retrieved and ranked based on the questions that the user has chosen to answer. On the other hand the conversation process involves the identification and ranking of the most appropriate questions to present to the user after each iteration. If no suitable solution is found then a new case may be defined by specifying a new set of questions (or reuse existing questions) and a solution for this new problem.

In what follows we will discuss the main theory behind TCCBR and present the adaptations that we will make to this theory in PreDiCtS to make use of ontologies.

4.4.1 PreDiCtS

In the PreDiCtS (Personalisation of Discovery and Composition of Services) framework, personalisation is based on CCBR. Through the CCBR dialogue process the system strives to obtain a clearer description of what the user requires. Then based on this description a set of suitable cases are presented, each containing a service composition scenario that closely reflects the required needs.

One of the concerns in PreDiCtS is the adoption of CCBR techniques that limit to the minimum the dialogue process, by presenting only those questions which are relevant during a specific cycle. One such technique is TCCBR which has the advantage of considering the relations between QAPairs and can therefore limit the number of redundant questions to present during successive retrieval iterations.
Nevertheless some aspects of TCCBR had to be adapted to work with ontologies which contain domain knowledge in the form of concepts and relations between them. These relations are richer than what is required by TCCBR and this presents a challenging research opportunity which we discuss in Chapter 9. Therefore in the present implementation of PreDiCtS we only consider the is-a-type-of relation which is defined through rdfs:subClassOf.

Figure 9 represents a high-level architecture of PreDiCtS which is based on the CCBR cycle as defined by Weber [73] and Aha [7]. However it also provides for TCCBR as defined by Gupta [28] through the use of domain and task ontologies. Service ontologies are used during the case-creation process when a set of QAPairs is associated with a particular composition scenario. The QAPairs and Case bases both store the relevant knowledge as sets of triples.

![Figure 9: CCBR cycle in PreDiCtS](image)

4.4.2 Step 1: Searching

The searching step in TCCBR starts by getting the user’s textually-defined query, $QT$, and mapping this with the most similar QAPair in $QA$. 

48
For each \( q_p \) in \( QP \), the relevant taxonomy \( T_i \) (see Figure 10), is traversed and all the descendants identified, thus the scope is expanded. For all the descendants, all the associated cases, \( C_Q \), are found. According to the TCCBR rules, each case can only be associated with only one QAPair from a particular taxonomy.

Case set \( C_Q = \{ c_1, c_2, c_3, c_4, c_5, \ldots, c_i \} \)

QAPairs in \( QP = \{ q_p 1, q_p 2, q_p 3, \ldots, q_p j \} \)

In PreDiCtS the system presents the full list of QAPairs though it is also possible to limit this by using a keyword matching process. The QAPairs are not directly taxonomised, but each question is associated with a triple \(<\text{subject}, \text{predicate}, \text{object}>\) which is defined in a problem related-ontology. This implicitly makes a QAPair part of a taxonomy. For the scope of similarity calculations though, we only consider the subject from each triple.

![Figure 10: CCBR taxonomy](image)

These problem-definition ontologies contain concepts which are extracted from a typical domain ontology. Though, the abstract taxonomic relations between concepts do not necessarily imply that one concept is a subClassOf another, rather they mean that one concept is more generic or specific than another and therefore an associated question will be asked before or after another question.

For the TCCBR rules to be applied, we consider an ontology to consist of a number of taxonomies. Where, the roots of each taxonomy are disjoint concepts in the ontology.
Case set $C_Q=\{c_1, c_2, c_3, c_4, c_5, \ldots, c_i\}$

QAPairs in $QP=\{qp_1, qp_2, qp_3, \ldots, qp_j\}$

Ontology $O=\{oc_1, oc_3, oc_5, oc_7, \ldots, oc_k\}$

Referring to Figure 11, $qp_1$ is present in two cases, $c_1$ and $c_3$, and is associated with ontology concept $oc_1$ (which is assumed to be the subject from the associated triple). Similarly $qp_3$ is associated with ontology concept $oc_3$ (which is the subject in yet another triple) and is present in two cases, $c_2$ and $c_4$. In this way $qp_1$ subsumes $qp_3$ based on the relation between the ontology concepts $oc_1$ and $oc_3$. Therefore during a case retrieval cycle, $qp_1$ will be asked before $qp_3$ since it is more generic. In a case $c_i$ there will either be $qp_1$ or else $qp_3$, this provides for a reduction in the redundant questions being presented to the user during case retrieval.

4.4.3 Step 2: Matching

This involves matching each QAPair element in $QP$ with the QAPairs in each of the candidate cases, to establish a ranked list of cases based on the following:
sim (qp_i, qp_j) = \begin{cases} 
1 & \text{if } p_j \subseteq qp_i \\
\frac{n+1-m}{n+1+m} & \text{if } qp_i \subseteq p_j \\
0 & \text{otherwise} 
\end{cases}

where, \( qp_i \) is the QAPair in the user’s query and \( p_j \) is the QAPair in a candidate case

\( n = \) number of edges between \( qp_i \) and the root of the taxonomy
\( m = \) number of edges between \( qp_i \) and \( p_j \)

Having calculated such similarity between QAPairs then an aggregate similarity metric is used to calculate the overall similarity between the user query \( QP \) and a case problem description, \( P_k \). This aggregate similarity is calculated as follows:

\[
Sim (QP, P_k) = \frac{\sum_{i \in QP, j \in P_k} sim(qp_i, p_j)}{T}
\]

where, \( T \) in the original taxonomic theory represents the number of taxonomies, here it represents the number of disjoint branches in the domain ontology, that are associated with the QAPairs. Cases are then ranked in a descending order based on this aggregate value.

In PreDiCtS we adapt the same similarity metric except that this similarity is computed on the QAPairs’ associated concepts rather then on the QAPairs themselves. With reference to Figure 11 above, suppose that in the user’s problem definition there are two QAPairs, \( qp_1 \) and \( qp_4 \), while in the problem definition of the candidate case there are three, \( qp_3 \), \( qp_2 \) and \( qp_5 \).

Based on the associated concepts \( oc_1 \) and \( oc_3 \), the QAPairs \( qp_1 \) and \( qp_3 \) are related by a parent-child relation, while the QAPairs \( qp_4 \) and \( qp_2 \), which are associated with the concepts \( oc_7 \) and \( oc_5 \), are bound by a child-parent relation (see Table 1).

<table>
<thead>
<tr>
<th>User’s Query</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>q-a pair</td>
<td>q-a pair</td>
</tr>
<tr>
<td>concept</td>
<td>concept</td>
</tr>
<tr>
<td>( qp_1 )</td>
<td>( oc_1 )</td>
</tr>
<tr>
<td>( qp_4 )</td>
<td>( oc_7 )</td>
</tr>
</tbody>
</table>

Table 1: QAPair/Concept relations
The user’s query though, does not contain a QAPair in the candidate case that is related with \( qp_5 \). Using the adapted similarity metrics defined by TCCBR, we assume the following values:

\[
\text{sim} (qp_1, qp_3) = \text{sim} (oc_1, oc_3) = 1 \\
\text{sim} (qp_4, qp_2) = \text{sim} (oc_5, oc_2) = 0.33 \\
(n = 1 \text{ and } m = 1)
\]

for which the aggregate similarity will be

\[
\sum \text{sim} (oc_i, oc_j) = \frac{1 + 0.33}{3} = 0.44
\]

where the number of taxonomies here is 3 since:

a. the concepts \( oc_1 \) and \( oc_5 \) represent 2 disjoint branches in the ontology and

b. the concept \( oc_2 \) has to be considered as a separate disjoint branch.

### 4.4.4 Step 3: Ranking and Selecting

Once a set of cases are retrieved these can be rank-ordered in descending order by the similarity score obtained from the previous step. Both the TCCBR and the PreDiCtS theories handle this step in a similar manner.

### 4.4.5 Conversation

This step presents the user with a ranked list of questions derived from the retrieved cases \( C_R \). Both the TCCBR and the PreDiCtS theories handle this conversation in a similar manner.

1. All QAPair taxonomies applicable to the cases in \( C_R \) are considered.
2. The score of each QAPair in a taxonomy is based on the similarity scores obtained for the node-related cases. Each node takes the score of all related cases and the similarity of each parent node is the accumulation of the scores of its child nodes. A backward pass algorithm is used to calculate the score of each node.
3. If the user problem definition contains a QAPair from the taxonomy then the system selects its child nodes, else the most specific node that subsumes the set of retrieved cases is selected.
The question score is a tuple that includes \(<\text{taxonomy score, QAPairs score}\>\) as in Figure 12.

Example: \(s(qp_4) = 1.3 = (\text{Sim}c_6 = 0.2 + s(qp_7) = 0.4 + s(qp_8) = 0.7)\)

4.4.6 Case Creation

During case creation the expert user can define and add a new case to the case base. As already explained earlier in the TCCBR theory a case consists of a case description and a problem and solution state. In PreDiCtS though, a case \(c_i\) is defined as:

\[
c_i = (dsc_i, ext_i, \{q_1a_3,...q_ia_j\}, act_i, hst_i)\text{ where:}
\]

- \(dsc_i\) is a textual description for the particular case.
- \(ext_i\) represents a set of context related features, such as \(Role\) and \(CaseCreator\) information based on the foaf ontology definition.
- \(\{q_1a_3,...q_ia_j\}\) is a representation of the problem state by a set of QAPairs.
- \(act_i\) denotes the solution which is represented by service composition knowledge stored in an abstract template.
- \(hst_i\), is the usage history associated with each case.

Case creation is based on the CCBROnto ontology which is described in more detail in the next chapter. The context description \(ext_i\) is important in an open-world such as
the Web since this will be used as a discriminating feature during case retrieval. The action or solution definition act, will represent the service compositional knowledge and can be defined through any composition language. In PreDiCtS we will be using parts of an OWL-S service description, but the framework can be easily extended to work with other service languages such as WSDL and WS-BPEL. \textit{hst} is another feature which is based on the history of each case. This history information represents information about the case usage and could provide either positive (i.e. case was found useful) or negative feedback (i.e. implying that aspects of the case were not found ideal by past users) to the case user. This history information will be translated into a reputation value during case retrieval.

4.5 Extension to the TCCBR theory

We consider TCCBR as being a natural methodology to use in our work since the taxonomic aspect is reflected in the way that ontologies define concepts. Though, ontologies include other abstract relations apart from that of subsumption which make them more expressive. Thus when resorting to TCCBR we had to make some adaptations to the original theory described by Gupta [28]. In what follows we will discuss and elaborate on these changes.

The original TCCBR theory defines a number of rules whose scope is that of limiting redundancy when questions are presented to the user. We have adapted these rules in whole or part to be able to use TCCBR in PreDiCtS.

Rule 1

Only one QAPair from a taxonomy can be included in a case (i.e. there is no abstract relation between concepts relating each question-answer pair in case). This is the similar to TCCBR, unless these concepts associated to these QAPairs are specifically defined as disjoint within the taxonomy.

Example: Given the above situation a case can contain both questions:

\textit{Accommodation required is Hostel?}

\textit{Accommodation required is Hotel?}

In this way the case covers the situation whereby a user might require staying at both a Hotel and a Hostel which are both subclasses of Accommodation.
Rule 2

The most specific available and applicable QAPair is used to represent the case. We adopt this rule as is defined in the TCCBR theory. We look at a taxonomy as a dialogue composed of an ordered set of nodes (QAPairs). We start by looking at both the domain of discourse and the different services that might be required (in the solution) to solve a particular issue. We extract those classes that are relevant to the problem that we want to model and give them an ordering. This ordering, though abstractly defined through the subClassOf relation, does not always imply that one class is in effect a subClassOf another, but rather that the question associated with that concept will be asked before or after another one associated with another concept.

Rule 3

We consider a QAPair to be associated with a unique concept in the taxonomy. Thus for example, the question:

Accommodation required is Hostel? will be associated with the Hostel concept while, Is accommodation required?, is associated with the Accommodation concept.

Nevertheless we can make use of reification to generate more knowledge about each statement. In fact a question can be associated with a reified statement that treats each component of a triple <subject, predicate, object>, as a Resource.

Example: For the question, Is accommodation required?, a reified statement similar to the one below will be generated.

<owl:Class rdf:ID="Hotel">
   <rdfs:subClassOf rdf:resource="#Accommodation"/>
   <owl:disjointWith rdf:resource="#Hostel"/>
</owl:Class>

<owl:Class rdf:ID="Hostel">
   <rdfs:subClassOf rdf:resource="#Accommodation"/>
   <owl:disjointWith rdf:resource="#Hotel"/>
</owl:Class>
We envision that this technique will allow us, in the future, to work with other types of abstract relations such as those similar to *is-a-part-of* by considering other properties that associate classes together.

### 4.6 Summary

In this chapter we gave a detailed look at CBR and CCBR providing background knowledge on these methodologies and comparisons between them. We then focused on the latter since in our work we will be using a mixed-initiative approach. We looked at uses of CCBR in different research areas, such as business process management, software component retrieval and recommendation systems. We compared these initiatives with our solution and then gave a detailed account of a specific CCBR technique which is called Taxonomic CCBR, in which special attention is given to the way the QAPairs are designed and used within cases. Here we went into quite some length to discuss how this technique will be adapted in PreDiCtS. In the next section we will extend the ideas discussed above by elaborating on the motivating issues, and the design of, an OWL ontology for CCBR which we call CCBROnto and which is at the core of all case definitions in PreDiCtS.
An important aspect of any CBR or CCBR system is the way in which cases are structured and indexed. In general a CBR case representation contains information about the problem that the case aims to solve, the solution to this problem, and maintenance information about the case itself. Cases are similarly structured in CCBR, except that the problem takes the form of a set of QAPairs. In PreDiCtS we have designed an OWL ontology to represent all the underlying concepts defined in each case and in what follows we will discuss the motivating issues behind the design and development of this Ontology which we call, CCBROnto (Ontology for Conversational Case-Based Reasoning).

5.1 Overview

In our research on work related to the definition of the case structure in CBR and CCBR, we came across a number of case-definition formats, which range from textual descriptions, to XML and ontologies. We will naturally focus on the latter since it directly combines this AI technique with the Semantic Web.
In Chen [14] and Diaz-Agudo [17] we find the definition of two ontologies, CaseML and CBROnto. These are both defined for CBR environments rather than CCBR, and thus, do not define concepts related to QAPairs, which are at the core of the CCBR process. In Leake [40], a domain specific ontology, related to grid computing, is used at the core of the case definitions in a web-based recommendation system. Our concern though, is more related to ways how ontologies could be used to define cases in a domain independent manner.

In the following text we start by looking at the motivations behind the CaseML and CBROnto initiatives and their relation to our work. We then continue by discussing some design and implementation issues behind CCBROnto, our Ontology for conversational case-based reasoning.

5.1.1 CaseML

CaseML is defined in Chen [14] as a case-knowledge representation language whose main goal is to integrate web ontologies and adaptation rules. Figure 13 represents the RDF description of the CaseML ontology. The topmost class is the CaseBase which is defined to be of type rdf:Bag and its members are Cases. Each case has both a Problem and a Solution, together with an adaptation rule definition.

![Figure 13: CaseML Ontology Structure](image-url)
Both Problem and Solution are defined as being a bag of Features and characterise case definitions. Each CaseBase has an associated property relating it to a Similarity Assessment class, which defines details about the similarity methods that can be employed. This leaves a place holder for different similarity metrics to be referenced when catering for diverse situations.

The way CaseML is designed was inspiring for our efforts in producing CCBROnto. The examples presented in the relevant paper considered different ways in which the language can be used to define cases for different CBR approaches, such as attribute-value approach and the object-oriented approach. Nevertheless, for CCBR, the problem definition is not just a set of feature descriptions; rather each feature has to be represented through a QAPair. Another issue which is not clear from the paper is related to the similarity assessment measures, which although mentioned in the paper, is underspecified and has been left as a future work option. We think that such references within the ontology could point to concepts related to actual similarity functions as defined in Diaz-Agudo [19] and discussed in the next section.

5.1.2 CBROnto

CBROnto is a mature and stable ontology that describes the main terms of CBR systems. This ontology is used by jColibri, Diaz-Agudo [18], as the underlying case representation language. Though CBROnto has been defined using a number of DL formalisms such as LOOM and RACER, the authors have now decided to use OWL as the case representation language. Figure 14 shows how the ontology is defined in OWL.

The topmost concept in this ontology is a CBROnto-Concept which could be a Case, a CaseComponent, a CaseAttribute, a Datatype or a SimilarityFunction. Each Case has a description that is composed of simple or compound attributes. Depending on the contained attributes, cases can be classified as either simple or compound cases. A case can have either a solution or a result, and could contain different data types. All these features allow for the definition of several case-types.
Since jColibri was designed to be able to support several types of CBR systems such as the simple nearest-neighbour based approaches, the more complex knowledge intensive ones and also textual and conversational approaches, CBROnto is able to support several case structures, ranging from plain attribute-value records to hierarchical trees with composed attributes.

As such, a case is defined as the composition of several individuals, where an individual can be composed by yet other individuals. Cases are related using individuals relations that contain the weight of the attribute they connect.

As with CaseML, CBROnto is not designed to handle the CCBR problem definition. Nevertheless the manner in which OWL is used as the underlying case representation language is important since it allows systems to reason about cases, their properties and also about how to classify them. Another interesting feature of CBROnto is the definition of a similarity function (see Figure 15) concept which allows for different similarity functions to be defined, depending on the type of cases. It is important to have such flexibility within a CCBR system as well, since different metrics could be used to calculate similarities between ontological concepts, as defined in Hefke [30], Ehrig [23] and Aamodt [3].

Figure 14: CBROnto case structure (taken from Diaz-Agudo [19])
CCBROnto: Motivation and Design

5.2 CCBROnto

The research work mentioned in the previous section presented suitable grounds for the design of the case representation that is used in PreDiCtS. The motivation behind the decision to create our own representation was based on the fact that both CaseML and CBROnto are not intended for CCBR and lack definition of problems as QAPairs. Nonetheless, we wanted to maintain the flexibility provided in CBROnto by defining CCBR-related concepts in an application-independent manner.

CCBROnto is an important component of PreDiCtS since it provides for:

(i) case and QAPair definitions, and

(ii) the association of domain, problem and case-specific knowledge.

In CCBROnto the topmost concept is a Case, see Figure 16. Its basic components are defined by the CaseContext, Problem and Solution classes. This structure is motivated by the underlying methodology used in PreDiCtS. In this framework we adopt the CCBR approach to help the user refine his query for a particular service request. The problem description is defined by a set of discriminating QAPairs, which characterize a particular solution. On the other hand, the solution is a place holder for a service
composition template and provides a container of knowledge about composition of generic service components. Ordering information is also considered since this is useful when binding with actual services is performed. In what follows we will explain in more detail the basic Case components and illustrate by means of an example how such a case is defined.

```
<ccbr:Case rdf:ID="Case_CID1">
  <ccbr:hasCaseContext rdf:resource="#CNTXTCase1"/>
  <ccbr:hasSolution rdf:resource="#SOLNCase1"/>
  <ccbr:hasProblem rdf:resource="#PRBCase1"/>
</ccbr:Case>

<ccbr:CaseContext rdf:ID="CNTXTCase1">
  <ccbr:hasProvenanceURI rdf:resource="http://www.health.org"/>
  <ccbr:hasCaseCreator rdf:resource="#CCRCase1"/>
  <ccbr:hasRanking rdf:resource="#RNKCase1"/>
</ccbr:CaseContext>

<ccbr:CaseCreator rdf:ID="CCRCase1">
  <ccbr:hasRole rdf:resource="&role;#Doctor"/>
  <foaf:Person>
    <foaf:name>Joe Black</foaf:name>
    <foaf:mbox rdf:resource="mailto:joe@test.org"/>
    <foaf:homepage rdf:resource="http://www.docpage.org/joe"/>
  </foaf:Person>
</ccbr:CaseCreator>

<ccbr:Rank rdf:ID="RNKCase1">
  <ccbr:hasRankValue>1</ccbr:hasRankValue>
</ccbr:Rank>

<ccbr:Problem rdf:ID="PROBCase1">
  <hasQAPairList>
    <ccbr:QAPairList rdf:ID="QAPLst1">
      <list:first>
        <ccbr:QAPair rdf:about="http://www.health.org/medicalQAP.owl#QAP9"/>
      </list:first>
      <list:rest>
        <ccbr:QAPairList rdf:ID="QAPLst2">
          <list:first>
            <ccbr:QAPair rdf:about="http://www.health.org/medicalQAP.owl#QAP6"/>
          </list:first>
          <list:rest>
            <ccbr:QAPairList rdf:ID="QAPLst3">
              <list:first>
                <ccbr:QAPair rdf:about="http://www.health.org/medicalQAP.owl#QAP5"/>
              </list:first>
              <list:rest>http://www.daml.org/generic/ObjectList.owl#nil</list:rest>
            </ccbr:QAPairList>
          </list:rest>
        </ccbr:QAPairList>
      </list:rest>
    </ccbr:QAPairList>
  </hasQAPairList>
</ccbr:Problem>

<ccbr:Solution rdf:ID="SOLNCase1">
  <hasAction>
    <ccbr:OWLSTemplate rdf:ID="TMPL1">
      <ccbr:hasServiceTemplate rdf:resource="#Case1Service"/>
      <ccbr:hasProcessTemplate rdf:resource="#Case1Process"/>
      <ccbr:hasProfileTemplate rdf:resource="#Case1Profile"/>
    </ccbr:OWLSTemplate>
  </hasAction>
</ccbr:Solution>
```

Figure 16: Case Definition
5.2.1 Context Definition

In Dey [16], the term context is defined as “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”.

