AN INTELLIGENT FRAMEWORK FOR DYNAMIC WEB SERVICES COMPOSITION IN THE SEMANTIC WEB

DHAVALKUMAR THAKKER

A thesis submitted in partial fulfilment of the requirements of
Nottingham Trent University for the degree of
Doctor of Philosophy

October 2008

Acknowledgements

I would like to express my deep and sincere gratitude to my director of studies, Dr Taha Osman. His expertise, understanding, encouragement and personal guidance have provided an excellent basis for the present thesis. I would also like to extend thanks to my supervisors, Dr Evtim Peytchev and Professor David Al-Dabass for their guidance, advice, and input regarding this research.

I am also grateful to the Maryland Information and Network Dynamics Lab Semantic Web Agents Project (MINDSWAP) members Bijan Parsia and Evren Sirin for providing open source access to the OWL reasoners Pellet and the OWL-S API.

I believe that intensive courses like PhD are impossible to succeed without the patience and support of a caring family. I owe my loving thanks to my wife Neha, without her encouragement and understanding it would have been impossible for me to finish this work. My special gratitude is due to my father and brother for their loving support. My adopted home in London with Thacker family has always been a great place to relieve the stress of studies. I owe them a big gratitude for providing such an environment. Acknowledgement is also due to my wonderful friends Raxit, Sonali, Kunal, Amee, Vivek and Gangotri for being part of this long journey.

My friend Vassias Vassiliades deserves special mention for his help during my stay in Nottingham. Finally, to friends and colleagues who have not been mentioned, thank you for your ideas and informal discussions.

Abstract

As Web services are being increasingly adopted as the distributed computing technology of choice to securely publish application services beyond the firewall, the importance of composing them to create new, value-added service, is increasing. Thus far, the most successful practical approach to Web services composition, largely endorsed by the industry falls under the static composition category where the service selection and flow management are done a priori and manually. The second approach to web-services composition aspires to achieve more dynamic composition by semantically describing the process model of Web services and thus making it comprehensible to reasoning engines or software agents. The practical implementation of the dynamic composition approach is still in its infancy and many complex problems need to be resolved before it can be adopted outside the research communities.

The investigation of automatic discovery and composition of Web services in this thesis resulted in the development of the e**X**tended **S**emantic **C**ase **B**ased **R**easoner (XSCBR), which utilizes semantic web and AI methodology of Case Based Reasoning (CBR). Our framework uses OWL semantic descriptions extensively for implementing both the matchmaking profiles of the Web services and the components of the CBR engine.

In this research, we have introduced the concept of runtime behaviour of services and consideration of that in Web services selection. The runtime behaviour of a service is a result of service execution and how the service will behave under different circumstances, which is difficult to presume prior to service execution. Moreover, we demonstrate that the accuracy of automatic matchmaking of Web services can be further improved by taking into account the adequacy of past matchmaking experiences for the requested task. Our XSCBR framework allows annotating such runtime experiences in terms of storing execution values of non-functional Web services parameters such as availability and response time into a case library. The XSCBR algorithm for matchmaking and discovery considers such stored Web services execution experiences to determine the adequacy of services for a particular task.

We further extended our fundamental discovery and matchmaking algorithm to cater for web services composition. An intensive knowledge-based substitution approach was proposed to adapt the candidate service experiences to the requested solution before suggesting more complex and computationally taxing AI-based planning-based

transformations. The inconsistency problem that occurs while adapting existing service composition solutions is addressed with a novel methodology based on Constraint Satisfaction Problem (CSP).

From the outset, we adopted a pragmatic approach that focused on delivering an automated Web services discovery and composition solution with the minimum possible involvement of all composition participants: the service provider, the requestor and the service composer. The qualitative evaluation of the framework and the composition tools, together with the performance study of the XSCBR framework has verified that we were successful in achieving our goal.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADoM	Aggregate Degree of Match
AI	Artificial Intelligence
B2B	Business-to-Business
BPEL	Business Process Execution Language
BPM	Business Process Management
BPML	Business Process Modelling Language
BPR	Business Process Reengineering
CBR	Case Based Reasoning
CSP	Constraint Satisfaction Problem
DDM	Domain Dependency Module
DL	Description Logics
DoM	Degree of Match
DPDWS	Dynamic Pool for Domain-Specific Web services
DSS	Domain Specification Stage
EAI	Enterprise Application Integration
GUI	Graphical User Interface
IR	Information Retrieval
KBS	Knowledge Based Substitution
OWL	Web Ontology Language
OWL-S	Semantic Mark-up for Web services
QoS	Quality of Service
RDF	Resource Description Framework
RR	Rule-based Relationships
SCBR	Semantic Case Based Reasoner
SOA	Service Oriented Architecture