We fully agree with this definition and in the Case Context we have included knowledge related to the case creator, case history, ranking value and case provenance, see Figure 17. We have also considered ideas presented in Maamar [41] and Bry [11] which discuss the importance of context in relation to Web Services. In PreDiCtS context knowledge helps to identify, why a case was created and by whom, certain aspects of case usage and the case relevance to problem solving. The Case Creator includes a reference to the Role description that the creator associates himself with, together with a foaf:Person instance definition that describes who this person is.

![Figure 17: CCBROnto Context structure](image)

The motivation behind using foaf is to keep track of reputation knowledge related to the person who created the case. The Case Context also provides a place holder for Case History. The knowledge associated with this feature is important when it comes to case ranking and usage, since it allows users to identify the relevance and usefulness of a case in solving a particular problem. It is also important for the case administrator when case maintenance is performed. Cases whose history indicates negative feedback...
may be removed from the case base. Case Provenance is also used in conjunction with trust issues, since it associates a case with a URL indicating the case origin.

5.2.2 Problem Definition

The Problem state description in a PrEDiCtS case has to allow for different similarity metrics to be used, including the taxonomic theory explained earlier. Every problem is described by a list of QAPairs rather than a bag. This is required since QAPairs have to be ranked when they are presented to the user. Each QAPair is associated with a CategoryName, a Question and an Answer (see Figure 18). Each question has a textual description and is associated with a concept from the domain ontology through the isRelatedTo relation. We further assume that answers could be either binary or nominal-valued. For this reason we have created two types of answer classes, YesNoAnswer and ConceptAnswer. The former is associated with a literal represented by either a Yes or a No. While the latter, requires an association with a concept in some domain ontology, through the previously mentioned isRelatedTo property.

![Figure 18: CCBROnto Problem structure](image)

The motivation behind the use of this property is related to the taxonomic theory, which requires that QAPairs are defined in a taxonomy so that during case retrieval, the number of redundant questions presented to the requester is reduced.

64
In Figure 19, we define a QAPair from the health domain, where the question, “New Patient?” is related to the Statement $QST9Stat$ which is a reified statement for the triple $<NewPatient, is-type-of, Patient>$ (Subject, Predicate, Object), defined in a health ontology and the answer in this case is Yes which is of type $YesNoAnswer$.

In this manner we can classify the QAPair by considering the location of the $NewPatient$ concept and the related triple values within the health ontology. Thus a question related to the concept $Patient$ is considered to be more generic than a question related to the concept $NewPatient$, since the latter is subsumed by the former. In this manner we can apply the taxonomic theory defined by Gupta [28] and at the same time have the possibility to query the set of QAPairs through a query language such as SPARQL [66], apart from using an OWL-based reasoner to infer other knowledge.

Thus during the case creation stage, each question and answer description has to be associated with a triple defined in the domain of discourse. This is similar to how Aamodt [3] associates ontology concepts with pre-defined questions. In PreDiCtS we want to make use of such $<\text{triple} - \text{question description}>$ association so that questions and answers are implicitly defined in a taxonomy. This association is also important when similarities between QAPairs and between cases are calculated.
5.2.3 Solution Definition

The Solution in PreDiCtS provides a hook where composition templates can be inserted. The main goal behind such a structure is to be able to present abstract composition knowledge as solutions to the user’s request and at the same time allow for more flexibility when searching for actual services. In fact each Solution is defined to have an Action which has a description and isDefinedBy an AbstractTemplate. A template can be sub-classed by any service composition description, such as that defined by OWL-S (see figure 20), though as explained earlier any process definition language can be used (e.g. WS-BPEL).

![Figure 20: CCBROnto Solution Structure](image)

As explained in Chapter 2, a service definition in OWL-S is just a place holder for information relating the profile, process and grounding. We are not considering any grounding knowledge at this stage, since this will be tackled later on in the Integration phase when actual service bindings are sought. As regards the profile, we only consider that knowledge which is relevant and which is not tied to specific providers. The profile part of the template includes the definitions of inputs and outputs, profile hierarchy and references to the process and service components. The profile hierarchy is considered to be of particular importance since it represents a reference to the service domain knowledge, that is, it identifies the taxonomic location of a particular set of service profiles. We think that such ontologies will become increasingly more important in
relation to best practice knowledge. The template also provides information related to how a number of service components are combined together. What is most important here, are the control constructs that determine the order of execution of the service components. OWL-S in particular, defines a number of such control constructs, such as **Sequence**, **If-Then-Else**, **Split** and others. The service components themselves are defined through the OWL-S **Perform** construct which associates a particular service component with another by binding its outputs to another service component’s inputs. (An example of a solution defined in OWL-S can be seen in the Appendix C).

### 5.3 Case Studies

In the various research works related to CCBR that were considered, no formal methodology is specified for the design of the QAPairs that model the problem state. Nor was there any specific methodology which mapped these QAPairs with the solution. Thus we tried to come up with our own method of analysing the domain and generating the QAPairs that reflect the associated solution.

In a number of CCBR related papers, Weber [73] and Gupta [28], the printing domain is used as a case study. This domain, as defined in the former paper, consists of around 20 cases each solving a particular printing problem. Now, comparing this domain with say a health related domain, the complexity of the problem definition in the latter is much higher, as are the possible solutions. Thus to illustrate and compare the effectiveness of cases defined through CCBROnto we are presenting two scenarios, of different complexities. The scenarios are:

i. related to the travelling domain

ii. related to the medical domain

The goal in the former example is to describe the problem and solution related to a user who wants to attend for an academic event. This requires domain knowledge about typical events, such as conferences, together with transportation and accommodation information. On the other hand, the second example will consider the creation of a case related to a patient entering hospital with a particular health condition.

When considering these examples we kept in mind the possibility that different similarity metrics and conversation-generation techniques are used within a system. Thus for the travelling domain problem we adapt the methodology presented by Gupta.
[28] and take into consideration the taxonomic aspect of QAPairs, while for the medical domain example we adopt the same methodology as defined by Weber [73].

5.3.1 Travelling Domain Scenario

The situation that we want to model here is related to the problems that are encountered whenever someone wants to go abroad to attend a conference. Such model or case should define the problem from an advisor’s perspective and present a solution based on this knowledge. An advisor in this situation could have the role of a travelling agent, who is asking his client questions to identify what the latter requires and then be able to suggest the best solution.

**Goal:** User is to attend an **event** on some particular **date** in some particular **location**. A part of a travelling domain ontology is shown in Figure 21:

![Figure 21: Travelling Domain Ontology](image)

Looking at the ontology it is noticed that the concept **Person** is associated with three disjoint branches or taxonomies, **Event**, **Accommodation** and **Transport**. Thus the questions should be related to any of these taxonomies.

After having identified the important aspects of the domain, we start by looking at the ontology to identify which typical questions might be asked in this situation.
Questions should ideally capture a single aspect of the domain. For example, the most generic questions that are immediately identified are: *Do you want to attend for an Event?*, *Do you want to use Transport?* and *Do you want to reserve an Accommodation?*. Other questions, such as *Do you want to use Plane?* And *Do you want to stay in a hotel?* can be considered as being subsumed by the former set. The associated answer types for such questions are typically either a *Yes* or a *No*. In Table 2 below we have listed some of these questions and have also associated them with a triple from the ontology. Each QAPair is assigned a unique QID reference for that particular set. The taxonomic representation for this set of QAPairs can be seen in Figure 22.

<table>
<thead>
<tr>
<th>QID</th>
<th>Description</th>
<th>Triples Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Problem is a Travelling Problem?</td>
<td>&lt;TravellingProblem, subClassOf, Problem&gt;</td>
</tr>
<tr>
<td>2</td>
<td>Do you want to attend a Conference?</td>
<td>&lt;AttendConference, subClassOf, TravellingProblem&gt;</td>
</tr>
<tr>
<td>3</td>
<td>Do you need transportation?</td>
<td>&lt;Transportation, subClassOf, AttendConference&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Do you want to use a plane?</td>
<td>&lt;Airplane, subClassOf, Transportation&gt;</td>
</tr>
<tr>
<td>5</td>
<td>Do you want to use a train?</td>
<td>&lt;Train, subClassOf, Transportation&gt;</td>
</tr>
<tr>
<td>6</td>
<td>Do you want accommodation?</td>
<td>&lt;Accommodation, subClassOf, AttendConference&gt;</td>
</tr>
<tr>
<td>7</td>
<td>Do you want to stay in a hotel?</td>
<td>&lt;Hotel, subClassOf, Accommodation&gt;</td>
</tr>
<tr>
<td>8</td>
<td>Do you want to stay in a hostel?</td>
<td>&lt;Hostel, subClassOf, Accommodation&gt;</td>
</tr>
<tr>
<td>9</td>
<td>Is Conference registration required?</td>
<td>&lt;Conference, subClassOf, AttendConference&gt;</td>
</tr>
</tbody>
</table>

Table 2: Travelling Problem Question/Triple association

Looking more closely at the sets of questions \{4, 5\} or \{7, 8\}, it is clear that these capture similar aspects of the domain. The first set is considered to be a sub-set of question 3. In this case a more generic type of question could be created: *Which type of transportation do you require?*. The answer to this question would then be one from a list of sibling concepts that include \{Plane, Train...etc\}. In this way questions are clustered and this facilitates the conversation. Nonetheless it might be the case that the user requires both a *Plane* and *Train* for transportation, in which case both questions 4 and 5 should be included in the case. For this reason we will be only defining question descriptions whose answers are binary-valued, that is *Yes* or *No*. 

69
Since we need to apply TCCBR, we have also to keep in mind the taxonomic aspects of the questions when these are associated with a specific case. In the situation presented above, the creator has the choice of including questions 3, 4, 5. The most reasonable choice should either be one from or both questions 4 and 5. Now, if question 3 is present then question 4/5 will have to be presented eventually anyway. So it is more efficient to directly present questions 4 or 5 and leave out question 3. In the event that the user does not require any transport he will not answer this question.

The appropriate link between the set of QAPairs and the solution has to be defined by the creator. We adopt the methodology presented by Kolonder [36] in which domain and task-knowledge are linked together. The domain is used to provide datatype information relevant to the service inputs and outputs. In our case the task knowledge is defined through an OWL-S definition. Thus for example the triple <Hotel, subClassOf, Accommodation> will provide, in the solution, a generic place holder for a Hotel Reservation service. Other services that might be useful to include in the solution are Flight Booking, Train Reservation, Hostel Reservation and Conference Registration services. Figure 23 represents a UML Activity Diagram of a particular solution for this domain.

---

**Figure 22: Travelling QAPairs Taxonomy**

![Figure 22: Travelling QAPairs Taxonomy](image_url)
5.3.2 Medical Domain Scenario

We now look at a more complex domain such as that presented by the following health related scenario, see Figure 24. It is assumed that such specialised Cases are created by experts which are health policy makers, such as professors and doctors. In what follows we will describe how a particular problem is modelled by a new case. For this situation we will not be considering the taxonomic aspects of the QAPair nodes, so that we can eventually test and compare the CCBR methodologies effectively.

The case creator in this situation is a Doctor John Care who specialises in URTI (Upper Respiratory Tract Infections) and cardiovascular conditions. The case in question that he wants to model represents the situation where a new patient has entered hospital with shortness of breath.

The creator adds context information about himself and any relations that he has with other persons. In this scenario, John has work relations with Professor Mary Nice. Such knowledge provides for a level of trust in the expertise of the creator. A typical definition is through foaf-RDF.

The problem definition includes a number of QAPairs, some of which may be newly defined while others may be reused from the list of existing pairs. When adding QAPairs to a case, the creator has to try to model as close as possible the problem.
Thus Dr. John might consider adding questions such as, “Check patient’s condition?” and “Patient has shortness of breadth?”. A suitable answer is then associated with the chosen question. John could either use a Yes or No type of answer or else one which is related to some concept in the domain ontology. A more elaborate (though not complete) list of question-answer pairs is compiled in Table 3.

These QAPairs are associated with a set of triples as we did in the travelling scenario, though here we are not giving any importance to the abstract relations that might exist between any two QAPairs.
The composition knowledge in this case represents a number of services, which John wants to use in this scenario to efficiently cater for patients entering hospital with the mentioned health condition. This particular functionality is required to monitor the patient from the moment that he enters the hospital until he is comfortably stationed in a room. The service requirements in this scenario could include:

- **Form-filling service**: these services will be used to allow both receptionist and ward nurse to enter preliminary information about the patient. The entered information has to be used by other services in the composition.

- **Room-finding and allocating service**: this service will use information (age, patient’s condition) from the previous services to find a suitable room for the patient.

- **Doctor-OnCall service**: this service sends, in real-time and possibly over a mobile phone, information about the patient to the doctor on-call. The doctor can immediately start considering what should be done and advise the nurse accordingly.

- **Update patient’s records service**: after a thorough examination, the doctor will, identify which preliminary treatment to start, whether any lab tests should be done, and is required to enter this information in the patient’s records.

The service workflow that is chosen by John as a solution for this problem is defined as a UML activity diagram, as shown in Figure 25.

<table>
<thead>
<tr>
<th>QID</th>
<th>Description</th>
<th>Triples Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Patient status is NewCase?</td>
<td>&lt; Subject, Predicate, Object&gt;</td>
</tr>
<tr>
<td>2</td>
<td>Condition is of type Respiratory?</td>
<td>&lt;RespiratoryCondition, is-a-type-of, Condition&gt;</td>
</tr>
<tr>
<td>3</td>
<td>Preliminary examination required?</td>
<td>&lt;PreliminaryExam, is-a-type-of, Examination&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Patient’s age less than 16?</td>
<td>&lt;Age, is-less-than, 16&gt;</td>
</tr>
<tr>
<td>5</td>
<td>GP is required?</td>
<td>&lt;GP, is-a-type-of, Doctor&gt;</td>
</tr>
<tr>
<td>6</td>
<td>History details required?</td>
<td>&lt;Patient, hasHistoryDoc, History&gt;</td>
</tr>
<tr>
<td>7</td>
<td>Patient is admitted to children’s ward?</td>
<td>&lt;Patient, isAdmittedTo, ChildrenWard&gt;</td>
</tr>
</tbody>
</table>

Table 3: Sample for Medical Problem Question/Triple Association
Medical Domain Ontology

Q-A pair

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient status is NewCase?</td>
<td>Yes</td>
<td>NewCase</td>
<td>is-of-type</td>
<td>Status</td>
</tr>
<tr>
<td>Patient's age less than 16?</td>
<td>Yes</td>
<td>Age</td>
<td>is-less-than</td>
<td>16</td>
</tr>
<tr>
<td>Condition is RespiratoryCondition?</td>
<td>Yes</td>
<td>RespiratoryCondition</td>
<td>is-of-type</td>
<td>Condition</td>
</tr>
</tbody>
</table>

Partial list of Question-Answer Pairs

Title: New patient enters hospital with shortness of breath.

Context Knowledge of Creator:
Role: Doctor
Name: John Care
Specialization: URTI and Cardiovascular Conditions
Works with: Profs. Mary Nick

Case Definition

Question-Answer pairs:
- Patient status is NewCase? Yes
- Patient's age less than 16? Yes
- Condition is RespiratoryCondition? Yes

Solution:
Sequence (details, assessment, splits, join to doctor, room allocation, update records)

UML Activity Diagram representation of the Solution

Case Creator

Figure 25: Creating a new case for the Medical Domain
5.4 Summary

In this chapter we explained in detail the motivation behind the creation of the CCBROnto ontology and explained through two different scenarios how this can be applied to create cases based on domains with different complexities. When building the QAPairs for the travelling domain we considered the abstract relations between the concepts associated with these QAPair nodes.

This resulted in the creation of a taxonomy of QAPairs, and therefore this aspect will be considered later on by the CCBR system so that the number of questions to present to the requester during the conversation is limited to the most specific. On the other hand when we created the QAPairs for the medical domain, we considered all that knowledge which is relevant for the particular problem with no consideration whatsoever of abstract relations between the QAPairs. We will later on make use of these scenarios to effectively evaluate our PreDiCtS application and identify which methodology performs best.
After having discussed in detail both the problems related to service reusability and the solutions that we want to adopt, we now discuss the relevant specification and design issues associated with the PreDiCtS prototype. We will use this prototype to evaluate the effectiveness of these solutions.

6.1 Specification

The main goal behind PreDiCtS is to create a framework whereby templates of reusable services can be created, stored, retrieved and adapted as required. For this reason, the adopted approach will be modelled over the CCBR approach, discussed at length in the previous chapters. This methodology combines well with the goal of PreDiCtS and it extends this by providing a mixed-initiative process through which the user’s request is identified more fully.
6.1.1 PreDiCtS overview

The main PreDiCtS’ subsystem can be seen in Figure 26. This includes four main subsystems, namely, the Creator, Retriever, Adaptation and Maintenance.

These subsystems are divided into yet other components which perform specific tasks in order to achieve the full functionality of each subsystem. All subcomponents use the predicts:ccbr package which provides for all the specific CCBR-related functionality. Other third-party packages are also used to provide for OWL-S manipulation and database connectivity.

The Creator will enable the workflow-template designer to create a workflow that includes the appropriate generic service definitions (see Figure 27). This task should be ideally handled by an expert in case creation and domain knowledge engineering. This workflow will be encapsulated within a Case definition which is based on the
CCBROnto ontology. Apart from the workflow, the designer has to also add context information which defines role, trust knowledge in the form of a foaf:RDF description and provenance. This context together with case history usage will later be used to calculate the reputation of both designer and case.

![PreDiCtS Use Cases](image)

**Figure 27: Main PreDiCtS Use Cases**

The Retriever subsystem has to provide both the designer and the requester with an interface through which templates (encapsulated in cases) are retrieved based on a conversational process which tries to identify as much as possible the required template. The designer might require an existing case to be adapted or revised, while the requester might require a case that includes a workflow that can eventually be bound with concrete services and executed.

The Adaptation subsystem will allow retrieved cases to be adapted whenever a case does not reflect exactly the user’s requirements. Adaptation might involve the addition or creation of new QAPairs to the case or the adaptation of the solution itself.

The Maintenance subsystem has to provide both an interface through which adapted cases can be stored within the model store, and also functionality to maintain the model store itself. This subsystem will also allow the designer to edit the underlying case descriptions (context, problem and solution), and then verify and retain these within the store.
6.1.2 Creator Subsystem

The Case-creation process must enable the designer to easily create a new Case which includes the definitions of the Context, Problem and Solution. These definitions will be based on the CCBROnto ontology discussed earlier. By providing an easy-to-use interface and hiding the complexity of these definitions, the designer can concentrate on what he wants to achieve with the new case rather then how the various definitions will be combined together.

The Context knowledge that the designer is required to add has to reflect his credibility and expertise. This will become very useful whenever cases are shared between PreDiCtS users. Such knowledge will provide for an element of trust in the form of a foaf:RDF description with the designer’s information. The system, though, has to be flexible enough to deal with other trust definition languages. Apart from trust in the designer himself, this context knowledge will be used to build a history of case usage. Together with user feedback, this will help in the generation of a reputation value associated with each case. Future case users will therefore be provided with a means to measure the utility of a particular case to solve a problem.

Problems must be represented through a set of QAPairs. PreDiCtS will provide for the creation, reuse and maintenance of QAPairs. Creating QAPairs will involve the use of a mapping tool that associates the newly defined question and answer with concepts from a chosen problem definition ontology. Questions will be defined in plain text while answers could either be binary-valued, that is Yes or No, or else be associated with a concept from the ontology. Reuse of QAPairs will be provided to allow case designers to minimise the time for the case creation process. QAPairs from a chosen problem domain will be displayed and the user can decide whether a QAPair is suitable for a particular case or not. QAPairs’ maintenance (i.e. revising and retaining QAPairs) is a complex process, since changes performed on a particular QAPair can have a negative impact on the existing case base. Therefore even though it is an important aspect, it is out of scope of this work we will not be dealing with it here.

The generation of a solution in a case description has to represent a template for a workflow of generic services. The reason why we opted to create a generic definition has been discussed at length in the previous chapters and is mainly due to the higher flexibility in rehearing such definitions. To generate the solution, PreDiCtS will provide a
workflow composer based on the approach adopted by Scicluna [59] and Buckle [12]. This includes a tool that allows for a workflow to be represented through a UML Activity Diagram and then transformed into a workflow definition. PreDiCtS will provide for multiple workflow definition languages, such as OWL-S and WS-BPEL, to be used. The present prototype will handle only OWL-S, though through the use of a mapping tool, BPEL4WS2OWLS [9], a WS-BPEL description can be successfully transformed into an OWL-S definition. Though such API exists, we will not implement this aspect of PreDiCtS at the moment.

6.1.3 Retriever Subsystem

Retrieval in PreDiCtS will provide the user with the possibility to find cases that fit as much as possible to his needs. This functionality will be provided through a mixed-initiative process that incrementally refines the user’s query and provides cases whose problem definition is similar to the query. This will involve the use of a similarity metric to find the most suitable cases and also a conversation generation algorithm to present new questions for the user during each cycle. To be able to evaluate the effectiveness of our approach in PreDiCtS, the retriever tool has to be flexible enough to allow for different similarity metrics and question-generation algorithms to be used.

The similarity metrics that we will be using are based on those described by Weber [73] and Gupta [28]. The former will provide for a simple calculation based on the direct similarities between the QAPairs. On the other hand the latter will be based on a more complex calculation that considers the taxonomic relations between the QAPairs. Nevertheless it is envisioned that other metrics such as those presented by Hefke [30] will be used in the future. This presents a set of similarity measures which are ideal when comparing ontologies. These similarities are not based solely on the subsumption relation but also on other relations such as distance-based and syntactical similarity. This type of metric combines well with our approach, since both problem and solution can be considered as parts of such a graph and therefore similarity will resolve into comparisons between multiple graph models (multiple, problem or solution definitions).

PreDiCtS will also provide for different conversation algorithms, also based on Weber [73] and Gupta [28]. While the former calculates the frequencies of QAPairs in
both user query and case problem definition, the latter will involve a backward calculation algorithm which was described in Chapter 4.

6.1.4 Adaptation Subsystem

The case adaptation subsystem will provide PreDiCtS’ users with the possibility to adapt retrieved cases whenever a case does not fully solve the problem. The adaptor tool will allow modification to the context, problem and/or solution descriptions in each case. Problem adaptation should provide for the reuse of existing QAPairs or the creation and addition of new QAPairs to a case. To adapt a solution the user will be presented with the workflow composer tool used by the case creation system, but with some added functionality such as a solution editor. A case that requires adaptation will be cloned, adapted and then stored in the case base as a new case. An important feature of the adaptor will be the inclusion of a continuous-suggestion component. This component will help the user by making suggestions based on past adaptation experience.

6.1.5 Maintenance Subsystem

This subsystem will provide for the maintenance of both the case base store and PreDiCtS itself. Figures 28 and 29 represents the use cases associated with this subsystem.

![Figure 28: Case Base Maintenance Use Cases](image)
A case in the case base is stored as an ontology model. The case base will be an RDF triples-store based on a typical database such as MySQL [46]. A case-base maintenance tool will take care of displaying and listing the models within the case base and also allow the addition and removal of models.

![Configuration Use Cases](image)

**Figure 29: Configuration Use Cases**

A configuration tool will deal with the configuration of PreDiCtS itself. The user will be able to define and configure the data to access the case base. This will include data related to the type of the database driver, username and password. Other functionality includes addition and removal of domain and problem definition ontologies used by the other tools.

### 6.1.6 CCBR API

All the functionality related to the CCBR process will be included within the CCBR API. This will provide for the following functionality:

- different retrieval algorithms
- different conversation-generation algorithms
- case base connectivity
- case definitions
- problem and solution, generation and visualisation

The API will also have to be easily extendible to handle any other CCBR approach, and other problem and solution generation and visualisation tools.
6.2 Design

The main parts of PreDiCtS are shown in Figure 30 below. During all the stages of the depicted process, the system will use the CCBR API mentioned earlier.

The user has an important role in the process since in every stage he will be consulted to give some sort of feedback, such as answer questions and to choose which cases to view or adapt. During all parts of the process PreDiCtS tries to hide the complexities of working with the different domain and problem definition ontologies and case models.
6.2.1 Case Creator

During the case-creation process both domain and problem definition ontologies will be used. The former is used to define the concepts used by all the case components. The latter represent QAPair-instances definitions. These are based on the CCBROnto ontology (see Figure 31).

The user (case designer or otherwise) will have to add context, problem and solution definitions to a case instance. Once created the instance is transformed into a set of triples and stored in the case base as a case model. The case base will be built over MySQL and manipulation of the models within the case base will be possible through the Jena API.

![Figure 31: DFD Level 1 for the Case Creator context from Level 0](image)

The context information, handled by the Context handler and required by each case instance will include knowledge about the case creator, such as his role (this will be associated with a role definition ontology), foaf-RDF data (extracted from the user’s own foaf instance definition) and also an initial case-retrieval value. This will usually be defined to be 1.0 since initially each case has the potential of solving the defining problem.
The Problem handler will deal with the problem definition component of each case. Such a definition will consist of a set of QAPairs that are either created from scratch or else reused from specific problem-definition ontologies, see Figure 32. Each of these ontologies will in fact consist of sets of QAPair instances.

![Figure 32: DFD Level 2 for the Problem Handler context in Level 1](image)

If a QAPair is created from scratch the user will add textual information related to the question and answer for that instance. A concept-mapping component will load an ontology from the cache or WWW and the user will create a link between chosen concepts and the question and/or answer descriptions (shown in Figure 33).

![Figure 33: DFD Level 3 for the QAPair Generation context in Level 2](image)
This mapping will generate an \textit{rdf:Statement} which is a reification for the chosen triple. Reification in RDF is the ability to treat a \textit{Statement} as if it were a \textit{Resource} and therefore this provides for assertions to be made about that statement. Thus for example if the textual representation of a question is: “\textit{Do you want to attend a Conference?}” and the chosen triple is “\textit{<AttendConference subClassOf TravellingProblem>}” then a reified statement will be generated as follows:

\begin{verbatim}
<rdf:Statement rdf:ID="QST9Stat">
  <rdf:Subject rdf:resource="http://www.domain.org/travel.owl#AttendConference"/>
  <rdf:Predicate rdf:resource="http://www.w3.org/2000/01/rdf-schema#subClassOf"/>
  <rdf:Object rdf:resource="http://www.domain.org/travel.owl#TravellingProblem"/>
</rdf:Statement>
\end{verbatim}

By treating each triple component as a \textit{Resource}, the system will be able to work with other triples and therefore questions, where this resource is used, such as, \textit{<Conference subClassOf AttendConference>} and \textit{<Conference hasStartDate StartDate>}.  

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure34.png}
\caption{Level 2 DFD for the Solution Creation context in Level 1}
\end{figure}

The task of the Solution handler (check Figure 34) is to create a mapping between the visual diagrammatic representation used to define the template and the underlying service description language. The UML Activity Diagram created by the designer represents a template or placeholder for generic services and from this an OWL-S Process definition is generated. The Process ontology of the OWL-S description language provides for the definition of a workflow of services and related properties. Since we wanted this description to be as generic as possible, each service definition is conceptually linked to an ontology of service-related concepts. Thus if the user adds a
node that represents a *Flight Reservation* service, an atomic service definition will be generated, whose inputs and outputs are also defined in this ontology. Below is the definition of such an atomic process, whose input and output resources are defined in some external service-related ontology.

```xml
<process:AtomicProcess rdf:ID="FlightReservationService">
  <process:hasInput>
    <process:Input rdf:about="#FlightReservationInput">
      <process:parameterType
        rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:about="#FlightReservationOutput">
      <process:parameterType
        rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

In this manner when searching for actual services, these generic placeholders will be bound to actual service inputs and outputs.

The generated definition is then encapsulated within the solution component of a case instance. The generation of the OWL-S definition is done through the use of the third party OWL-S API [50].

### 6.2.2 Case Retriever

The Case Retriever (see Figure. 35) is responsible for the CCBR retrieval process. It takes as input the choice of similarity measure and problem domain and presents questions for the user to answer. The answered questions will then be used to generate a list of cases based on the similarity measure component.

It is up to the user to decide whether a case from the retrieved set of cases is suitable enough to solve his problem. In the situation where further problem-filtering is required, the user can decide to answer more questions, with the consequence that the list of retrieved cases is also filtered down.

The set of questions presented with every step in this filtering process is generated through a conversation-generation component which takes care of identifying which questions are best suited to be presented to the user in the next step. Different
conversation-generation algorithms will be available in PreDiCtS, depending on the type of similarity measure chosen initially by the user.

6.2.3 Case Adaptor

Figure 36 below shows a DFD level 1 for the Case Adaptor component. The input to this component will be one of the retrieved Case instance definitions.

Figure 36: DFD Level 1 for Case Adaptation context in Level 0
Based on the adaptation input from the user, a case is decomposed into its constituent, context, problem and solution definitions. Each definition can then be adapted separately. Problem adaptation will involve the possibility to either create a new QAPair and to add it to a particular problem-definition ontology or else to reuse existing QAPairs. Adapting a solution will involve adaptation to individual components of a template, for example change the sequencing order of services, and/or change the workflow constructs used and/or adding and removing services. Whenever a case is adapted, the initial case is cloned and a new one is generated with the confirmed adaptations.

6.2.4 Maintenance

The Maintenance component includes system configuration and case base maintenance. Configuration information will be stored in an XML document and this includes information related to case base access and the setting of URLs for the domain and problem definition ontologies. Figure 37 shows a DFD representation for the configuration component. An instance of the configuration document can be found in Appendix C.

The task of the case base maintainer is to provide an easy access to the models stored within the case base (see Figure 38). Case definitions are transformed into models of triples through the use of the Jena API. The user can respectively manipulate and query a case model through the OWL interface and through RDF query languages such as SPARQL. In future extensions to PreDiCtS, we envision this tool to either evolve into
an ontology editing suite or else to interface with some established ontology editing software such as Protégé.

![Figure 38: DFD Level 1 for Case Base Maintenance](image)

6.3 Summary

In this chapter we discussed the specification and design issues behind our prototype, PreDiCtS. We discussed how the major components each play an important role in providing tools through which PreDiCtS’ users can create cases, in which the solution is a template with service composition knowledge, retrieve cases using different CCBR approaches, adapt those cases which do not fully satisfy the user’s requirements and finally provide maintenance of both case base and prototype itself.
Chapter 7

Implementation

Based on the previously elaborated specification and design, a research prototype has been developed. The prototype was implemented in pure JAVA using the Eclipse Open Source IDE [22]. The use of JAVA as a programming language has been motivated by its platform independence and the availability of a rich set of Open Source Tools. The prototype has been developed on a Windows based PC and the backend triples store is based on MySQL.

The implementation of the prototype demonstrates how conversational case-based reasoning and the reuse of service templates can be integrated. The prototype focuses on demonstrating the effectiveness of different CCBR methodologies towards the discovery of generic service templates. These templates can then be used as a starting point towards a more effective service matchmaking process.

7.1 Overview

The PreDiCtS prototype integrates a number of tools from the tools package with the core ccbrr package. The tools provide an interface for the main CCBR processes of case
creation, retrieval, adaptation and maintenance. Unfortunately, due to time constraints the adaptation component, though designed, had to be left as part of the future work.

The ccbr package provides all the necessary functionality for the different CCBR approaches and the facilities to communicate with the case base. This package consists of other sub-packages namely:

1. **builders package**: provides functionality to create the case’s sub-components.
2. **tasks package**: each sub-process of the CCBR cycle is considered as a task and this package provides the necessary functionality to initialise and complete each task. The implemented tasks include, `CaseCreatorTask` (handles the instantiation of a CaseCreator GUI), `CaseBuilderTask` (handles the creation of a case-definition instance) and the `CaseRetrieverTask` (handles the case-retrieval process).
3. **rdb package**: provides classes that allow for the effective communication with, and querying of, the case base.
4. **similarity package**: this provides a flexible and easily extendible set of classes whose main goal is that of presenting the similarity metrics and associated algorithms that can be used in PreDiCtS
5. **solutions package**: provides the functionality that allows the composition templates to be created and the mapping from UML Activity Diagrams to a chosen service description language.
6. **utilities package**: provides utility classes that monitor the progress of the different tasks and allow for visualisation of the taxonomies.

Apart from the CCBROnto ontology discussed in detail in Chapter 5, a number of other ontologies had to be designed and built. For the evaluation process, discussed in more detail in the next chapter, we have considered the two domains which were also discussed and analysed in Chapter 5. These are related to the travelling and medical domains. A base ontology for each domain was built, from which, problem-definition ontologies were then developed. These are especially important when comparing the different CCBR approaches that PreDiCtS can utilise. A copy of the ontologies defined in OWL can be found in Appendix C.
Though the prototype requires further work and extensions it will be shown, in this chapter and the next, that the main goal has been reached since concrete knowledge about the effectiveness of the proposed solution has been found.

In the next sections we will discuss in detail the main issues related to implementation of the PreDiCtS’ components.

7.2 PreDiCtSComposer package

This package consists of the main aggregating class, Composer (Figure 39). It is the application’s main user interface and gives the user access to all the other functionalities, including the PreDiCtS’ tools group.

Figure 39: Composer Class
Launching this Composer class initialises a splash screen and during this process a number of important components are loaded. The reason behind the use of this splash screen is not just ornamental but rather as a time-filler during which the configuration data from the `config.xml` is extracted. Based on the extracted information a number of ontologies, including CCBROnto, required later-on during the execution of other tasks, and the information related to setting up and accessing the case base, are loaded.

### 7.3 Tools package

The `tools` package provides all the tools necessary to create, retrieve, adapt and maintain cases. The adaptation tool has not been fully implemented yet. All the main components implement the `Itools` interface and extend a `JInternalFrame` (see Figure 40). This provides for a flexible way in which to add more tools in the future.

![Figure 40: All tools implement the Itools interface](image)

Each tool has its own sub-package which contains all the necessary classes that provide the user interface and define the specific behaviour of the tool. All of them, though, make use of the `ccbr` package which provides all the functionality related to the CCBR
process. In what follows we will look at each of these tools separately. We will discuss implementation issues and explain in each case how the various sub-packages from the `ccbr` package are used.

### 7.4 Case Creation

The `creator` package includes the main class that aggregates all the functionality required by the case designer to generate and build a CCBROnto case definition instance. Whenever a new case is to be generated, the user is presented with a dialog to choose the domain with which this case will be associated. In this way all the relevant domain, problem-definition and service-related ontologies are made available for the creator’s sub-components. The `CaseCreator` class defines the main UI and uses three other classes, which handle a specific aspect of case creation. These classes are found in the `creator.gui` package. Figure 41 shows the relations between the `CaseCreator` and the other classes, which are:

1. **ContextPanel** class: provides the UI to enter information related to context and triggers the generation of the context component of a Case instance definition.
2. **ProblemPanel** class: provides the UI to work with the QAPairs. It provides access to the functionality which allows the creation, reuse and editing of QAPairs.
3. **VisualComposer** class: provides the UI to create UML Activity Diagrams which will eventually be transformed into a reusable service workflow (or composition) template, defined in OWL-S. The approach adapted by this component is based on the work done by Scicluna [59] on the OWL-S editor.
4. The `CaseCreator` uses some functionality defined in the `ccbr` package to create the case definition instances and store it in the case base. In particular it uses the `ccbr.builders` package and the `ccbr.tasks` package. The former provides specific definition builders for context, problem and solution definitions. The latter provides encapsulation and control of the case creation process within a task. Tasks that take some time to complete are run as threads and the UI is kept active by the display of a `ProgressDialog` that shows information about the executing task. Case storage is done through the `ccbr.rdb.PersistentStore` class.
which uses the Jena API. This class provides case base connectivity and case-model manipulation and querying through RDQL.

We will discuss how each of the first three classes mentioned above contributes to the creation of a case instance definition. In our discussion we will consider in detail how the components from the ccbr package, listed in 4 above, are used in the process.

7.4.1 Overview

This process involves the generation and aggregation of context, problem and solution definitions into a case definition instance. The case designer is expected to add relevant context information which is then combined with the information about the problem definition, in the form of a set of QAPairs.
This is then augmented with the solution definition which is generated from a mapping process that takes a UML Activity diagram and transforms it into an OWL-S equivalent service template. Other service definition languages such as BPEL can be used through a similar mapping approach (i.e. from UML Activity Diagrams to BPEL), though this would require a further mapping to OWL-S by using existing APIs such as BPEL4WS2OWL [9].

Whenever the user decides to create a new case, a CaseCreator class is instantiated. The designer is initially presented with a dialog in which he can choose a domain from a dropdown list that is extracted from the configuration data (see Figure 42).

![Figure 42: Domain Chooser](image)

Two threads are then launched. One that triggers a ProgressDialog (from the ccbr.utilities package, see Figure 43) and another that instantiates the CaseCreator encapsulated as a CaseCreatorTask (from the ccbr.tasks package). This design feature provides for a smooth process whenever a task takes some time to complete. This task, instantiates all the GUI components for the CaseCreator, and a connection with the case base through the PersistentStore class.

![Figure 43: ProgressDialog launching Creator task](image)
The case-designer is then presented with a rather complex UI, which is composed from instances of the `ContextPanel`, `ProblemPanel` and `VisualComposer` classes mentioned above (we will refer to these three classes as the Creator’s UI classes), as shown in Figure 44. Context-relevant data is entered and the UML activity diagram that represents the solution is created and validated. The problem definition maybe defined either from reusable, existing QAPairs, or else by the creation of new ones. When satisfied with the case structure, the case-designer triggers the actual creation and storage of the new case-definition instance which is handled by the `CaseBuilderTask`.

![Figure 44: CaseCreator UI](image)

During the execution of this task, calls are made to a method in the Creator’s UI classes (see code excerpts in Figures 45, 46 and 47), which instantiates an appropriate builder class from the `ccbr` package. Context-generation in the `ContextPanel` is therefore handled by the `ContextBuilder` class, problem-generation in the `ProblemPanel` is handled by the `ProblemBuilder` class and the solution-generation in the `VisualComposer` is handled by the `SolutionBuilder` class. A call to the `generateMarkup` method of each builder class generates the actual definition code.
7.4.2 CaseBuilderTask

When the designer clicks on the Save Case button in the CaseCreator UI, a ProgressDialog is instantiated to monitor this process. This class identifies from which class instance it has been called and starts an appropriate task. In this situation a CaseBuilderTask will be instantiated and executed as a new thread.

```java
if (tsk instanceof CaseCreator) {
    CaseCreator cc = (CaseCreator) tsk;
    task = new CaseBuilderTask(this, cc);
}
```

A CaseBuilderTask extends the abstract CCBRTask class and defines its own internal ActualTask class which is executed in a SwingWorker thread. During the execution of this class, continuous status information is sent to the ProgressDialog so that the user is kept informed about the ongoing progress.
ActualTask in CaseBuilderTask

The ActualTask uses the PersistentStore class instantiated in the CaseCreator to create a base model in the case base for the case definition. The createBaseModel method is used to get a temporary connection to the case base and to verify whether a model with a similar name already exists. If this is the case, then a randomly generated figure is attached to the original model name and a new base model created.

```java
if(modName.equals(it.next().toString())){
    found=true;
    modName=modName+java.lang.Math.round(100*(java.lang.Math.random()));
    base = maker.createModel(modName);
    flag=true;
}
```

ActualTask instantiates a CaseBuilder class which is the super class for all the other builder classes. It defines accessor methods for an OWLOntology and an OWLKnowledgeBase models (see Figure 48). Both these models are defined in the OWL-S API [50] and are used to generate and manipulate OWL models.

![Figure 48: Builders Classes hierarchy](image)
A call to the `storeNewSolution` method of the `PersistentStore`, creates a new `OWLOntology` model which is passed on as argument to the `CaseBuilder`. This model is then passed on to each builder, by calling the `generate*` method in the Creator’s UI classes in turn so that the relevant definition is added to the model, as shown in Figure 49.