SOAP	Simple Object Access Protocol
SWRL	Semantic Web Rule Language
TR	Taxonomy Relationships
UDDI	Universal Description and Discovery Interface
UI	User Interface
URI	Uniform Resource Identifier
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
WSMO	Web Services Modelling Language
WWW	World Wide Web
XML	eXtensible Mark-up Language
XSLT	XML Stylesheet Language

1 Chapter One: Introduction

System development using Web services is encouraging scenarios where individual or integrated application services can be seamlessly and securely published on the web without the need to expose their implementation details. However as Web services proliferate, the importance of accurate, yet flexible, matchmaking of similar services gains importance both for the human user and for dynamic composition engines. The goal of this research is to investigate the utilization of the semantic web in building developer-transparent frameworks facilitating the automatic matchmaking and composition of Web services.

This chapter provides information about Service Oriented Architectures (SOA), focusing on Web services, the orchestration of which is the subject of this research. The Web services technology provides new opportunities to harness and complement the capability of World Wide Web (WWW). These opportunities are discussed, together with the problems preventing a wider adoption of this new technology. The focus is particularly on analyzing the complexity of development and facilitation provided to the users of the technology. The following subsections introduce the concepts of Web services, Web services composition and the semantic web.

1.1. Web services

The Internet has become a market-place for a colossal variety of application services ranging from e-commerce and Internet information services, to services that facilitate trading between business partners, better known as Business-to-Business (B2B) relationships. Traditionally these services are facilitated by distributed technologies such as Remote Procedure Call (RPC), Common Object Request Broker Architecture (CORBA), Remote Method Invocation (RMI), and more recently Web services.

Web services are an instance of Service Oriented Architecture (SOA) [1]. The SOA is an application architecture focused on business services. These business services can be any business application contributing to the information system of enterprises, services that organizations provide to clients, partners and employees. For example, a bank offers ATM transactions to clients, payroll for employees and secure card transactions to business partners.

SOA can be thought of as an approach to building IT systems in which business services are the key principle to align IT systems with the needs of businesses [2]. In contrast, earlier approaches to building IT systems intended to directly use specific implementation methodologies such as object-orientation, procedure-orientation or message-orientation to solve these business problems, resulting in systems that were often tied to the features and functions of a particular execution environment making interoperability unfeasible. SOA solves the problem of interoperability by abstracting implementation details of applications and facilitates the developers with the possibility of transparent and seamless integration of various applications also resulting in reduced cost of operations and development.

Owing to these features, the Service Oriented Architecture has made a huge impact on how IT applications are implemented, reused and integrated in organizations [3]. Following are some examples from industry highlighting the benefits of SOA when employed to develop and integrate real-world applications.

- Amazon.com is a pioneer E-Commerce company that specializes in selling goods over the Internet [4]. Amazon has spent over a decade and $2 billion building a superior web-scale computing platform. However, the initial growth of the Amazon.com computing platform was in the direction of interoperating feature components inside the firewall; e.g., the catalogue, shopping cart, and personalization engine. Through their web services platform, Amazon is beginning to open these features up to public use by providing open access to their Web services to perform various tasks on their web site that involves business partners and consumers. For example, they provide openly available Web services [5] which allows client programs to browse Amazon's databases, locate books and other products and put them in a Web ‘shopping cart’ that can be accessed from the main Amazon Web site using a browser to finalize purchases. The business partners can utilize value-added services such as Amazon Flexible Payment Service (Amazon

FPS) which provides a set of web services APIs allowing the movement of money between any two entities, humans or computers.

- The auction e-commerce web site, eBay™ has built a service architecture and successfully uses it to enable integration across disparate technology stacks [4]. For example, they use SOA for enabling open interoperation between their C++ and Java technologies.

- Another success story is Avis Budget group which is a recognized brand in the global vehicle rental has implemented SOA in form of OMEGA (One Merged Enterprise and Global Architecture) to help ensure a positive and consistent customer experience and keep loyal renters coming back time and again [6]. For example one of the applications in OMEGA is E-Receipts, which provides convenience for customers by allowing them to receive electronic receipts via email. Using OMEGA, Avis Budget connects customer contact channels i.e., call centres, airport rental counters, standalone facilities, and the Internet in over 70 countries and are able to reuse services created for one application on subsequent projects. For example, drivers who need rental cars while their personal cars are being repaired can often have their insurance companies pay for the rentals. Avis Budget is using the notification service originally built for E-Receipts to communicate with insurance carriers and drivers about payment authorization.