```java
try{
    if(vsc.validTree()){
        cbuilder.setOWLOntology(vsc.generateSolution());
        pd.addText("Solution Generated");
        cbuilder.setOWLOntology(cp.generateContext(ps.getNewOntModel()));
        pd.addText("Context Generated");
        cbuilder.setOWLOntology(pp.generateProblem(ps.getNewOntModel(),ps.getQAPairsModel("qaModel",ns)));
        pd.addText("Problem Generated");
        pd.addText("Case Saved");
        ProblemPanel.QAPairs.clear(); //clear temp. list of QAPairs
        ps.closeConn(); //closes connection to case base
        done=true; //flag that terminates the thread
    }
} catch(Exception io){
    io.printStackTrace();
}
```

**Figure 49: Adding definitions to the OWLOntology**

### 7.4.3 ContextPanel and ContextBuilder Classes

This `ContextPanel` class provides the case-designer with a set of `textfields` where to enter context-relevant data. Context information is considered important when it comes to identifying whether a solution is relevant to the user’s request. It is also important when trust and reliability values need to be defined that reflect the case-designer’s ability to create useful cases. These values will be based on a feedback mechanism which has not yet been implemented.

When the `ContextBuilder` is triggered by the `CaseBuilderTask`, all the context information is encapsulated within a `Context` instance and passed on to the `ContextBuilder`. A new Jena `OntModel` of the `CCBROnto` ontology is created and this is used in the context-definition generation process that takes place in the `generateMarkup` method.
7.4.4 ProblemPanel and ProblemBuilder Classes

The ProblemPanel class provides the case-designer with two main functions, the reuse or creation of QAPairs and the generation of the problem-definition which is then added to the case-definition model. We will look at both aspects in detail and see how the former contributes to the latter.

**QAPair Creation**

The creation from scratch of a QAPair involves the use of the UI class QAPairDefinition (Figure 50) from creator.gui package. Through this class the user can create a mapping between the question and answer descriptions, and the problem-domain ontologies.

![QAPair Design](image)

*Figure 50: QAPairDefinition UI*

A TaxonomyBuilder is used to display a tree-like structure of the problem-domain in the right hand pane. The designer can associate a concept from this taxonomy display and drag it into the hasConcept or Other fields in the left pane. This data, together with the question and answer descriptions are encapsulated in a QuestionAnswerPair instance and added to a vector of QAPairs defined in the ProblemPanel. When the ProblemBuilder is instantiated this vector is passed on as argument.
ProblemBuilder

As in the case of the other builders, the ProblemBuilder is called by the CaseBuilderTask to generate the definition of the QAPairs. In this builder, the generateMarkup method makes use of two models. The first an OntModel with all the domain-related QAPair instances found in the case base, and the second is an OWLOntology model to which the new QAPairs will be added as part of the new case definition.

The generateMarkup first identifies the number of existing QAPairs in the model retrieved from the case base. This is important because for any problem-domain, a QAPair is assigned a unique identifier. It then takes the Vector with the list of new QuestionAnswerPairs instances and recursively calls the createQAPairList method to create definitions for each QuestionAnswerPair. Two versions of a QAPair-definition instance are created simultaneously. The first is a complete version and is generated by the createQAPairList. This version is added to the OntModel and is then written back to the case base. The other version is a somewhat reduced definition and is generated by the createQAPair method, which is called from within the createQAPairList. This version includes only references to the actual question/answer definition. It is added to the OWLOntology model and contributes to the new case-definition.

7.4.5 VisualComposer Class

Through this component the designer can create a UML Activity Diagram which is then mapped to a service definition based on OWL-S. The VisualComposer (or composer) class makes use of a number of other classes from the ccbr package. These include the SolutionBuilder class from the ccbr.builders package and the OWLSActionBuilder, NewServiceInfo, ProcessProperties classes from the ccbr.solution package. This package can be extended by the addition of different solution handling mechanisms. This may include a set of classes to deal with the mapping from UML Activity diagrams to BPEL and vise-versa. An external package, the umlforowls, from the owlsedit API is used to handle the visualisation and behaviour of the activity diagrams components. In the following text we will look in detail at the interactions between these classes which result in the generation of a service-definition based on OWL-S.
Create the UML Activity Diagram

The composer’s UI provides three main components. The first is a toolbar with a set of buttons that allows the designer to add UML components to a canvas. This canvas makes use of the third-party ComposerPanel class that provides line drawing algorithms. A third component is the orchestration validation tool that provides validation of the created diagram. In the following text we will just give an overview of how the UML components added to a new workflow are handled, but will not go into the working details of this process since this has been done in Scicluna [59]. Though we borrowed the idea from Scicluna [59], most of the underlying code has been adapted to work with different APIs.

Whenever the designer clicks on a component from the toolbar (our composer does not make use of all the UML component tools at the moment), a ProcessProperties dialog (shown in Figure 51) is triggered that allows him to enter the details related to that particular component and to keep track of all the other defined components.

![Process Properties Dialog](image)

**Figure 51: ProcessProperties Dialog**
The `ProcessProperties` class uses the `umlforowls` package to create an object for the particular component and creates a link to any previously defined components. A `Hashtable` called `constructs` is used to store information about all the created components and to keep future instances of `ProcessProperties` dialogs updated, in particular the `Connect To` combo box.

An instance of the `OWLSActionBuilder` (shown in Figure 52), which is created in the `VisualComposer`, is used by the `ProcessProperties` to create a `ServiceInfo` instance and store this in yet another `Hashtable` called `serviceInfo`. This is later used in the generation of the OWL-S service definition.

**Figure 52: OWLSActionBuilder Class**
Once the designer is satisfied with the created workflow, this can be validated with the validation tool, which checks whether the syntax of the workflow is correctly defined. When the designer clicks the *Save* button in the *CaseCreator* UI, all the created objects are passed on to the next phase, during which the actual service-definition generation occurs.

**Create the Mapping to OWL-S**

The UML to OWL-S mapping-process triggered by the *Save* event, is part of the case creation process handled by a *ProgressDialog* instance and the *CaseBuilderTask*. A call to the *generateSolution* method in the *VisualComposer* instantiates a *SolutionBuilder* class that takes as arguments the initial node, the *OWLSActionBuilder* instance, a base URI for the service and the service name. Yet another call to the *generateMarkup* of the *SolutionBuilder*, starts off the actual service-definition generation.

The service-definition generation process considers the object components created previously and maps these to the actual OWL-S definition. The *generate* method in the *SolutionBuilder* works as follows:

```java
if(component.type == INITIALNODE){
    // recursive call to generate
    generate(nextComponent)
}
else if (component.type == ATOMICPROCESS){
    // if next component is a FINALNODE then this is an Atomic service
    if(nextComponent.type == FINALNODE && serviceList < 1){
        generateAtomicService(component)
    }
    else{
        // more service components to come therefore a Composite service
        generateSequence(component)
        if(nextComponent.type == FINALNODE){
            // no more services, generate OWL-S
            create Service, Profile and Process objects for new service
            createSequence(Process, serviceList, OWLOntology)
        }
    }
}
```

This method caters for the mapping of only a subset of UML Activity diagrams components to OWL-S. Basically only a workflow with a single service and another with a number of services in a sequence are handled. Nevertheless it should not be difficult to extend this to handle other components and constructs.
The method therefore checks for these components and identifies:

1. whether the present component is an initial node, in which case a recursive call to *generate* with the next component as argument is made.

2. whether the component is an atomic process node, in which case two conditions may apply.
   a. It could be the case that a workflow with a single service has been created. If this is the case then *generateAtomicService* is called. Through this method a new OWL-S service definition is created by using the *createSingleAtomicService* method in the *OWLSActionBuilder*. This definition is then added to the *OWLOntology* model with the new case definition.
   b. It could also be the case that a workflow with a sequence of services has been created. In this situation *generateSequence* is called until the next component is a final node. This implies that all the services have been handled and that the actual OWL-S definition can be generated through the *createSequenceProcess* method in the *OWLSActionBuilder*. This will then be added to the new case definition model.

Therefore it can be seen that the *SolutionBuilder* feeds into the *OWLSActionBuilder* the appropriate UML component and the latter then generates the appropriate OWL-S code by using the OWL-S API.

### 7.5 Case Retriever Package

The *retriever* package contains the *CaseRetriever* and *CaseDisplay* classes that provide the UI for the CCBR case retrieval process. The former is directly related to the retriever process while the latter is used to display those cases that the user decides are similar enough to solve his problem. The main retrieval functionality is defined in the *ccbr.similarity* package.

#### 7.5.1 Overview

When the user chooses to search for a suitable case that solves a particular problem, he is presented with the *CaseRetriever*, whose UI is divided into three main components (see Figure 53). The top-most pane consists of two combo boxes; one displays a list of
problem-domain ontologies while the other displays the different types of similarity methodologies that the retriever is capable of using. At present this is limited to two, the Default CCBR (implying that if no other technique is chosen then this is adopted) and the TCCBR (Taxonomic CCBR) methodologies, we envision the use of a graph-based retrieval method, GCCBR, in the near future. Their respective classes can be found in the ccbr.similarity package together with other relevant classes. The difference between these two methodologies has been discussed in the Chapter 4, and this mainly depends on the similarity measures and conversation generation algorithms.

The middle left and right panes show two tables. The table on the left presents those questions that still require an answer from the requester. These are generated by the conversation-generation algorithm. They act as a filtering and refinement mechanism for the list of retrieved cases. The middle right-hand pane presents a table of already answered questions. The bottom pane also contains a table component, but this one shows the cases that have been retrieved during a CCBR retrieval cycle. Each retrieved case is clickable, upon which a CaseDisplay instance is triggered that shows the details of the clicked case instance. It is envisioned that in future extensions, this will present a
better visualisation of the case instance and will also allow a case to be adapted by using PreDiCtS’ adaptation package.

Whichever retrieval technique is chosen, this process is handled through a ProgressDialog that launches the CaseRetrieverTask. The DefaultCCBR and the TaxonomicCCBR extend the AbstractCCBR class. The latter, on the other hand, implements the IAbstractCCBR interface (see Figure 54).

Figure 54: Case Retrieval Process Classes
Each of the different CCBR handling classes uses a specific similarity measure which is defined in the *similarity.measure* package. The similarity calculations consider both case and QAPair similarity. The basic interface is called *ISimilarityMeasure* and both the *CCBRSimilarityMeasure* and *TaxonomicSimilarityMeasure* implement this interface (shown in Figure 55). A placeholder class for the similarity calculations for the graph-based methodology, *GraphSimilarityMeasure*, is already in place.

![Diagram of similarity measure classes hierarchy](image)

**Figure 55: SimilarityMeasure class hierarchy**

In the next section we will look in more detail at the implementation issues of each of the retrieval methodologies provided by PreDiCtS. In particular we will consider the different similarity measures and conversation-generation algorithms of each methodology and how the *CaseRetriever* interact with each one.

### 7.5.2 CaseRetrieverTask

This is launched by the *CaseRetriever* via a *ProgressDialog*. It handles the case-retrieval process independently of the choice of CCBR technique chosen by the user. The retrieval process is divided into two steps. The initialisation step, called *STEP_1*, represents the initial phase of the case-retrieval process in which the system is preparing
itself (the UI and functionality), to start the retrieval cycle. The other step, referred to as \textit{STEP\_2}, represents the rest of the retrieval cycle, from when the user decides to start searching for a case, until he has chosen a particular case that fully or partially satisfies his needs.

\textbf{STEP\_1}

During the first step, the main task is to get a reference to the problem-domain definition model which is stored in the case base through the \textit{PersistentStore}. The model is queried (via a SPARQL query) and all the instances of QAPairs (in the form of a set of triples) within the model are retrieved. From these triples information related to each question and answer, such as question and answer description, QAPair ID and the concepts related to both question and answer, are returned and stored in Vector of \textit{QuestionAnswerPair} objects.

After this process, the type of CCBR technique is identified and based on this, a new instance of either a \textit{DefaultCCBR} or a \textit{TaxonomicCCBR} is initialised. A reference to the chosen technique is passed on to the \textit{CaseRetriever} for future use. The Vector of \textit{QuestionAnswerPair} objects is passed to the constructor of either class. The difference between the \textit{DefaultCCBR} and the \textit{TaxonomicCCBR}, at this stage, is that the former requires that the initial list of questions to present to the user is ranked; therefore a call is made to the \textit{initQAPs} method. On the other hand, in the constructor of the latter, the returned list of questions is processed to create a taxonomy of \textit{QuestionAnswerPair} objects (will discuss this in detail in section 7.5.4).

The processed vector of \textit{QuestionAnswerPair} is then passed on to the \textit{CaseRetriever} and displayed in the table of unanswered questions.

\textbf{STEP\_2}

After the user answers some of the initial questions the actual case-retrieval process starts. The \textit{CaseRetrieverTask} takes as input these answered questions and polymorphically creates an instance of the appropriate CCBR handling class (either \textit{DefaultCCBR} or \textit{TaxonomicCCBR}). The super method, \textit{ccbrCyle} is called, and this handles the CCBR cycle. Through this method the \textit{findRankCases} and the
findRankQuestions methods of the appropriate CCBR process-handling class are called. During each cycle, the retrieved cases and the set of questions, generated by the conversation-generation algorithm, are passed on to the CaseRetriever and displayed in the respective tables.

The process continues until the user is either satisfied with the status of case-filtering reached and chooses a suitable case or else decides that no such case has been retrieved and either case adaptation or creation are required.

7.5.3 Default CCBR Methodology

The DefaultCCBR class uses the CCBRSimilarityMeasure class to compute both case and question similarity. This technique requires that initially a ranked list of unanswered questions has to be presented to the user. This is done through the initQAPs method which uses the rankQuestions method. The latter makes use of the internal class RankComparator that implements the Comparator interface. The QuestionAnswerPair objects are ranked based on their frequency in the case models existing in the case base. To calculate this frequency, the calculateQuestScore method of the CCBRSimilarityMeasure is used. The frequency value is stored within each QuestionAnswerPair object for future use.

```java
for(Enumeration en=ansQAPLst.elements();en.hasMoreElements();){
    QuestionAnswerPair qap= (QuestionAnswerPair)en.nextElement();
    for(Enumeration cn=caseLst.elements();cn.hasMoreElements();){
        Case c =(Case)cn.nextElement();
        if(c.getProblem().contains(qap.getQAPairID())){
            qap.incrFrequency();
        }
    }
    qaps.add(qap);
}
```

The case-ranking process is handled by the findRankCases method. This in turn uses the calculateCaseSimilarity method in the CCBRSimilarityMeasure. The similarity process considers the number of QuestionAnswerPairs in the answered list and those found in each case instance. If a case contains a particular question then a same value is incremented, otherwise a diff value is incremented.
These values, together with the number of QAPairs in each case, are then used in the
calculation of the case similarity. $simVal1$ is used to store the initial similarity value,
while the $simVal2$ is a normalisation of $simVal1$, so that this ranges between 0 and 1.
Each case is then assigned $simVal2$.

```java
for(Enumeration qen=ansQAPLst.elements();qen.hasMoreElements();){
    QuestionAnswerPair qp=(QuestionAnswerPair)qen.nextElement();
    if(c.getProblem().contains(qp.getQAPairID())){
        same++;
    }
    else
        diff++;
}
simVal1 = ((same-diff)/(c.getProblem().size()));
simVal2 = (0.5 * (simVal1+1));
c.setCaseSimilarity(simVal2);
```

The $\text{findRankQuestions}$ handles the conversation-generation process. In the
$DefaultCCBR$ this is a simple process based on a similarity-based threshold value. This
value is used to present the user with the questions that are included in those cases
whose similarity is greater than this threshold.

### 7.5.4 Taxonomic CCBR Methodology

The taxonomic CCBR technique is more complex than the one previously described.
The main goal behind this complexity is a reduced amount of questions to present to the
user in every stage. For this reason the taxonomic features of the problem-definition
domain have to be taken into consideration. As explained in Chapter 4, in PreDiCtS we
create a taxonomy of QAPairs by considering the associated ontology concepts defined
by the designer when creating QAPairs.

The $\text{TaxonomicCCBR}$ class, through its constructor, creates this taxonomic aspect of
$QuestionAnswerPair$ objects by calling the $\text{createQAPairTaxonomy}$ method. For each
$QuestionAnswerPair$ returned by querying the problem-definition model, this method,
uses the respective associated concept to find the direct sub-concepts through the
TaxonomyModel.getDescendentNodes. The returned set of QuestionAnswerPair children is stored in the parent QuestionAnswerPair through the setSubQuestionsList.

Case Similarity

The case similarity is handled through the findRankCases. The set of answered questions from the CaseRetriever is passed as argument to this method, which uses this to find all the descendent QuestionAnswerPair objects (that is, all the descendent objects along a particular branch, not just the direct descendents). Then for each case-model in the case base a new Case object is created. Its problem definition includes a set of QuestionAnswerPair objects.

The calculateCaseSimilarity method (shown in Figure 56) from the TaxonomicSimilarityMeasure uses this list to compute the respective similarity values.

```plaintext
for each Case c in the case base
    simValues = new double[problem size in c][size of answeredquestLst]
    for each QAPair qpc in the c
        for each QAPair qaa in answeredquestLst
            if(qpc in descendentLst of qaa)
                simValues[i][j] = 1
            else
                simValues[i][j] = 0
            endif
        endfor
    endfor
    aggrSimValue= aggCaseSimilarity(simValues)
    set aggrSimValue for c
endfor
```

Figure 56: Case Similarity calculation algorithm

The algorithm checks whether a QAPair in the Case object is in the descendent list of each QAPair in the list of answered questions. A value of 1 is assigned if subsumption (QAPair in the answered QAPairs-list is subsumed by the QAPair in the Case) exists and a 0 otherwise. The original Taxonomic theory considered also the situation whereby a QAPair in the Case is a parent of a QAPair in the answered QAPairs-list. But given that there are differences in the way that the initial list of questions is presented to the user, this aspect is not considered (an explanation for this was given in Chapter 4). An aggregate similarity value is then computed in the aggCaseSimilarity method as follows:
The aggregate similarity algorithm in Figure 57 above, counts the number of 1s in the `simValues` array of computed taxonomy-dependent similarities. It also considers the number of encountered 0s along a column. A check on each column verifies whether all values in the column are in fact 0s. If this is the case then the `tax` value, which represents the number of taxonomies to consider, is incremented. The computation of the aggregate similarity value is based on the total number of 1s in the array and the total number of taxonomy branches consulted during this case similarity process. In general, the number of taxonomy branches is calculated by taking into account the number of QAPairs in the problem-definition of each Case and adding any number of columns in which the similarity values are all 0s.

**Conversation-Generation**

The generation of the list of unanswered questions to present during each Taxonomic CCBR cycle is based on a backward-pass algorithm, which was amply discussed in Chapter 4.

The `findRankQuestions` method in the `TaxonomicCCBR` class, handles this process by using the `calculateQuestScore` method found in the `TaxonomicSimilarityMeasure` class (shown in Figure 58). In brief the score of each `QuestionAnswerPair` depends on

```plaintext
col = number of columns in simValues[][]
row = number of rows in simValues[][]
tax = 0;//counts the number of taxonomies
sim = 0;//stores total number of 1;
for each col i
    cntc = 0 //counts the number of 0’s in a column
    for each row j
        if(simValues[j][i]> 0)
            sim++
        elseif(simValues[j][i] == 0)
            cntc++
            endelseif
        endif
    endfor
    if(cntc == number of rows)
        tax++
    endif
endfor
aggSimValue = sim / (number of rows in simValues + tax)
```

**Figure 57: Aggregate Similarity calculation algorithm**
the computed similarities of the cases that are indexed by it and the summation of the
generated similarities of its descendents.

```java
for each QAPair qaa, in answeredQAPairLst
    getSubQuestionsList of descendanth QAPairs
    score = 0
    if(getSubQuestionsList is not empty)
        for each QAPair qad in descendents List
            score += setQuestScore(qad, caselst)
        endfor
        setScore(score) for qaa
    elseif
        score += setQuestScore(qaa, caselst)
        setScore(score) for qaa
    endelsif
    endif
endfor
```

**Figure 58: Calculate Question Similarity**

Thus the algorithm (shown in Figure 59) starts at the leaves of the QAPair taxonomy
and sets its score to the summation of the similarity values coming from the associated
cases. Moving backwards in a child to parent direction, the score of a parent QAPair is
computed by adding together, the similarities from associated cases and the similarities
from its descendent QAPairs. This process is handled successfully by the recursive call
to the method `setQuestScore`.

```java
double setQuestScore(QuestionAnswerPair qap, Vector caselst){
    double score=0;
    for (Enumeration cen=caselst.elements(); cen.hasMoreElements();)
        Case c = (Case)cen.nextElement();
        if (c.getProblem().contains(qap.getQAPairID())){
            // compute score by adding case similarity
            qap.setScore(c.getCaseSimilarity());
        }
    // calculate score for descendents
    Vector descQAPs=qap.getSubQuestionsList();
    if (descQAPs.isEmpty()){
        score=qap.getScore();
    } else{
        for (Enumeration en=descQAPs.elements(); en.hasMoreElements();){
            score+=
                setQuestScore((QuestionAnswerPair)en.nextElement(), caselst);
        }
        qap.setScore(score);
    }
    return score;
}
```

**Figure 59: Calculating Question scores**
7.6 Maintenance Package

The maintenance package includes a tool for the maintenance of the case base together with another to configure PreDiCtS. The ModelStoreMaintenance class provides a simple UI through which the models in the case base can be added, viewed, edited and deleted. The case base contains two types of models, those representing cases and those that represent the problem in the form of sets of QAPairs. Adding and editing of both case and QAPairs models is best done through the appropriate creation tools (CaseCreator for creation of cases, CaseAdaptor for editing and changing cases and QAPairDefinition for QAPair creation). Nevertheless this tool provides a means to add case and QAPair models directly to the case base. We envision that case models, in the form of case-definition instances can be shared. In such situations, the ModelStoreMaintenance will become very important.

![Figure 60: Configurator UI](image)

The other tool is provided through the Configurator class. This provides a UI which is used to configure PreDiCtS and is shown in Figure 60. The user can define the information to access the case base. This includes the db driver used, username and password. The tool also provides means to store the different URIs related to the domain, problem-definition and core ontologies used. This Configurator makes use of the ConfiguratorData class to generate an XML file called config.xml. This is saved in
the same maintenance directory. The ConfiguratorData has a set of methods that are used to parse this configuration file and extract the relevant data, which is categorised and presented to the requested class in the form of a HashMap. This information is used by a number of other tools, such as the CaseCreator and CaseRetriever.

### 7.7 Summary

In this chapter we went through the implementation details of PreDiCtS. We looked at the various packages and how the contained classes interact with each other to provide the tools and CCBR related functionality. In fact PreDiCtS provides for case and QAPairs creation, case retrieval and maintenance. The adaptation functionality is still in the pipeline.

The case creation process allows the designer to create all aspects of a case-definition instance through various components, including a visual tool that maps the created UML Activity diagram to a service-definition such as OWL-S. On the other hand case retrieval is based on two CCBR techniques. The first is a simple one, and we called this the Default CCBR, while the second considers the abstract taxonomic aspect of QAPairs to reduce the number of questions to present the user during the retrieval cycle.

In the next chapter we will evaluate these tools by using the domains that were analysed in Chapter 5. We will check how effective PreDiCtS is in finding solutions to the different problems related to these two domains by referring to a number of examples and tests that were carried out.
Testing and Evaluation

In this chapter we want to highlight whether the goals that we targeted at the start of this work were reached and whether the adopted technique was successful or not. We will evaluate the developed prototype by considering case-creation, which uses the CCBROnto ontology, and case-retrieval for two different domains. During this evaluation we will make reference to a number of examples and tests that were carried out.

8.1 Testing PreDiCtS Retrieval System

One of the main goals of this thesis and the developed prototype was to integrate and verify the effectiveness of CCBR in the reuse of process templates and as a pre-processing step to the actual service discovery and composition process. For a better understanding of the implications in the use of CCBR, PreDiCtS provides the user with two retrieval techniques. One which uses a similarity measure based on QAPairs’ frequency and another which is based on the taxonomic relations between QAPairs. To test our prototype we have designed two scenarios, which we discussed at length in
Chapter 5, one related to the travelling domain and the other related to the medical domain. We have developed two ontologies for these domains, from which it was possible to generate a set of QAPairs that represent particular problems related to these domains.

In the following sections we will use these domains to show the effectiveness of each of the CCBR similarity measure adopted in PreDiCtS.

8.1.1 Default Similarity Measure

To test this similarity measure, we used the medical domain. Testing mainly focused on the suitability of the retrieval of the most similar case (MSC) to solve a particular problem and was based on a small set of cases and QAPairs.

Requests related to the Medical Domain

We start by listing the questions that were generated for this domain, see Table 4. The domain ontology can be found in Appendix C.

<table>
<thead>
<tr>
<th>QID</th>
<th>Description</th>
<th>Answers Available</th>
<th>Main Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is patient admitted to ward?</td>
<td>Yes</td>
<td>#Ward</td>
</tr>
<tr>
<td>2</td>
<td>Patient’s status?</td>
<td>NewCase</td>
<td>#Status</td>
</tr>
<tr>
<td>3</td>
<td>Patient has pathological results?</td>
<td>Yes</td>
<td>#PathologicalResult</td>
</tr>
<tr>
<td>4</td>
<td>Patient has shortness of breadth?</td>
<td>Yes</td>
<td>#ConditionCause</td>
</tr>
<tr>
<td>5</td>
<td>Condition type?</td>
<td>Respiratory</td>
<td>#Condition</td>
</tr>
<tr>
<td>6</td>
<td>Patient’s status?</td>
<td>OldCase</td>
<td>#Status</td>
</tr>
<tr>
<td>7</td>
<td>History details required?</td>
<td>Yes</td>
<td>#HistoryDetails</td>
</tr>
<tr>
<td>8</td>
<td>Preliminary examination required?</td>
<td>Yes</td>
<td>#PreliminaryExam</td>
</tr>
<tr>
<td>9</td>
<td>Condition type?</td>
<td>HighFever</td>
<td>#Condition</td>
</tr>
<tr>
<td>10</td>
<td>GP is required?</td>
<td>Yes</td>
<td>#GP</td>
</tr>
</tbody>
</table>

Table 4: QAPairs for the Medical domain

We used these questions to generate a set of cases. For each case we defined a solution which includes a subset of services from the list below:
1. **NewCase Service**: requires information about the patient as input and generates a history document for the patient which is stored in the hospital’s database.

2. **OldCase Service**: requires the patient’s ID and the doctor’s ID and returns all the documents related to that particular patient.

3. **Assessment Service**: requires as input a patient’s and doctor’s ID and information related to tests that will be carried out and as output returns a number of appointments together with relevant information about the tests. This information will include the test-preparation details for patient.

4. **HistoryFinding Service**: requires as input the patient’s ID and doctor’s ID and returns an electronic version of the patient’s history document.

5. **ResultFinding Service**: requires as input the patient’s ID and the doctor’s ID and returns electronic versions of the patient’s results. If the test type is known then the search would be more specific and would only return that particular test’s results.

6. **RoomAllocation Service**: requires as input a patient’s ID together with some other relevant information and the admitting doctor’s ID and returns a ward/room number together with details about the patient’s dietary requirements.

7. **DoctorOnCall Service**: requires as input a nurse’s ID and doctor’s ID and generates a message as output that is sent to the doctor either through a sms or through a paging system.

Table 5 shows the list of cases that were generated. Each case has a unique ID in the particular domain, together with the services included in the solution (the services are assumed to be defined in a sequence, since the other OWL-S control constructs are not implemented yet) and the QAPairs ID.

We have assumed that this case base is used by a hospital and simulated a number of problems to test PreDiCtS’ retrieval system. For each problem we list the questions that were answered, the cases returned (only the first three cases are returned, and their similarity values are displayed). The first case in the ranked list represents the MSC. The threshold value for the conversation-generation is set to > 0.4.
<table>
<thead>
<tr>
<th>Case_ID</th>
<th>Solution</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID1</td>
<td>NewCase + DoctorOnCall</td>
<td>QID2, QID10</td>
</tr>
<tr>
<td>CID2</td>
<td>NewCase + DoctorOnCall + RoomAllocation</td>
<td>QAP1, QAP2, QAP4, QAP10</td>
</tr>
<tr>
<td>CID3</td>
<td>NewCase + DoctorOnCall + RoomAllocation</td>
<td>QAP1, QAP2, QAP5, QAP10</td>
</tr>
<tr>
<td>CID4</td>
<td>OldCase + Assessment + DoctorOnCall</td>
<td>QAP6, QAP8, QAP9, QAP10</td>
</tr>
<tr>
<td>CID5</td>
<td>OldCase + ResultFinding + DoctorOnCall</td>
<td>QAP3, QAP6, QAP10</td>
</tr>
<tr>
<td>CID6</td>
<td>NewCase + Assessment + DoctorOnCall + RoomAllocation</td>
<td>QAP1, QAP2, QAP4, QAP8, QAP10</td>
</tr>
<tr>
<td>CID7</td>
<td>OldCase + Assessment</td>
<td>QAP2, QAP5, QAP8</td>
</tr>
<tr>
<td>CID8</td>
<td>OldCase + Assessment + HistoryFinding + DoctorOnCall</td>
<td>QAP4, WAP6, QAP7, QAP8, QAP10</td>
</tr>
<tr>
<td>CID9</td>
<td>OldCase + Assessment + RoomAllocation + DoctorOnCall</td>
<td>QAP1, QAP5, QAP6, QAP8, QAP10</td>
</tr>
</tbody>
</table>

Table 5: Cases for Medical Domain

Problem 1: *The patient is a new case and the doctor is required*

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID1</td>
<td>1.0</td>
</tr>
<tr>
<td>QID10</td>
<td>CID3</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>CID2</td>
<td>0.75</td>
</tr>
</tbody>
</table>

The result of 1.0 for CID1 is reasonable since the answered questions reflect exactly the problem definition in this case. Cases CID2 and CID3 are both 0.75 similar to the requester’s problem since the problem definition in each contains two other QAPairs, not answered by the requester.
Problem 2: The patient is a new case and the doctor will perform some tests

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID6</td>
<td>0.8</td>
</tr>
<tr>
<td>QID8</td>
<td>CID1</td>
<td>0.75</td>
</tr>
<tr>
<td>QID10</td>
<td>CID7</td>
<td>0.67</td>
</tr>
</tbody>
</table>

The MSC here is CID6 since the problem definition contains the answered questions but it also contains other questions which the requester has not answered.

Problem 3: The patient is an old case and the doctor will perform some tests

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID5</td>
<td>CID9</td>
<td>0.9</td>
</tr>
<tr>
<td>QID6</td>
<td>CID4</td>
<td>0.75</td>
</tr>
<tr>
<td>QID8</td>
<td>CID8</td>
<td>0.7</td>
</tr>
<tr>
<td>QID10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When the answered questions’ list contained QID5, QID6, QID8 and QID10, CID9 was returned as the MSC. We can assume that the requester also answers QID5 since it is reasonable to think that the patient is an Old Case and therefore the doctor has already some knowledge about his past health conditions. It is also reasonable to add question QID7 since an OldCase would most probably have an existing history document available. This increases the similarity of CID9 to 1.0.

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID5</td>
<td>CID9</td>
<td>1.0</td>
</tr>
<tr>
<td>QID6</td>
<td>CID8</td>
<td>0.8</td>
</tr>
<tr>
<td>QID8</td>
<td>CID4</td>
<td>0.625</td>
</tr>
<tr>
<td>QID10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QID7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We can also notice the swapping, in the ranking positions, of CID4 and CID8. The former was more similar when QID (5, 6, 8, 10) were answered, since its problem definition contains QID (6, 8, 9, 10). It became less similar when more QAPairs were answered. On the contrary, CID8, was less similar initially because it contained more
QAPairs then the list of answered QAPairs. But its similarity value increased when the answered QAPairs reflected better the problem definition within the case.

**Problem 4:** *The patient is an old case and the doctor requests that the patient is kept under observation.*

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID1</td>
<td>CID3</td>
<td>0.75</td>
</tr>
<tr>
<td>QID5</td>
<td>CID9</td>
<td>0.7</td>
</tr>
<tr>
<td>QID6</td>
<td>CID8</td>
<td>0.5</td>
</tr>
<tr>
<td>QID10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this situation CID3 is ranked as the MSC. Though on inspection, this case includes the conflicting QAPair, QID2. At the moment in PreDiCtS, though we allow the same questions to be defined more than once with different answers, the similarity measure does not take this into consideration and therefore, the result may be misleading. If the requester answers further questions to further filter the cases, the result may improve.

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID1</td>
<td>CID9</td>
<td>0.8</td>
</tr>
<tr>
<td>QID5</td>
<td>CID3</td>
<td>0.625</td>
</tr>
<tr>
<td>QID6</td>
<td>CID8</td>
<td>0.6</td>
</tr>
<tr>
<td>QID10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QID7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here the requester has answered the QID7, which is logical in this situation, since the patient is an Old Case and therefore, a history document will most probably already exist in the hospital’s database. This increases the similarity of CID9 to the user’s problem, relegating CID3 to position two. Furthermore, answering QID8 reflects better CID9 and the similarity value will be increased to 0.9.

**Concluding Remarks**

On the whole the results were satisfactory and any misleading results were accounted for. This similarity measure depends on the frequency of QAPairs found in the
answered list and in the cases, thus the design of both QAPairs set and the problem definition in the cases, highly affects the retrieval results. The design methodology that we used, first considers the different services that might be available (that is the possible solutions), and then identifies the QAPairs’ descriptions that can be used to lead the requester to find particular solutions.

Another issue to consider is related to the fact that though QAPairs are associated with ontology concepts, no considerations about this fact were taken and we have neither exploited this in the similarity measure computation. Rather, we only based the QAPairs’ frequency values on an exact matching. We envision that given more time to do further research, this issue would have been improved.

### 8.1.2 Comparison between the CCBR Similarity Measures

In this section we will focus on the comparison between the two similarity-techniques used in PreDiCtS. We make use of both the travelling domain and the medical domain; we first list the information about the sets of QAPairs, cases and services that we used to carry out these tests.

<table>
<thead>
<tr>
<th>QID</th>
<th>Description</th>
<th>Answers Available</th>
<th>Main Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Problem is a Travelling Problem?</td>
<td>Yes</td>
<td>#TravellingProblem</td>
</tr>
<tr>
<td>2</td>
<td>Do you want to attend a Conference?</td>
<td>Yes</td>
<td>#AttendConference</td>
</tr>
<tr>
<td>3</td>
<td>Do you want to use an airplane?</td>
<td>Yes</td>
<td>#Airplane</td>
</tr>
<tr>
<td>4</td>
<td>Do you need transportation?</td>
<td>Yes</td>
<td>#Transportation</td>
</tr>
<tr>
<td>5</td>
<td>Do you want to use a train?</td>
<td>Yes</td>
<td>#Train</td>
</tr>
<tr>
<td>6</td>
<td>Do you want to stay in a hotel?</td>
<td>Yes</td>
<td>#Hotel</td>
</tr>
<tr>
<td>7</td>
<td>Do you want accommodation?</td>
<td>Yes</td>
<td>#Accommodation</td>
</tr>
<tr>
<td>8</td>
<td>Do you want to stay in a hostel?</td>
<td>Yes</td>
<td>#Hostel</td>
</tr>
<tr>
<td>9</td>
<td>Is Conference registration required?</td>
<td>Yes</td>
<td>#Conference</td>
</tr>
<tr>
<td>10</td>
<td>Do you have some problem?</td>
<td>Yes</td>
<td>#Problem</td>
</tr>
</tbody>
</table>

**Table 6: QAPairs for the Travelling Problem**

In Table 6 we list a set of QAPairs relevant to this travelling domain. When building this set of QAPairs we considered the abstract relations between the QAPair’s description and the associated ontological concepts. The process through which this
association was created has been dealt with in previous chapters. Through these relations PreDiCtS identifies which question to ask next during the conversation.

A list of services that we considered as relevant to this domain have been mentioned in Chapter 5 section 5.3.1, therefore we will just refer to this list here. The cases that make use of both the QAPairs and solutions based on this list of services are defined in Table 7 below.

<table>
<thead>
<tr>
<th>Case_ID</th>
<th>Solution</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID1</td>
<td>Conference + Hotel + Train</td>
<td>QID 5, QID6, QID9</td>
</tr>
<tr>
<td>CID2</td>
<td>Conference + Hotel + Airplane + Train</td>
<td>QID3, QID5, QID6, QID9</td>
</tr>
<tr>
<td>CID3</td>
<td>Conference + Hotel + Airplane</td>
<td>QID3, QID6, QID9</td>
</tr>
<tr>
<td>CID4</td>
<td>Conference + Hostel + Train</td>
<td>QID5, QID8, QID9</td>
</tr>
<tr>
<td>CID5</td>
<td>Conference + Hostel + Airplane + Train</td>
<td>QID3, QID5, QID8, QID9</td>
</tr>
<tr>
<td>CID6</td>
<td>Conference + Hostel + Airplane</td>
<td>QID3, QID8, QID9</td>
</tr>
<tr>
<td>CID7</td>
<td>Conference + Hostel + Airplane + Hotel + Train</td>
<td>QID3, QID5, QID6, QID8, QID9</td>
</tr>
<tr>
<td>CID8</td>
<td>Conference + Hotel</td>
<td>QID6, QID9</td>
</tr>
<tr>
<td>CID9</td>
<td>Conference + Hostel</td>
<td>QID8, QID9</td>
</tr>
</tbody>
</table>

Table 7: Cases related to the Travelling domain

We now consider how both the similarity measures behaved for the same set of problems.

**Problem 1:** The person (requester) wants to register for a conference, travel by train and stay in a hotel.

**Default CCBR**

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID5</td>
<td>CID1</td>
<td>1.0</td>
</tr>
<tr>
<td>QID6</td>
<td>CID2</td>
<td>0.875</td>
</tr>
<tr>
<td>QID9</td>
<td>CID7</td>
<td>0.8</td>
</tr>
</tbody>
</table>
As expected the default similarity algorithm correctly identifies CID1, which matched perfectly with the requester’s query. CID2 is less accurate than CID1 since its problem definition contains QID3 as well and this is not part of the requester’s query. Nevertheless the retriever presents this QAPair to the user during the conversation.

**Taxonomic CCBR**

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID5</td>
<td>CID1</td>
<td>1.0</td>
</tr>
<tr>
<td>QID6</td>
<td>CID2</td>
<td>0.75</td>
</tr>
<tr>
<td>QID9</td>
<td>CID8</td>
<td>0.67</td>
</tr>
</tbody>
</table>

This is similar to the Default CCBR result and the MSC is correctly identified to be CID1. Though we note that:

(i) the retriever does not provide further questions for the requester to answer
(ii) the similarity values for the next two retrieved cases are different and CID8 is reputed to be more similar than CID7.

The reason for (i) is that the questions answered by the requester are all considered as leaves in the QAPair taxonomy (travellingQAP.owl) and therefore have no sub-QAPairs to present to the requester. In the case of (ii), CID8 is considered to be more similar than CID7 since the number of taxonomies used in the former is less than that in the latter (3 former, 5 latter).

**Problem 2:** The person (requester) wants to attend for a conference, requires accommodation and transportation.

**Default CCBR**

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID4</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>QID7</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>QID9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this case the retriever does not find any suitable cases or any other questions to present to the requester for the next cycle. This is because most of the answered questions, though present in the QAPairs’ taxonomy are not present in any of the cases
(this is due to the taxonomic aspect of the QAPairs used here) and this resulted in a set of case-similarity values which are below the threshold value (set at 0.4). If we compare this list of QAPairs with those in the problem definition of the cases, we find that only QID9 is common. In fact if this threshold value is decreased to 0.35 a number of cases and QAPairs will be returned as in the table below:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID4</td>
<td>CID7</td>
<td>0.4</td>
</tr>
<tr>
<td>QID7</td>
<td>CID2</td>
<td>0.375</td>
</tr>
<tr>
<td>QID9</td>
<td>CID5</td>
<td>0.375</td>
</tr>
</tbody>
</table>

QAPairs QID3, QID5, QID6, QID8 are returned as a result of the conversation generation algorithm.

**Taxonomic CCBR**

The TCCBR similarity on the other hand presents the following situation when QID4, QID7 and QID9 are answered:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID4</td>
<td>CID1</td>
<td>1.0</td>
</tr>
<tr>
<td>QID7</td>
<td>CID2</td>
<td>1.0</td>
</tr>
<tr>
<td>QID9</td>
<td>CID3</td>
<td>1.0</td>
</tr>
</tbody>
</table>

This is due to the fact that QID4, QID7 and QID9 are high up in the QAPairs taxonomy and therefore represent a very generic query which fits the problem definition of a large number of cases. Therefore further filtering is required. In fact the conversation generation algorithm presents questions QID3, QID5, QID6 and QID8 to the requester. If for example, the requester chooses to answer QID5 and QID6 the result is the following:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID4</td>
<td>CID1</td>
<td>1.0</td>
</tr>
<tr>
<td>QID7</td>
<td>CID2</td>
<td>0.75</td>
</tr>
<tr>
<td>QID9</td>
<td>CID8</td>
<td>0.67</td>
</tr>
<tr>
<td>QID3</td>
<td>CID8</td>
<td></td>
</tr>
<tr>
<td>QID6</td>
<td>CID6</td>
<td></td>
</tr>
</tbody>
</table>
This indicates that the filtering mechanism based on the taxonomic relations between QAPairs is working correctly.

**Problem 3:** The person (requester) wants to travel by airplane for a conference and requires accommodation.

**Default CCBR**

The following result is initially obtained when the threshold is $> 0.35$:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID7</td>
<td>0.4</td>
</tr>
<tr>
<td>QID3</td>
<td>CID2</td>
<td>0.375</td>
</tr>
<tr>
<td>QID7</td>
<td>CID5</td>
<td>0.375</td>
</tr>
</tbody>
</table>

The questions QID5, QID6, QID8, QID9 are presented for the next cycle. Thus if the requester chooses to be more specific by answering QID8 and QID9, the situation is improved as follows:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID5</td>
<td>0.625</td>
</tr>
<tr>
<td>QID3</td>
<td>CID7</td>
<td>0.6</td>
</tr>
<tr>
<td>QID7</td>
<td>CID2</td>
<td>0.375</td>
</tr>
<tr>
<td>QID8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QID9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The retriever continues to push on the conversation until no further questions are available.

**Taxonomic CCBR**

When the questions QID2, QID3 and QID7 were answered, this produced the following situation: since QID2 is parent to both QID3 and QID7, then according to the Taxonomic theory, these last two replaced QID2 in the requester’s query and the process was considered as if QID3 and QID7 have been asked. The following result was obtained:
The conversation-generation algorithm then presented QID6 and QID8, since QID3 is already a leaf and therefore there are no more sub-questions along that branch. Choosing any or both of these questions refined the number of cases and their similarity values. This is an indication that this technique presents only those questions which are relevant to the answered questions.

**Problem 4:** The person (requester) wants to attend for a conference, has to travel by airplane and train and stay at a hotel.

**Default CCBR**

Assuming that the requester answers only QID2 initially the following results were obtained:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID7</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>CID2</td>
<td>0.375</td>
</tr>
<tr>
<td></td>
<td>CID5</td>
<td>0.375</td>
</tr>
</tbody>
</table>

The questions generated for the next cycle included QID3, QID5, QID6, QID8 and QID9. When the requester answered, QID3, 5, 6 and 9, the result is as below:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID2</td>
<td>0.875</td>
</tr>
<tr>
<td>QID3</td>
<td>CID7</td>
<td>0.8</td>
</tr>
<tr>
<td>QID5</td>
<td>CID7</td>
<td>0.625</td>
</tr>
<tr>
<td>QID6</td>
<td>CID5</td>
<td></td>
</tr>
<tr>
<td>QID9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CID2 is considered as the MSC. Though the fact that the requester has initially answered QID2, this has a slight negative effect on the overall similarity of the MSC.
Performing the same requests using TCCBR presented the following initial result:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID1</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>CID2</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>CID3</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The reason for this result has already been explained as being dependent on the generic aspect of the answered question. Assuming that the requester answered, QID (3, 5, 6 and 9) as before, this produced the following result:

<table>
<thead>
<tr>
<th>Answered Questions</th>
<th>Cases</th>
<th>Similarity [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>QID2</td>
<td>CID2</td>
<td>1.0</td>
</tr>
<tr>
<td>QID3</td>
<td>CID7</td>
<td>0.8</td>
</tr>
<tr>
<td>QID5</td>
<td>CID1</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>QID6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QID9</td>
<td></td>
</tr>
</tbody>
</table>

This means that the contribution of QID2 (which is parent to the other QAPairs) to the final similarity value calculation was not considered and this made the result more accurate.

**Concluding Remarks**

Both similarity measures returned similar results and the main differences were in:

1. the number of questions that were presented to the requester by the conversation-generation algorithm, by each technique, at the end of each retrieval cycle.
2. the accuracy of the similarity values.
3. the effect of leading the requester towards the MSC.

The reason behind the first point is that Taxonomic CCBR considers the abstract relations between QAPairs and thus limits the number of redundant questions to present to the user. This though, does not come without an initial effort on behalf of the case
base designer. In fact time has to be dedicated to create suitable QAPairs taxonomies that reflect a particular problem domain and then to include these in the appropriate cases.

The second and third points are both an extended result of the above. The former is due to the fact that redundant QAPairs are not considered in the case-similarity computation, which gives a more accurate figure at the end. The latter is the end result of the taxonomic aspect when designing the QAPairs set. In fact this leading effect to solution-finding in more pronounced in the Taxonomic CCBR then in the Default CCBR. Though again this is highly dependent on the design aspect of the case base.

8.2 Suitability of the PreDiCtS Solution

The results above showed that the retrieval process is highly dependent on the design of the case base, and how much it reflects the problems that the requester wants to solve. In most reviewed papers it is in fact suggested that case design is best left for the knowledgeable expert. PreDiCtS adopts this philosophy and the included design tools are best used by someone who is quite familiar with the domain and the problems that the designed cases can solve.

In this section we will evaluate the case-design tool. In particular we will consider how effective this is in designing QAPairs and solutions.

8.2.1 QAPair design

The implementation of the QAPair design tool has been discussed at length in a previous chapter. This tool allows the designer to integrate a question description and an answer into a QAPair node.

The structure of a QAPair is defined in the CCBROnto ontology and this also provides a means for the designer to create a richer QAPair by associating concepts from a domain ontology with either question and answer or with both. In PreDiCtS this association (or annotation) process is fundamental when it comes to Taxonomic CCBR, because the taxonomic aspect of these concepts within the ontology is considered and this allows the similarity and conversation-generation algorithm to identify which cases and questions to present to the requester. Nevertheless this is a time consuming process and the designer has to make sure that a question is not associated with different
concepts and that the same concept is not associated with more than one question (this is also a requirement of the taxonomic theory defined by Gupta [28]).

To effectively create a set of QAPairs for a particular domain the designer has to first look at the generic domain ontology and then identify the concepts that might be important to include in the problem-domain. Then based on this list of concepts a new ontology is created. This will be consulted during the QAPair design where the actual question-description/concept association is performed. This methodology is a bit too restrictive and will surely be reconsidered in the future.

The issue here is basically how to generate a taxonomy of QAPairs that reflects the problem-definition. Generic domains are not problem oriented, as their name implies. Therefore we envision problem-specific ontologies to be created and shared over the Web. This will facilitate the QAPair design process but will introduce other issues such as trust and security.

The context component in each Case is precisely defined for this purpose, though it requires further work. The idea is that given a set of cases built by some third party designer, these are evaluated in a similar manner as that of recommender systems. Through this technique, service requesters are encouraged to give feedback on the suitability of a particular service or website in solving their specific problems.

The UI for the QAPair design tool is a bit too simplistic at present, since the association is provided through a drag-n-drop process and a tree structure representation of concepts. It would be interesting to use other visualisation techniques such as that presented by Henze [31] for the personal reader application.

8.2.2 Solution Representation

The solution representation in a PreDiCtS case is also based on the CCBROnto ontology, whereby an Action is represented by an OWL-S template (though there is no restriction on the templates’ definition language).

The visual composer tool, in the case creator, provides for a mapping process which is based on work done by Scicluna [59] and which has been dealt with in previous chapters. It allows for the generation of an OWL-S process description (or template) from a UML Activity diagram; though we have only implemented the most straightforward UML components that generate the most basic of OWL-S control constructs.
Nevertheless this was considered as enough for the scope of this prototype. Different solutions could be created which allowed PreDiCtS to be tested using different similarity approaches.

The inputs and outputs for each service are automatically defined during the generation process. At present we only consider services with a single input and output. Moreover these inputs and outputs are not yet being associated with inputs and outputs defined in some service domain ontology. When this association is in place, the template could be used as part of the initial query to a service registry where the actual service binding process will be done.

Another issue to consider is that there is no way, at present, in which constraints over the different services can be added to a template. In OWL-S these are defined through SWRL rules. We envision that a repository of such rules will also be available and these could be imported and used as required (taking into consideration trust issues).

8.3 Summary

In this chapter we performed a number of tests with the different similarity measures that are provided by PreDiCtS. We considered the results and noted that they were satisfactory within the limitations of the prototype and the small number, but different, sets of QAPairs and cases available.

We also discussed the effectiveness of the solution adopted in PreDiCtS, which uses the CCBROnto ontology to define the cases’ structure. We also considered the suitability of the case creator tool to design QAPairs and solutions, highlighting shortcomings and possible improvements.
Chapter 9

Future Work and Conclusion

In the previous chapter we have seen that the results of the tests we conducted with different sets of QAPairs and cases have been satisfactory. In the following sections we will first consider how the work presented in this thesis can be extended and improved. We end this chapter with some final statements as conclusion to this thesis.

9.1 Adaptation mechanism

The addition of this component to PreDiCtS completes the CCBR cycle. Adaptation is closely related to the retrieval process, since it can be considered as the personalisation of a retrieved case to suit more effectively the requester’s needs.

In Chapter 6 we discussed some of the design issues for this component, highlighting the decomposition of a case into its basic components and then the ability to adapt each component separately. An adapted case will then be added to the case base and tested to ascertain its effect on the case base, after which it might be re-adapted or retained in the case base.
The adaptation of cases can be considered as a personalisation process through which the requester or designer can change aspects of a case to provide a more suitable problem-solution relation.

The possible changes can include:

1. the addition and removal of services from the template definition (or solution)
2. editing of input/output for particular services, including changes to associated ontological concepts
3. changes in the order of execution of services or changes to the control constructs used
4. adding or removing QAPairs that make up the problem definition. Editing QAPairs is a more complex process and might lead to a negative effect on the case base. So great care has to be taken in this case.

The most important aspect of this adaptation component will be the UI that provides the visualisation of all subcomponents of the case requiring adaptation. A mapping from the template definition to UML activity diagrams is required. This will provide an easy way to adapt the solution. This component will be accessed also from within the retrieval component, where it would be possible to adapt the MSC to obtain a higher similarity to the problem at hand.

9.2 Integration phase

This aspect of PreDiCtS has been briefly mentioned in Chapter 1. We defined this as being the next logical step towards the actual discovery and composition of services. The idea is to take the MSC or its adapted copy and present it to an AI planner that will use the relevant knowledge to query a matchmaker.

The PreDiCtS’ process has some advantages over other composition techniques mentioned in previous chapters, since the user’s involvement, through the CCBR process, identifies more clearly what the user is requesting. As discussed, various other initiatives have taken the direction of using AI planning, albeit to describe a process which leaves the requester out of the loop during the service retrieval process, assuming that complete information is available to the planner. On the other hand we want the requester to be in control and to specify as clearly as possible what his needs are. Thus while certain initiatives, that are based on OWL-S, present a query to the matchmaker,
which is based solely on the profile, we allow the query to be based over a generalised profile definition (which is linked to a profile hierarchy ontology) and we also provide some process model knowledge which can be used by the planner to effectively create a successful stream of services that satisfy the requester’s needs.

The implementation of this functionality requires the use of a suitable AI planning technique. In Chapter 3 we discussed different AI planning algorithms, but we think that POP could be used in this situation as described by Peer [54]. We envision that a mapping component that takes as input an OWL-S description and transforms this into the planner’s plan definition language is required. For this purpose we will make use of a third-party component, called OWLS2PDDL [51], that takes as input an OWL-S description and transforms this into an XML-like representation of PDDL 2.1 called PDDXML.

PDDL is a FOL language with lisp-like syntax, and PDDXML is a dialect that uses XML syntax for PDDL expression, which makes it easier to use. Essentially, the owls2pddl converter creates the PDDL problem description (in PDDXML syntax) by using the following conversions: (this is taken from the owls2pddl Readme file)

1. **OWL-S properties -> PDDL predicates**;
2. **OWL-S types -> PDDL types**;
3. **OWL-S objects -> PDDL objects**;
4. **OWL-S resources and datatypes -> instantiated PDDL predicates for the initial and goal state**;

Given that services have been associated to the actions, in the planning domain, it is straight forward to take the generated PDDXML file and the domain and problem definitions and present these as input to an AI planner. Since the initial OWL-S definition was incomplete, the planner has to be able to work with partial knowledge, thus POP will be ideal in this situation. Such a planner will then try to produce a concrete OWL-S service that can eventually be executed.

### 9.3 Feedback Mechanism

In previous chapters we have more than once discussed the importance of having a feedback mechanism included in PreDiCtS. It is a process that strives to maximise the usefulness of the case base.
We have identified that such a mechanism can help:

i. the requesters to identify how a case has been used, by whom (here we refer to a process which clusters users not the actual person) and whether it has been reputed useful.

ii. the designer when importing cases from third-parties, since, based on the reputation of cases, then he can decide whether to import or not further cases from this source.

iii. the designer when performing maintenance on the case base; those cases that have a negative reputation may be removed from the case base.

Such reputation mechanism may be based on user feedback such as that discussed in Weber [74] and in Hefke [29]. It is similar to the feedback mechanism of recommendation systems and considers the overall feedback given by case users to generate a reputation score. This score represents the usefulness of a particular case to solve a specific problem. For this reason we have included in CCBROnto a way to structure this information as part of the case history. We have also provided the means by which a trust value can be computed for the case creator. This trust value will be based on the global reputations of the cases provided by that particular case supplier. If the overall case-reputation is less than a certain threshold then this implies a lower trust level and that source is not used any more. In a way this is similar to how eBay [21] reputes its sellers and buyers, though PreDiCtS will take action and remove this source from the list of possible case-suppliers.

9.4 Context Awareness

As explained above, the motivation behind the Context component in each Case definition is to capture context information about the case creator. During case retrieval this information is used to identify the suitability and reputation of a case in solving a particular problem.

We would like to extend this concept in PreDiCtS to consider also the context of the requester, in the form of user preferences and service constraints. These user preferences will be used to filter the list of returned cases to a finer granularity. They will also be important when case adaptation is required. On the other hand through the
defined constraints a more realistic service composition may be created during the Integration phase.

9.5 Graph-Based Technique

In PreDiCtS we have implemented two similarity techniques to test the suitability of our approach towards providing a personalisation of service composition. Through the use of the Taxonomic CCBR similarity technique we have exploited only a fraction of the advantages that ontologies may present when calculating similarities.

Ontologies provide many other relations, apart from subsumption, which can be used to capture similarities based on semantics. For this reason we can look into graph-based similarities since OWL ontologies can be considered as directed-graphs. Thus the case similarity process could take the form of ontology-instance similarity.

Such techniques are becoming increasingly more popular in work related the Semantic Web. We can refer to work done by Gu [4], Bernstein [13] and Hefke [30] on creating a similarity technique for ontologies. Apart from the well known taxonomic relations these techniques consider also other non-taxonomic relations. For example, Hefke [30] discusses Set-similarity (which computes similarity between sets of instances by using vector representations and cosine similarity) and Relational-similarity (which considers the incoming and outgoing relations and assumes that instances are more similar if they have relations to similar entities). In Maedche [43] we find references to a Taxonomic similarity based on what is referred to as Upward Cotopy, which was used in the development of the SEAL portal Maedche [42]. Further more Henze [31] has adopted a similarity distance calculation to compute concept similarities for the PersonalReader application.

Further research is required to identify the requirements and eventually implement and include this similarity component within PreDiCtS. We agree that this type of similarity measures represent the way forward and can also be useful in areas such as those of ontology matching and alignment.

9.6 Conclusion

The goal of this thesis was to research the problem behind the reuse of past composition knowledge. We identified a number of motivating points including the higher level of
abstraction that has to be considered to maintain flexibility and ease adaptation of these compositions or templates (as we refer to them). Furthermore we highlighted the importance of identifying more clearly what the requester’s needs are, as regards service composition.

To tackle these problems we started by looking in detail at a number of service definition languages and service composition techniques based on AI planning. This knowledge was considered as the basis of our research. It provided the necessary background to identify how we can create flexible and adaptable templates, and also which technique is most suitable when it comes to identifying more clearly the requester’s needs.

We chose to adopt CCBR as the base of our template creation and retrieval process. We have discussed in depth the details of this technology and the reasons behind the need to create a CCBR ontology that defines the structure of each CCBR case instance. The design and implementation of this CCBR onto ontology were detailed. We also considered a number of CCBR similarity measures that could be adopted through which we wanted to experiment the potential of the CCBR technology. We therefore required a prototype to be able to show that this approach can be a suitable solution.

The PreDiCtS prototype was designed and implemented. To effectively test this prototype we used a number of scenarios for which we designed and created a number of domain and problem ontologies. We also used different CCBR similarity measures to identify which one is the most suitable. One of these measures was based on a simple frequency measure while the other considered the taxonomic relations between the QAPairs.

The results of these tests were considered satisfactory even though only a small number of cases were available. We eventually hope to have a larger set of cases related to other domains on which further testing can be performed.

Through the tests that we conducted it was shown that the design of both the case base and QAPairs affects the retrieval process and to some extend, this also depended on the similarity measure. The most important difference between these two similarity measures was in fact the number of relevant questions that the taxonomic similarity measure presented vis-à-vis the frequency based similarity measure during the conversation.
Chapter 9   Future Work

Through these tests it was also shown that the way the solution was designed and encapsulated within the cases, was effective though care has to be taken when associating QAPairs (as parts of the problem definition) with these solutions.

We finally discussed which aspects of the prototype can be improved or extended and which future directions we could consider to further exploit the ideas presented in this thesis.

After considering the attained results, we can say that we have successfully reached the targets that we proposed at the start of this thesis. We have shown that the CCBR-approach adopted in PreDiCtS has important advantages over present service composition techniques, since the mixed-initiative process provided both a process through which the requester can express more clearly his needs and also an effective way in which past templates can be reused.

It will be interesting to see how PreDiCtS will continue to develop. Meanwhile we hope that the work presented in this thesis provides an initial step towards the adoption of such mixed-initiative processes in the personalisation of the discovery and composition of Web services.
Appendix A

CCBR Ontology for Reusable Service Templates

Charlie Abela, Matthew Montebello
Department of Computer Science and AI
University of Malta
{charlie.abela, matthew.montebello}@um.edu.mt

ABSTRACT

We present the motivation and design of CCBROnto, an OWL Ontology for Conversational Case-Base Reasoning (CCBR). We use this ontology to define cases that can eventually be stored, retrieved and reused by a mixed-initiative approach based on CCBR. We apply this technique for retrieving Web Service Composition templates.

Categories and Subject Descriptors
I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods

General Terms
Design, Implementation

Keywords
CCBR, Ontologies, OWL, Web Services

1. Introduction

Web Services composition is usually interpreted as the integration of a number of services into a new workflow or process. A number of compositional techniques have been researched [9,10] that attempt to address service composition by composing web services from scratch while ignoring reuse or adaptation of existing compositions or parts of compositions. Furthermore composing web services by means of concrete service interfaces leads to tightly-coupled compositions in which each service involved in the chain is tied to a web service instance. This approach may lead to changes in the underlying workflow which range from slight modifications of bindings to whole redesigning of parts of the workflow description. Therefore we interpret services at an abstract level to facilitate their independent

---

1This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Program project REWERSE number 506779

Accepted in the Demos and Posters session of the 3rd European Semantic Web Conference (ESWC 2006), Budva, Montenegro, 11th - 14th June, 2006
composition. In fact our approach is more similar to [8,11,12], which use pre-stored abstract workflow definitions or templates in their composition framework. Abstract workflows allow for more generalisations and a higher level of reusability [5]. The use of such templates can be thought of as a pre-processing stage towards service discovery and composition, whereby abstractly defined workflow knowledge can be concretely bound to actual services that satisfy a template. To make effective reuse of such templates we have considered CCBR [6]. This extends from CBR and allows for partial definition of the problem by using a mixed-initiative refinement process to identify more clearly the user’s problem state.

2. Related Work

In recent work relating CBR to the Semantic Web [2, 4], we find the definition of two ontologies, CaseML and CBROnto. These are both defined for CBR rather than CCBR and thus do not define concepts related to question-answer (QA) pairs, which are at the core of the CCBR process. Nonetheless we considered these when we designed and implemented our OWL-based ontology, which we call CCBROnto (this has no relation to CBROnto). We make use of this ontology within our personalised service discovery and composition framework (PreDiCtS) to define cases of best practice composition knowledge. In what follows we make explanatory references to this ongoing work.

3. CCBROnto

In CCBROnto the basic components of a Case are defined by the CaseContext, Problem and Solution classes. This structure is motivated by the underlying methodology used in PreDiCtS. In this framework we adapt the CCBR approach to help the user refine his query for a particular service request. The problem description is defined by a set of discriminating QA pairs, which characterize a particular solution. On the other hand, the solution is a place holder for a reusable service composition template which is a container of best practice knowledge about composition of generic service components. In the following sections we will explain in more detail the basic Case components and illustrate by means of an example how such a case is defined.

4. Context

In [3], the term context is defined as “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”.

We fully agree with this definition and in the CaseContext we have included knowledge related to the case creator, case history, and case provenance. We have also considered ideas presented in [7] and [1] which discuss the importance of context in relation to Web Services. In PreDiCtS context knowledge
helps to identify, (i) why a case was created and by whom, (ii) certain aspects of case usage and (iii) the case relevance to problem solving. The CaseCreator includes a reference to the Role description, that the creator associates himself with, together with a foaf:Person instance-definition that describes who this person is. The motivation behind using foaf is to keep track of reputation knowledge which could be used to reliably share cases between PreDiCtS users.

The CaseContext also provides a place holder for CaseHistory, which becomes important when it comes to case ranking and usage, since it allows users to identify the relevance and usefulness of a case in solving a particular problem. It is also important for the case administrator when case maintenance is performed. Cases whose history indicates negative feedback may be removed from the case base. Case Provenance is also used in conjunction with reputation issues, since it associates a case with a URL indicating the case-origin.

5. Problem

The Problem state description in a PreDiCtS case is based on the taxonomic theory of [6]. Every problem is described by a list of QA pairs rather than a bag. This is required since QA pairs have to be ranked when they are presented to the user. Each QA pair consists of a CategoryName, a Question and an Answer. Since the taxonomic theory requires that QA pairs are defined in a taxonomy during the case creation stage, each question description is associated, through the property hasConcept, with a Statement resource which is a reification of a triple defined in the domain of discourse. This relation is not intended to fully capture the natural semantics of the QAs, rather it is important when calculating similarities.

A typical QA pair example from the traveling domain might include the question, “What type of transportation? This is related, by means of the hasConcept property, to the concept Transportation, which is defined in the Traveling domain. On the other hand, we assume that Answers could have either a binary or nominal value and are respectively defined in the ontology by the YesNoAnswer and ConceptAnswer classes. The former points to the binary literals, while the latter is used to represent answers that are associated to a concept in a domain ontology through the previously mentioned hasConcept property.
6. Solution

The solution in PreDiCtS provides a hook where composition templates can be inserted. Each Solution is defined to be an Action which has a description and isDefinedBy an AbstractTemplate. A template can be sub-classed by any service composition description, such as that defined by OWL-S (see figure 16), though as explained earlier any process definition language can be used (e.g. WS-BPEL).

7. Conclusion

Through the use of CCBROnto we are able to define cases whose solutions are composition templates. This allows our PreDiCtS framework to retrieve such templates by consulting the user in every stage and presenting her with the most suitable composition knowledge available to choose from. The user can then decide whether to reuse as is, or possibly adapt this to fit her personal needs.
References


Appendix B

PreDiCtS: A Personalised Service Discovery and Composition Framework
Charlie Abela, Matthew Montebello

Department of Computer Science and AI
University of Malta
{charlie.abela, matthew.montebello}@um.edu.mt

Abstract. The proliferation of Web Services is fostering the need for applications to provide more personalisation during the service discovery and composition phases. An application has to cater for different types of users and seamlessly provide suitably understandable and refined replies. In this paper, we describe the motivating details behind PreDiCtS, a framework for personalised service discovery and composition. The underlying concept behind PreDiCtS is that, similar service composition problems could be tackled in a similar manner by reusing past composition best practices. These have to be useful and at the same time flexible enough to allow for adaptations to new problems. For this reason we are opting to use template-based composition information. PreDiCtS’s retrieval and refinement technique is based on conversational case-based reasoning (CCBR) and makes use of a core OWL ontology called CCBROnto for case representations.

Keywords: CCBR, Ontologies, Semantic Web, Web services

1. Introduction

Reusability and interoperability are at the core of the Web Services paradigm. This technology promises seamlessly interoperable and reusable Web components that facilitate rapid application development and integration. When referring to composition, this is usually interpreted as the integration of a number of services into a new workflow or process. A number of compositional techniques have been researched ranging from both, manual and semi-automatic solutions through the use of graphical authoring tools [18], [19], to automated solutions based on techniques such as AI planning [17] [20] and others.

The problem with most of the composition techniques mentioned above is three fold (i) such approaches attempt to address service composition by composing web services from scratch, ignoring reuse or adaptation of existing compositions or parts of compositions, (ii) it is assumed that the requester
knows exactly what he wants and how to obtain it and (iii) composing web services by means of concrete service interfaces leads to tightly-coupled compositions in which each service involved in the chain is tied to a web service instance. Using this approach for service reuse, may lead to changes in the underlying workflow which range from slight modifications of the bindings to whole re-designing of parts of the workflow description. Therefore in our opinion, services should be interpreted at an abstract level to facilitate their independent composition. [10] adds, “abstract workflows capture a layer of process description that abstracts away from the task and behaviour of concrete workflows”, and this allows for more generalisation and a higher level of reusability. A system can start by considering such abstractly defined workflow knowledge and work towards a concrete binding with actual services that satisfy the workflow.

To make effective reuse of such abstract workflow definitions one could consider CBR that is amenable for storing, reusing and adapting past experience for current problems. Nevertheless CBR restricts the user to define a complete problem definition at the start of the case-retrieval process. Therefore a mixed-initiative technique such as CCBR [3] is more appropriate since it allows for a partial definition of the problem by the user, and makes use of a refinement process to identify more clearly the user’s problem state.

In summary we have identified the following motivating points:

1. Reusability of compositions has the advantage of not starting from scratch whenever a new functionality is required.
2. For effective reusability a higher level of abstraction has to be considered, which generalises service concepts and is not bound to specific service instances.
3. Personalisation of compositions can be achieved by first identifying more clearly the user’s needs and then allowing for reuse and adaptation of past compositions based on these needs prior to binding with actual services.

The goal of this work is to present, the motivation behind, and prototype of PreDiCtS, a framework which allows for personalisation of service discovery and composition through the reuse of past composition knowledge. One could say that we are trying to encode and store common practices of compositions which could then be retrieved, reused and adapted through a personalisation technique. The solution we propose in PreDiCtS has two phases.

For the first phase, which we call the Similarity Phase, we have adopted a mixed-initiative technique based on CCBR. This provides for the personalisation process. Given a new problem or service composition request, this approach allows first to retrieve a ranked list of past, similar situations which are then ranked and suggested to the requester. Through a dialogue process the requester can decide when to stop this iterative-filtering phase, and whether to reuse or adapt a chosen case. Case definition is through an OWL-based ontology which we call CCBROnto [2] and which provides for the description of context, problem and solution knowledge. At present PreDiCtS allows for case creation and retrieval (adaptation is in the pipeline) and once a case (or set of cases) is retrieved, it can be presented to the next phase, which we call the Integration Phase where a mapping is attempted, from the features found in the
chosen solution, to actual services found in a service registry. Due to space restrictions this is dealt with in a future paper.

The rest of this paper is organized as follows. In Section 2 we will give some brief background information on CCBR. Then in Section 3 we will give an overview of the OWL case ontology, CCBROnto. In Section 4 we will present the architecture of PreDiCtS and some implementation details mainly focusing on the case-creator and case-retriever components. After which we present the last section with future work and concluding remarks.

2. Conversational Case-Based Reasoning

Case-Based Reasoning is an artificial intelligence technique that allows for the reuse of past experience to solve new problems. The CBR process requires the user to provide a well-defined problem description from the onset of the process. But users usually cannot define their problem clearly and accurately at this stage. On the other hand, CCBR allows for the problem state to be only partially defined at the start of the retrieval process. Eventually the process allows more detail about the user’s needs to be captured by presenting a set of discriminative and ranked questions automatically. Depending on the user’s supplied answers, cases are filtered out and incrementally the problem state is refined. With each stage of this problem refinement process, the system presents the most relevant solutions associated to the problem. In this way the user is kept in control of the direction that this problem analysis process is taking while at the same time she is presented with solutions that could solve the initial problem. If no exact solution exists, the most suitable one is presented and the user is allowed to adapt this to fit her new requirements. Nevertheless, this adaptation process necessitates considerable domain knowledge as explained in [4], and is best left for experts.

One issue with CCBR is the number of questions that the system presents to the user at every stage of the case retrieval process. This issue was tackled by [11] which defined question-answer pairs in a taxonomy and by [1] through the use of knowledge-intensive similarity metrics. In PreDiCtS we have adapted the former method\(^4\) since a QA pairs taxonomy is defined to be an acyclic directed graph in which nodes are related to other nodes through parent-child relations and it is assumed that a node subsumes all its descendent nodes. This is very similar to how classes in OWL are related via the subClassOf relation and this fits well with the underlying case structure that we use in PreDiCtS.

3. CCBROnto

CCBROnto is an important component of PreDiCtS since it provides for (i) case and question-answer pair definitions, and (ii) the association of domain and case-specific knowledge. In CCBROnto the topmost concept is a Case. Its basic components are defined by the CaseContext, Problem and Solution classes. In [8] context is defined as “any information that can be used to characterize the situation of an entity. An

\(^4\) Whenever we refer to this taxonomic theory we will be referring the work done by Gupta
entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”. We fully agree with this definition and in the CaseContext, we have included knowledge related to the case creator, case history, ranking and case provenance. We have considered ideas presented in [6], [7] and [15] which discuss the importance of context in relation to Web Services and stresses on the importance of the use of context in CBR, especially when cases require adaptation. Such context knowledge makes it possible to differentiate between users and thus the system could adapt cases accordingly. For example in the travelling domain, both going to a conference and going for a holiday may require similar services, such as hotel booking and flight reservation, though the use of a conference booking service is only required in the former. Thus, based on the contexts or roles of the users (a researcher the former and a tourist the latter) the CBR system can adapt the case knowledge to present cases that satisfy the requirements of both. A researcher can adapt the case for the tourist by including a suitable conference booking service.

In PreDiCtS we consider highly important such context knowledge since it helps to identify, why a case was created and by whom, together with certain aspects of case usage and its relevance to solving a particular problem. The CaseCreator provides a Role description that the creator associates himself with, together with a foaf:Person instance definition that describes who this person is. The motivation behind using foaf is to eventually be able to embed some level of reputation relevant to the person who created the case. The importance of this feature will become more visible and important when cases are shared.

The CaseContext also provides a place holder for CaseHistory. The knowledge associated with this feature is important when it comes to case ranking and usage, since it allows users to identify the relevance and usefulness of a case in solving a particular problem. It is also important for the case administrator when case maintenance is performed. Cases whose history indicates negative feedback may be removed from the case base. Case Provenance is also used in conjunction with reputation since it indicates a URL from where the case originated. Encapsulating such information in each case will help in maintaining a reliable case base.

The Problem state description in a PreDiCtS case is based on the taxonomic theory. Every problem is described by a list of QA pairs rather then a bag. This is required since QA pairs have to be ranked when they are presented to the user. Each QAPair is associated with a CategoryName, a Question and an Answer (see Fig.1). Each question has a textual description and is associated with a concept from the domain ontology through the isRelatedTo relation. We further assume that Answers could be either binary or nominal-valued. For this reason we have created two types of answer classes, YesNoAnswer and ConceptAnswer. The former is associated with a literal represented by either a Yes or a No. While the latter, requires an association with a concept in some domain ontology, through the previously mentioned isRelatedTo property. The motivation behind the use of this property is related to the taxonomic theory, which requires that QA pairs are defined in a taxonomy so that during case retrieval, the number of redundant questions presented to the requester is reduced. Thus during the case creation stage, each question and answer description is associated with an ontological concept defined in the domain of discourse. This is similar to how [1] associates ontology concepts with pre-defined questions. In PreDiCtS we want to make use of such <concept-question> association so that questions and answers are
implicitly defined in a taxonomy. This association is also important when similarities between QAPairs and between cases are calculated.

![Diagram](image)

**Fig.1:** CCBROnto Problem structure

The Solution in PreDiCtS provides a hook where composition templates can be inserted. The main goal behind such a structure is to be able to present abstract composition knowledge as solutions to the user’s request and at the same time allow for more flexibility when searching for actual services. In fact each Solution is defined to have an Action which has a description and is DefinedBy an AbstractTemplate. A template can be sub-classed by any service composition description, such as that defined by OWL-S. An OWL-S template in this case is an intersection between a service, profile and process definitions.

4. PreDiCtS: implementation issues

As explained in other sections, the PreDiCtS framework allows for the creation and retrieval of cases in its Similarity phase (see Fig. 2). The respective components that perform these two tasks are the CaseCreator and the CaseRetrieval. PreDiCtS is written in Java and is developed in Eclipse. It uses a MySQL database to store the cases and makes use of both Jena and the OWL-S APIs.

The Similarity phase is triggered by the user whenever she requires knowledge related to past compositions. In PreDiCtS the user is not expected to know exactly which type of services or service composition are required but she is required to answer a set of questions such that the system identifies more clearly what is required. Given information related to the domain, the retrieval process is initiated whereby all questions in a taxonomy relevant to that particular domain are presented to the user. Given the set of questions to choose from, the user can then decide to answer some of these questions. Depending on the answers provided, the system will try to find cases in which questions where answered in a similar manner. A similarity measure is used to rank cases. The questions which are present in the retrieved cases but which are still unanswered, yet are related to the problem, are then presented in a ranked order to the user. The process continues until the user either chooses a case which includes a suitable solution or else, in absence of such a case, decides to adapt one of the most similar cases, thus
further personalising the solution to her needs. The user can also opt to create a case from scratch to meet her requirements.

![Diagram of CCBR in PreDiCtS](image)

**Fig.2: Taxonomic CCBR in PreDiCtS (adapted from Weber03)**

In the next sections we will describe the above mentioned PreDiCtS components by referring to an example from the health domain which deals with the combination of services that are used when a patient is admitted to hospital.

### 4.1 Case Creation

The CaseCreator component allows the expert user to add a new case to the case base. A case \( c \) can be defined as \( c = (dsc, ctx, \{q_1a_1, \ldots, q_ia_j\}, act, frq) \) where;

- \( dsc \) is a textual description of the case.
- \( ctx \) represents a set of context related features, such as Role and CaseCreator information based on foaf.
- \( \{q_1a_1, \ldots, q_ia_j\} \) is a representation of the problem state by a set of question-answer pairs.
- \( act \) denotes the solution which is represented by service composition knowledge stored in an abstract template.
- \( frq \), is the frequency with which a case is reused.
The example presented in Fig. 3 represents the combination of knowledge that is required to build a new case. PreDiCtS takes into consideration both domain and composition knowledge and combines them, based on the knowledge of the creator. In the example, the case creator is a Doctor (John) who specialises in URTI (Upper Respiratory Tract Infections) and cardiovascular conditions. The case in question represents the situation where a new patient, who is more than 16 years old, has entered hospital with shortness of breath. The creator enters context information about himself and any relations that he has with other persons. In this scenario, John has work relations with Professor Mary Nice. This information provides for a level of reputation in the expertise of the creator. The composition knowledge in this case represents a number of services that the hospital system wants to use to efficiently cater for patients entering hospital. This particular functionality is required to monitor the patient from the moment that he enters the hospital until he is comfortably stationed in a room.

To add service information to a case, the creator can use a visual component which is based on UML activity diagrams, though other representations, which are more user-friendly, are being considered. Each visual representation is mapped into a process model representation. In this work we use OWL-S as the underlying language for this representation.

A service definition in OWL-S is just a place holder for information relating the profile, process and grounding. We are not considering any grounding knowledge at this stage, since this will be tackled later on in the Integration phase when actual service bindings are sought. As regards the profile, we only consider that knowledge which is relevant and which is not tied to specific providers. The profile part of the template includes the definitions of inputs and outputs, profilehierarchy and references to the process and service components. The profile hierarchy is considered to be of particular importance since it...
represents a reference to the service domain knowledge, that is, it identifies the taxonomic location of a particular set of service profiles. We think that such ontologies will become increasingly more important in relation to best practice knowledge. The template also provides information related to how a number of service components are combined together. What is most important here, are the control constructs such as Sequence, If-Then-Else, and Split that determine the order of execution of the service components. These service components are defined through the OWL-S Perform construct which associates a particular service component with another by binding its outputs to another service component’s inputs.

An important aspect of case-creation in CCBR is the addition of question-answer pairs since they are fundamental for the case retrieval process. Through PreDiCtS we allow the creator to either reuse existing QA pairs or create new ones. Textual questions are associated with concepts defined in ontologies and this provides an implicit taxonomic structure for QA pairs. Such association provides the possibility to reason about these concepts, and also to limit the number of questions to present to the user during the retrieval process. The taxonomic theory requires that each case includes the most specific QA pair from a particular taxonomy. Given the open-world assumed by ontologies on the Web, we assume that the knowledge (triples) associated with a set of QA pairs is closed by adapting the idea of a local-closed world defined by [12].

Adding a new case to the case base is mainly the job of the knowledge expert, nevertheless we envision that even the not so expert user may be able to add cases when required. For this reason we have used the same technique as that used by recommending systems and also adopted by [21], which allows case-users to give feedback on the utility of a particular case to solve a specific problem.

4.2 Case Retrieval

Similarity is based on an adaptation of the taxonomic theory, and is divided into two steps, similarity between question-answer pairs and an aggregate similarity to retrieve the most suitable cases. The prior, involves the similarity between the QA pairs chosen by the user and those found in a case. In the taxonomic theory two pairs are defined to be more similar if the one found in the case is a descendant (therefore more specific) of the other, rather then its parent (therefore more generic). Though we have adopted this similarity assessment metric, we take into consideration that each QA pair is a set of triples or rather an acyclic directed graph. Thus similarity between QA pairs is based on the similarity between two such graphs. The taxonomic similarity is calculated as follows:

\[
\text{sim} \left(C_{Q1}, C_{Q2}\right) = \begin{cases} 
1 & \text{if } C_{Q2} \subseteq C_{Q1} \\
\frac{(n+1-m)}{(n+1+m)} & \text{if } C_{Q1} \subseteq C_{Q2} \\
0 & \text{otherwise}
\end{cases}
\]

where, \(C_{Q1}\) and \(C_{Q2}\) are concepts
\(n\) = number of edges between \(C_{Q1}\) and the root i.e. the concept Thing
\(m\) = number of edges between \(C_{Q1}\) and \(C_{Q2}\)
Having calculated such similarity between QA pairs then an aggregate similarity metric is used to calculate the overall similarity between the user query $Q_U$ and a case problem description, $P_C$. This aggregate similarity is calculated as follows:

$$
\text{sim} (Q_U, P_C) = \frac{\sum_{i \in Q_U, j \in P_C} \text{sim}(C_{Q_i}, C_{Q_j})}{T}
$$

where, $T$ in the original taxonomic theory represents the number of taxonomies, here it represents the number of different ontologies that are used to define the concepts found in the QA pairs.

We are also looking at other research work which provides for similar measures, in particular work related to ontology-based similarity measures [13], [16] and semantic distance [5], [14]. Such work is important since it does not only consider the taxonomic similarity between concepts but also similarity based on the number of relations and attributes associated with the concepts.

5. Conclusion

In this paper we presented the main concepts behind PreDiCtS. The use of CCBR as a pre-process to the service discovery and composition is promising since it provides for inherent personalisation of the service request and thus as a consequence also more personalised compositions. We also presented CCBROnto as a case definition language which allows for seamless integration between CCBR and the Semantic Web, by providing reasoning capabilities about concepts within the case definitions. Nevertheless, there is still a lot to be done, especially where it comes to case generation and evaluation. A case base can only be evaluated effectively if the number of cases is large. We are in fact considering the possibility of generating cases, for experimental purposes, by extracting the required template knowledge from already available service descriptions and then adding context information and QA pairs. Other issues for future consideration include the design of the questions and the way in which they are associated with ontology concepts, the effective evaluation of the similarity metrics used with an eye on work being done on semantic similarity and also the inclusion of an adaptation component. The latter will provide for more personalisation of the solutions presented by PreDiCtS and thus also of the services that will be presented to the user.

References

2. C. Abela, CCBROnto, http://www.semantech.org/ontologies/CCBROnto.owl
9. B. Diaz-Agudo et al, On Developing a Distributed CBR Framework through Semantic Web Services, Workshop on OWL: Experiences and Directions, Galway’05
10. A. Goderis et al. Seven bottlenecks to workflow reuse and repurposing. 4th Int. Semantic Web Conference, Galway, Ireland, 6-10 Nov. 2005
11. K. Gupta, Taxonomic Conversational Case-Based Reasoning, Proceedings of the 4th International Conference on Case-Based Reasoning, 2001
12. J. Heflin, H. Muñoz-Avila, LCW-Based Agent Planning for the Semantic Web, AAAI Workshop WS-02-112002
15. Z. Maamar et al, Context for Personalised Web Services, 38th Hawaii International Conference on system Science, 2005
16. A. Maedche, V. Zacharias, Clustering Ontology-based Metadata in the Semantic Web, Joint Conferences (ECML’02) and (PKDD’02), Finland, Helsinki, 2002
17. J. Peer, A POP-based Replanning Agent for Automatic Web Service Composition ESWC’05
18. J. Scicluna et al, Visual Modelling of OWL-S Services, IADIS International Conference WWW/Internet, Madrid Spain, October 2004
19. E. Sirin et al, Semi-automatic composition of web services using semantic descriptions, in ICEIS 2003, Angers, France, April 2003
Appendix C

CCBROnto Ontology

<?xml version="1.0"?>
<rdf:RDF xml:base="http://www.semantech.org/ontologies/CCBROnto.owl"
xmlns="http://www.semantech.org/ontologies/CCBROnto.owl#"
xmld:am="http://www.daml.org/2001/03/daml+owl#"
xmld:process="http://www.daml.org/services/owl-s/1.1/Process.owl#"
xmld:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#"
xmld:service="http://www.daml.org/services/owl-s/1.1/Service.owl#"
xmld:dc="http://purl.org/dc/elements/1.1/"
xmld:list="http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#"
xmld:owl="http://www.w3.org/2002/07/owl#"
xmld:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmld:xsd="http://www.w3.org/2001/XMLSchema#"
<owl:Ontology rdf:about=""
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#"
<owl:Ontology rdf:about=""
<owl:Class rdf:ID="QAPair"
<owl:Restriction
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality
<owl:onProperty rdf:resource="#hasName"/>
<owl:Restriction
<owl:subClassOf
<owl:Restriction
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality
<owl:onProperty rdf:resource="#hasRole"/>
<owl:Restriction
<owl:Class rdf:ID="Solution"/>
<owl:Class rdf:ID="ConceptAnswer"/>
<owl:Class rdf:ID="Answer"/>
</rdfs:subClassOf>
<owl:Restriction>
<owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality
<owl:onProperty rdf:resource="#hasConcept"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:Class rdf:about="#Answer">
<owl:equivalentClass>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#YesNoAnswer"/>  
<owl:Class rdf:about="#ConceptAnswer"/>
</owl:unionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="CaseUser">
<rdfs:subClassOf rdf:resource="http://www.mindswap.org/2003/owl/foaf#Person"/>
</owl:Class>
<owl:Class rdf:ID="YesNoAnswer">
<rdfs:subClassOf rdf:resource="#Answer"/>
</owl:Class>
<owl:Class rdf:ID="CaseBase">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:FunctionalProperty rdf:ID="hasCases"/>
</owl:onProperty>
<owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality
</owl:Restriction>
</rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#HistoryEntry"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom>
<owl:Class rdf:ID="Case"/>
</owl:allValuesFrom>
<owl:onProperty>
<owl:FunctionalProperty rdf:about="#hasCases"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<owl:Class rdf:ID="CaseHistory">
<rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#List"/>
<rdfs:subClassOf>
<owl Restriction>
<owl:Property
rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#first"/>
<owl:allValuesFrom rdf:resource="#HistoryEntry"/>
<owl:Class rdf:ID="OWLSTemplate">
<owl:intersectionOf rdf:parseType="Collection">
<owl:Restriction>
<owl:onProperty rdf:resource="#hasServiceTemplate"/>
<owl:hasValue rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl#Service"/>
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasProfileTemplate"/>
<owl:hasValue rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl#Profile"/>
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasProcessTemplate"/>
<owl:hasValue rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl#Process"/>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasAction">
<rdfs:range rdf:resource="#Action"/>
<rdfs:domain rdf:resource="#Solution"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasQAPairList">
<rdfs:range rdf:resource="#QAPairList"/>
<rdfs:domain rdf:resource="#Problem"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasQAPairCategory">
<rdfs:domain rdf:resource="#QAPair"/>
<rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasFoafInfo">
<rdfs:domain rdf:resource="#FoafInfo"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasProblem">
<rdfs:domain rdf:resource="#Problem"/>
<rdfs:domain rdf:resource="#Case"/>
<rdfs:range rdf:resource="#Problem"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasCaseHistory">
<rdfs:range rdf:resource="#CaseHistory"/>
<rdfs:domain rdf:resource="#CaseContext"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasConcept">
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Question"/>
<owl:Class rdf:about="#Answer"/>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasProfileTemplate">
<rdfs:domain rdf:resource="#OWLSTemplate"/>
<rdfs:range rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl#Profile"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasProcessTemplate">
<rdfs:domain rdf:resource="#OWLSTemplate"/>
<rdfs:range rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#Process"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasServiceTemplate">
<rdfs:domain rdf:resource="#OWLSTemplate"/>
<rdfs:range rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl#Service"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasRanking">
<rdfs:domain rdf:resource="#CaseContext"/>
<rdfs:range rdf:resource="#Ranking"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasUser">
<rdfs:domain rdf:resource="#HistoryEntry"/>
<rdfs:range rdf:resource="#CaseUser"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasQAPair">
<rdfs:domain rdf:resource="#QAPairList"/>
<rdfs:range rdf:resource="#QAPair"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasCaseContext">
<rdfs:domain rdf:resource="#Case"/>
<rdfs:range rdf:resource="#CaseContext"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasSolution">
<rdfs:range rdf:resource="#Solution"/>
<rdfs:domain rdf:resource="#Case"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasAnswer">
<rdfs:domain rdf:resource="#QAPair"/>
<rdfs:range rdf:resource="#Answer"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:id="hasQuestion">
<rdfs:range rdf:resource="#Question"/>
Ontologies

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

<owl:DatatypeProperty rdf:ID="hasAnswerID">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/>
<rdfs:domain rdf:resource="#Answer"/>
</owl:DatatypeProperty>

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

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

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

<owl:DatatypeProperty rdf:ID="hasRankingValue">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/>
<rdfs:domain rdf:resource="#Ranking"/>
</owl:DatatypeProperty>

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

<owl:DatatypeProperty rdf:ID="hasProvenenceURI">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
<rdfs:domain rdf:resource="#CaseContext"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasTimestamp">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
<rdfs:domain rdf:resource="#HistoryEntry"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasQuestionID">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#positiveInteger"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasYesNoValue">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#hasCases">
  <rdfs:domain rdf:resource="#CaseBase"/>
  <rdfs:range rdf:resource="#Case"/>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:DatatypeProperty>

<owl:FunctionalProperty rdf:about="#hasCases">
  <rdfs:domain rdf:resource="#CaseBase"/>
  <rdfs:range rdf:resource="#Case"/>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>

<owl:DatatypeProperty rdf:ID="hasName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  <rdfs:domain rdf:resource="#CaseCreator"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasRole">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:domain rdf:resource="#CaseCreator"/>
</owl:ObjectProperty>

<!--Deprecated stuff-->

<!--owl:ObjectProperty rdf:ID="hasConcept">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Question"/>
        <owl:Class rdf:about="#ConceptAnswer"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>-->

<!--owl:Class rdf:ID="OWLSTemplate">
  <rdfs:subClassOf rdf:resource="#Action"/>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="http://www.daml.org/services/owl-s/1.1/Service.owl#Service"/>
    <owl:Class rdf:about="http://www.daml.org/services/owl-s/1.1/Profile.owl#Profile"/>
    <owl:Class rdf:about="http://www.daml.org/services/owl-s/1.1/Profile.owl#Process"/>
  </owl:intersectionOf>
</owl:Class--> 

<!--owl:Class rdf:ID="ProcessTemplate">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl#Process"/>
</owl:Class>

<!--owl:Class rdf:ID="ServiceTemplate">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl#Service"/>
</owl:Class>

<!--owl:Class rdf:ID="ProfileTemplate">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl#Profile"/>
</owl:Class--> 

<!--owl:ObjectProperty rdf:about="#hasOWLSTemplate">
  <rdfs:domain rdf:resource="#Action"/>
  <rdfs:range rdf:resource="#OWLSTemplate"/>
</owl:ObjectProperty>-->

<!--owl:ObjectProperty rdf:ID="hasHistoryEntry"->
<rdfs:range rdf:resource="#HistoryEntry"/>
<rdfs:domain rdf:resource="#CaseHistory"/>
</owl:ObjectProperty>
<!--owl:DatatypeProperty rdf:ID="hasCaseCategoryName">
<rdfs:domain rdf:resource="#Case"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty)-->
<!--owl:ObjectProperty rdf:ID="hasConcept">
<rdfs:domain rdf:resource="#ConceptAnswer"/>
<rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>-->
<!--owl:ObjectProperty rdf:ID="hasConceptValue">
<rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:domain rdf:resource="#ConceptAnswer"/>
</owl:ObjectProperty>-->
<!--owl:ObjectProperty rdf:ID="hasTriple">
<rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
<rdfs:domain rdf:resource="#Question"/>
</owl:ObjectProperty>-->
</rdf:RDF>

Medical Domain

<?xml version="1.0"?>
<owl:Class rdf:ID="Examination"/>
<owl:Class rdf:ID="MedicalStaffMember"/>
<owl:Class rdf:ID="Person"/>
<owl:Class rdf:ID="Doctor"/>
<owl:Class rdf:ID="GP"/>
<owl:Class rdf:ID="Consultant"/>
<owl:Class rdf:ID="Nurse"/>
<owl:Class rdf:ID="Patient"/>
<owl:Class rdf:ID="PreliminaryExamination"/>
<owl:Class rdf:ID="DetailedExamination"/>
<owl:Class rdf:ID="ECGResult"/>
</owl:Class>
</rdf:RDF>
Appendix C

Ontologies

```xml
<owl:Class rdf:ID="DiagnosticImage">
  <rdfs:subClassOf rdf:resource="#Result"/>
</owl:Class>
<owl:Class rdf:ID="PathologicalResult">
  <rdfs:subClassOf rdf:resource="#Result"/>
</owl:Class>
<owl:Class rdf:ID="Result">
  <owl:equivalentClass>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#ECGResult"/>
      <owl:Class rdf:about="#DiagnosticImage"/>
      <owl:Class rdf:about="#PathologicalResult"/>
    </owl:unionOf>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Ward"/>
<owl:Class rdf:ID="Room"/>
<owl:Class rdf:ID="Condition"/>
<owl:Class rdf:ID="Respiratory">
  <rdfs:subClassOf rdf:resource="#Condition"/>
</owl:Class>
<owl:Class rdf:ID="Highfever">
  <rdfs:subClassOf rdf:resource="#Condition"/>
</owl:Class>
<owl:Class rdf:ID="ConditionOnSet"/>
<owl:Class rdf:ID="ConditionDescription"/>
<owl:Class rdf:ID="ConditionProgress"/>
<owl:Class rdf:ID="ConditionCause"/>
<owl:Class rdf:ID="History"/>
<owl:Class rdf:ID="HistoryDetails"/>
<owl:Class rdf:ID="Status"/>
<owl:Class rdf:ID="NewCase">
  <rdfs:subClassOf rdf:resource="#Status"/>
</owl:Class>
<owl:Class rdf:ID="OldCase">
  <rdfs:subClassOf rdf:resource="#Status"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="performedBy">
  <rdfs:domain rdf:resource="#Examination"/>
  <rdfs:range rdf:resource="#MedicalStaffMember"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="performs">
  <owl:inverseOf rdf:resource="#performedBy"/>
  <rdfs:domain rdf:resource="#MedicalStaffMember"/>
  <rdfs:range rdf:resource="#Examination"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="performedOn">
  <rdfs:domain rdf:resource="#Examination"/>
  <rdfs:range rdf:resource="#Patient"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasResult">
  <rdfs:domain rdf:resource="#Examination"/>
  <rdfs:range rdf:resource="#Result"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasCondition">
  <rdfs:domain rdf:resource="#Patient"/>
</owl:ObjectProperty>
```
<owl:ObjectProperty rdf:ID="hasStatus">
    <rdfs:domain rdf:resource="#Patient"/>
    <rdfs:range rdf:resource="#Status"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasHistoryDoc">
    <rdfs:domain rdf:resource="#Patient"/>
    <rdfs:range rdf:resource="#History"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isAdmittedTo">
    <rdfs:domain rdf:resource="#Patient"/>
    <rdfs:range rdf:resource="#Ward"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasRoom">
    <rdfs:domain rdf:resource="#Ward"/>
    <rdfs:range rdf:resource="#Room"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasHistoryDetails">
    <rdfs:domain rdf:resource="#History"/>
    <rdfs:range rdf:resource="#HistoryDetails"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasOnSet">
    <rdfs:domain rdf:resource="#Condition"/>
    <rdfs:range rdf:resource="#ConditionOnSet"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasDescription">
    <rdfs:domain rdf:resource="#Condition"/>
    <rdfs:range rdf:resource="#ConditionDescription"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasProgress">
    <rdfs:domain rdf:resource="#Condition"/>
    <rdfs:range rdf:resource="#ConditionProgress"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasCause">
    <rdfs:domain rdf:resource="#Condition"/>
    <rdfs:range rdf:resource="#ConditionCause"/>
</owl:ObjectProperty>

</rdf:RDF>

Travelling Domain

<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
xml:base="http://www.semantech.org/ontologies/travellingProblem.owl"
xmlns="http://www.semantech.org/ontologies/travellingProblem.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:ID="Problem"/>
    <owl:Class rdf:ID="TravellingProblem"/>
    <rdfs:subClassOf rdf:resource="#Problem"/>
</owl:Class>
    <owl:Class rdf:ID="AttendConferenceProblem"/>
    <rdfs:subClassOf rdf:resource="#TravellingProblem"/>
</owl:intersectionOf rdf:parseType="Collection"
<owl:Class rdf:about="#Accommodation"/>
<owl:Class rdf:about="#Transportation"/>
</owl:intersectionOf-->
</owl:Class>
<owl:Class rdf:ID="Conference">
<rdfs:subClassOf rdf:resource="#AttendConferenceProblem"/>
<owl:disjointWith rdf:resource="#Transportation"/>
<owl:disjointWith rdf:resource="#Accommodation"/>
</owl:Class>
<owl:Class rdf:ID="Accommodation">
<rdfs:subClassOf rdf:resource="#AttendConferenceProblem"/>
<owl:disjointWith rdf:resource="#Transportation"/>
<owl:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#accommodationORtransport"/>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Transportation">
<rdfs:subClassOf rdf:resource="#AttendConferenceProblem"/>
<owl:disjointWith rdf:resource="#Accommodation"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#accommodationORtransport"/>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Hotel">
<rdfs:subClassOf rdf:resource="#Accommodation"/>
<owl:disjointWith rdf:resource="#Hostel"/>
</owl:Class>
<owl:Class rdf:ID="Hostel">
<rdfs:subClassOf rdf:resource="#Accommodation"/>
<owl:disjointWith rdf:resource="#Hotel"/>
</owl:Class>
<owl:Class rdf:ID="Train">
<rdfs:subClassOf rdf:resource="#Transportation"/>
<owl:disjointWith rdf:resource="#Airplane"/>
</owl:Class>
<owl:Class rdf:ID="Airplane">
<rdfs:subClassOf rdf:resource="#Transportation"/>
<owl:disjointWith rdf:resource="#Train"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="accommodation">
<rdfs:subPropertyOf rdf:resource="#accommodationORtransport"/>
<rdfs:domain rdf:resource="#AttendConferenceProblem"/>
<rdfs:range rdf:resource="#Accommodation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="transportation">
<rdfs:subPropertyOf rdf:resource="#accommodationORtransport"/>
<rdfs:domain rdf:resource="#AttendConferenceProblem"/>
<rdfs:range rdf:resource="#Transportation"/>
</owl:ObjectProperty>
Complete Case Definition Example

<?xml version="1.0"?>
<rdf:RDF xml:base="http://www.semantech.org/ontologies/Case1.owl"
xmlns="http://www.semantech.org/ontologies/Case1.owl#"
xmlns:ccbr="http://www.semantech.org/ontologies/CCBROnto.owl#"
xmlns:daml="http://www.daml.org/2001/03/daml+owl#"
xmlns:expression="http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#"
xmlns:list="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:process="http://www.daml.org/services/owl-s/1.2/Process.owl#"
xmlns:profile="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:service="http://www.daml.org/services/owl-s/1.2/Service.owl#"
xmlns:swrl="http://www.w3.org/2003/11/swrl#"
xmlns:travellingQAP="http://www.semantech.org/ontologies/travellingQAP.owl#">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#" />
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Process.owl#" />
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl#" />
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Service.owl#" />
  </owl:Ontology>
  <service:Service rdf:ID="Case1Service">
    <service:describedBy>
      <process:CompositeProcess rdf:ID="Case1Process"/>
    </service:describedBy>
    <service:presents>
      <profile:Profile rdf:ID="Case1Profile"/>
    </service:presents>
  </service:Service>
  <profile:Profile rdf:about="#Case1Profile">
    <service:presentedBy rdf:resource="#Case1Service"/>
  </profile:Profile>
  <process:CompositeProcess rdf:about="#Case1Process">
    <process:composedOf>
      <process:Sequence>
        <process:components>
          <process:ControlConstructList>
            <list:rest>
              <process:Perform rdf:nodeID="A1"/>
            </list:rest>
          </process:ControlConstructList>
          <list:rest>
            <process:Perform rdf:nodeID="A0"/>
          </list:rest>
        </process:components>
      </process:Sequence>
    </process:composedOf>
  </process:CompositeProcess>
</rdf:RDF>
<list:rest>
  <process:ControlConstructList>
    <list:first>
      <process:Perform rdf:nodeID="A2"/>
    </list:first>
    <list:rest rdf:resource="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#nil"/>
  </process:ControlConstructList>
  <process:ControlConstructList>
    <process:components>
      <process:Sequence>
        <process:composedOf>
          <process:hasInput>
            <process:Input rdf:ID="Case1SolutionInput">
              <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
            </process:Input>
          </process:hasInput>
          <process:hasOutput>
            <process:Output rdf:ID="Case1SolutionOutput">
              <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
            </process:Output>
          </process:hasOutput>
          <process:hasResult>
            <process:Result>
              <process:withOutput>
                <process:OutputBinding>
                  <process:toParam rdf:resource="#Case1SolutionOutput"/>
                  <process:valueSource>
                    <process:ValueOf>
                      <process:fromProcess> <process:Perform rdf:nodeID="A2"/>
                      </process:fromProcess>
                      <process:theVar rdf:resource="#TrainOutputConcept"/>
                    </process:ValueOf>
                  </process:valueSource>
                </process:OutputBinding>
              </process:withOutput>
            </process:Result>
          </process:hasResult>
        </process:composedOf>
        <service:describes rdf:resource="#Case1Service"/>
      </process:Sequence>
      <process:hasDataFrom>
        <process:InputBinding>
          <process:toParam rdf:resource="#ConferenceInputConcept"/>
          <process:valueSource>
            <process:ValueOf>
              <process:fromProcess>
                <process:Perform rdf:nodeID="A2"/>
                <process:fromProcess>
                  <process:theVar rdf:resource="#Case1SolutionInput"/>
                </process:fromProcess>
              </process:fromProcess>
            </process:ValueOf>
          </process:valueSource>
        </process:InputBinding>
      </process:hasDataFrom>
    </process:Sequence>
  </process:ControlConstructList>
</list:rest>
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
</process:Output>
</process:hasOutput>
</process:AtomicProcess>

<process:AtomicProcess rdf:ID="ConferenceService">
  <process:hasInput>
    <process:Input rdf:ID="ConferenceInputConcept">
      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="ConferenceOutputConcept">
      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>

<ccbr:Case rdf:ID="Case_CID1">
  <ccbr:hasCaseContext rdf:resource="#CNTXTCase1"/>
  <ccbr:hasSolution rdf:resource="#SOLNCase1"/>
  <ccbr:hasProblem rdf:resource="#PRBCase1"/>
</ccbr:Case>

<ccbr:CaseContext rdf:ID="CNTXTCase1">
  <ccbr:hasCaseCreator rdf:resource="#CCRCase1"/>
  <ccbr:hasProvenenceURI>file:///ontologies/travel.owl</ccbr:hasProvenenceURI>
  <ccbr:hasRanking rdf:resource="#RNKCase1"/>
</ccbr:CaseContext>

<ccbr:CaseCreator rdf:ID="CCRCase1">
  <ccbr:hasName>Charlie</ccbr:hasName>
  <ccbr:hasRole>Knowledge Engineer</ccbr:hasRole>
</ccbr:CaseCreator>

<ccbr:Rank rdf:ID="RNKCase1">
  <ccbr:hasRankValue>1</ccbr:hasRankValue>
</ccbr:Rank>

<ccbr:Problem rdf:ID="PROBCase1">
  <hasQAPairList>
    <ccbr:QAPairList rdf:ID="QAPLst1">
      <list:first>
        <ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP9"/>
      </list:first>
      <list:rest>
        <ccbr:QAPairList rdf:ID="QAPLst2">
          <list:first>
            <ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP6"/>
          </list:first>
          <list:rest>
            <ccbr:QAPairList rdf:ID="QAPLst3">
              <list:first>
                <ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP5"/>
              </list:first>
              <list:rest>
            </ccbr:QAPairList>
          </list:rest>
        </ccbr:QAPairList>
      </list:rest>
    </ccbr:QAPairList>
  </hasQAPairList>
</ccbr:Problem>
QAPairs for the Travelling Domain

```xml
<rdf:RDF
    xmlns:ccbr="http://www.semantech.org/ontologies/CCBROnto.owl#"
    xmlns="http://www.semantech.org/ontologies/travellingQAP.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#">
  <owl:Ontology rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#"/>
  <ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP2">
    <ccbr:hasAnswer>
      <ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS2">
        <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
      </ccbr:Answer>
    </ccbr:hasAnswer>
    <ccbr:hasQuestion>
      <ccbr:Question rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST2">
        <ccbr:hasQuestionDescription>Attending a conference?</ccbr:hasQuestionDescription>
      </ccbr:Question>
    </ccbr:hasQuestion>
  </ccbr:QAPair>
  <ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP5">
    <ccbr:hasAnswer>
      <ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS5">
        <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
      </ccbr:Answer>
    </ccbr:hasAnswer>
    <ccbr:hasQuestion>
      <ccbr:Question rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST5">
        <ccbr:hasQuestionDescription>Attending a conference?</ccbr:hasQuestionDescription>
      </ccbr:Question>
    </ccbr:hasQuestion>
  </ccbr:QAPair>
</rdf:RDF>
```
Appendix C

Ontologies

</ccbr:hasAnswer>
<ccbr:hasQuestion>
<ccbr:Question rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST5">
<ccbr:hasQuestionDescription>Transportation type is train?</ccbr:hasQuestionDescription>
</ccbr:Question>
</ccbr:hasQuestion>
</ccbr:QAPair>
<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP8">
<ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS8"/>
<ccbr:hasQuestion rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#QST8">
<ccbr:hasQuestionDescription>Accommodation type is hostel?</ccbr:hasQuestionDescription>
</ccbr:Question>
</ccbr:QAPair>
<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP10">
<ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS10"/>
<ccbr:hasQuestion rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#QST10">
<ccbr:hasQuestionDescription>Having a problem?</ccbr:hasQuestionDescription>
</ccbr:Question>
</ccbr:QAPair>
<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP7">
<ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS7"/>
<ccbr:hasQuestion rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#QST7">
<ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#Train"/>
</ccbr:Question>
</ccbr:QAPair>
<ccbr:Question rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST7">
    <ccbr:hasQuestionDescription>Accommodation is required?</ccbr:hasQuestionDescription>
    <ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#Accommodation"/>
</ccbr:Question>

<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP1">
    <ccbr:hasAnswer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS1">
        <ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS1">
            <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
        </ccbr:Answer>
    </ccbr:hasAnswer>
    <ccbr:hasQuestion rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST1">
        <ccbr:hasQuestionDescription>Problem is a travelling problem?</ccbr:hasQuestionDescription>
        <ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#TravellingProblem"/>
    </ccbr:Question>
</ccbr:QAPair>

<ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS4">
    <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
</ccbr:Answer>

<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP4">
    <ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS4">
        <ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS4">
            <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
        </ccbr:Answer>
    </ccbr:hasAnswer>
    <ccbr:hasQuestion rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST4">
        <ccbr:hasQuestionDescription>Transportation required?</ccbr:hasQuestionDescription>
        <ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#Transportation"/>
    </ccbr:Question>
</ccbr:QAPair>

<ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS3">
    <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
</ccbr:Answer>

<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP3">
    <ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS3">
        <ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS3">
            <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
        </ccbr:Answer>
    </ccbr:hasAnswer>
    <ccbr:hasQuestion rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QST3">
        <ccbr:hasQuestionDescription>Transportation type is airplane?</ccbr:hasQuestionDescription>
        <ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#Airplane"/>
    </ccbr:Question>
</ccbr:QAPair>
<ccbr:hasQuestion rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#QST3"/>
</ccbr:QAPair>
<ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS6">
  <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
</ccbr:Answer>
<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP6">
  <ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS6"/>
  <ccbr:hasQuestion rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#QST6">
    <ccbr:hasQuestionDescription>Accommodation type is hotel?</ccbr:hasQuestionDescription>
    <ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#Hotel"/>
  </ccbr:hasQuestion>
</ccbr:QAPair>
<ccbr:Answer rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#ANS9">
  <ccbr:hasAnswerDescription>Yes</ccbr:hasAnswerDescription>
</ccbr:Answer>
<ccbr:QAPair rdf:about="http://www.semantech.org/ontologies/travellingQAP.owl#QAP9">
  <ccbr:hasAnswer rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#ANS9"/>
  <ccbr:hasQuestion rdf:resource="http://www.semantech.org/ontologies/travellingQAP.owl#QST9">
    <ccbr:hasQuestionDescription>Conference registration required?</ccbr:hasQuestionDescription>
    <ccbr:isRelatedTo rdf:resource="http://www.semantech.org/ontologies/travellingProblem.owl#Conference"/>
  </ccbr:hasQuestion>
</ccbr:QAPair>
</rdf:RDF>

Configuration file

<?xml version="1.0" encoding="UTF-8"?>
<config>
  <taxonomies>
    <taxonomy path="/ontologies/problems/travellingQAP.owl"/>
    <taxonomy path="/ontologies/problems/medicalQAP.owl"/>
  </taxonomies>
  <domains>
    <domain path="/ontologies/domains.owl"/>
    <domain path="/ontologies/domains/travel.owl"/>
    <domain path="/ontologies/domains/travellingProblem.owl"/>
    <domain path="/ontologies/domains/travel_reduced.owl"/>
</domains>
</config>
<domain path="/ontologies/domains/medicalProblem.owl"/>
<domain path="/ontologies/domains/printing.owl"/>
</domains>
<dbSettings dbName="MySQL" password="testDB"
    url="jdbc:mysql://localhost/test" username="testDB"/>
</config>
References


References


[28] Gupta, K., Taxonomic Conversational Case-Based Reasoning, 4th International Conference on Case-Based Reasoning, 2001


References


[33] IUCBRF, Indiana University Case-Based Reasoning Framework http://www.cs.indiana.edu/~sbogaert/CBR/

[34] Jena API, http://jena.sourceforge.net/


[43] Maedche, A., Zacharias, V., Clustering Ontology-based Metadata in the Semantic Web, Joint Conferences (ECML'02) and (PKDD'02), Finland, Helsinki, 2002


[48] OWL, http://www.w3.org/TR/owl-features/


References


[54] Peer, J., A POP-based Replanning Agent for Automatic Web Service Composition, Second European Semantic Web Conference (ESWC'05), 2005


[56] RDF, http://www.w3.org/TR/rdf-schema/


[59] Scicluna, J., Abela, C., Montebello, M., Visual Modelling of OWL-S Services, IADIS International Conference WWW/Internet, Madrid Spain, October 2004


[65] SOAP, http://www.w3.org/TR/soap/


[70] SWSO, http://www.w3.org/Submission/SWSF-SWSO/#fc-section


References


[80] WSCI http://www.w3.org/TR/2002/NOTE-wsci-20020808/

[81] WSCL http://www.w3.org/TR/2002/NOTE-wscl10-20020314/


[83] WSDL, http://www.w3.org/TR/wsd1

[84] WSDL 2.0 specifications, http://www.w3.org/TR/wsd120/

[85] WSMO, http://www.wsmo.org/TR/d30/v0.1/