- The HP Corporation that specializes in diverse product range from personal computers to digital cameras achieved a $70 million cost savings from its global IT operations as a direct result of SOA deployments [7]. The main contributors to these savings were in terms of reduction of redundancy and reuse across services and a long-term payoff from increased business agility, and ability to react quicker to the marketplace.

The majority of these SOA implementations use Web services as an implementation technology. Web services are implementation of SOA with three participants: service provider, service requestor and service registry. The service provider implements a Web service and provides a description file for such a service via service registry. The service requestor is essentially a client program which retrieves the service description from a service registry and invokes it locally. The service registry is the meeting point

for service requestors and service providers. Figure 1 shows the SOA with Web services and the base protocols consumed in the architecture [2].

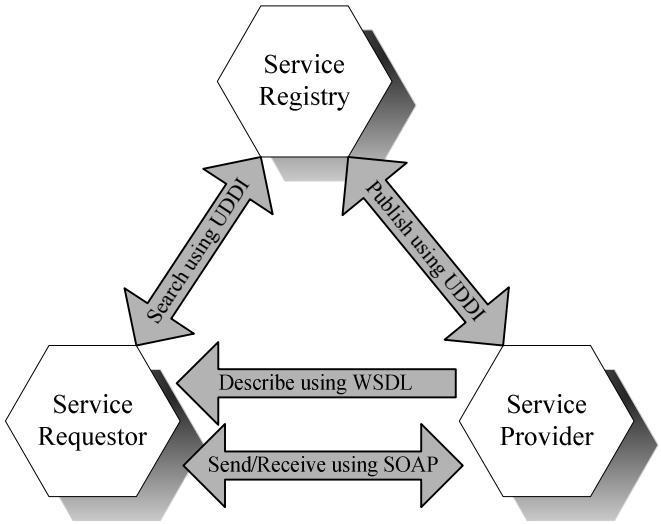


Figure 1 Web services base protocols

The key to SOA is that services needs to be interoperable and location independent. For Web services these requirements refer to standardizing the protocols for searching and publishing with registry, describing service and communication with bi-directional messages between requestor and providers. Web services protocols for these components are standardized using eXtensible Mark-up Language (XML) as defined by the World Wide Web Consortium's working group on Web services architecture [8]:

"Web services as a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

As the definition suggests, XML messaging is centred to the Web services technology and being used in data formatting, serialization, and transformation. The main advantage XML offers Web services is data independence so that data types and structures are not tied to the underlying implementations of the services. The use of standardized XML makes Web services platform neutral and language independent technology. Here, the XML serialization refers to Simple Object Access Protocol (SOAP) [9] a protocol which uses ubiquitous HTTP for the transport mechanism. HTTP is considered as a secure protocol, thus it allows Web services to be securely exposed beyond the firewall.

Web services expose a business service to the outside world, using the WSDL (Web Services Description Language) [10] standard, which provides the grammar for describing services as a set of endpoints that exchange messages.

The SOA architecture of Web services is centred on WSDL, SOAP and UDDI specifications. In this architecture, the service provider has the service implemented and described using WSDL, while service requestor is looking for the service to carry out their task. The Web Service architecture needs a registry to provide Web services information so that publishers and consumers can find each other. This specification for registry is Universal Description Discovery and Integration [11]. Web Services can be published and discovered using UDDI protocol.

To summarize, Web services based on SOA architecture can be published using UDDI, with WSDL based description, and can be searched, called and bound at run time making it loosely-coupled and highly-accessible.

To take advantage of these features of Web services, network applications services have to be developed as Web services or converted into Web services using some wrapping mechanism to allow non-Web services to function as Web services. For example, in [12] an application-independent service wrapper is proposed in order to ease the migration of existing application code in the service-based framework. Moreover, multiple Web services can be integrated either to provide a new, value-added service to the end-user or to facilitate co-operation between various business partners. This integration of Web services is called “Web services composition” [13] and is feasible to achieve because of the Web services advantages of being platform, language neutral and loosely coupled.

1.2. Web services composition

Web services composition provides a value-added dimension to the Web services advantages. Using composition techniques, developers and users can solve complex problems by combining available services and arranging their workflow to best suit the problem requirements. The logic for Web services composition mainly involves two sub-problems: “discovery” and “matchmaking” of candidate Web services that fulfil the problem requirements and flow management for such Web services [13].

1.2.1. Web services discovery and matchmaking

Composition is applicable when the individual services are not sufficient to address the problem requirements or the individual services need to be integrated to provide new value-added services. The problem of discovery and matchmaking refers to the searching and matching of services from the available services that in accumulation provides the required functionality or creates the value-added service. The problem of discovering and matchmaking Web services is sometimes also referred to as "Web services selection problem".

1.2.2. Flow management

Flow management is supplementary to the discovery and matchmaking problem where the control and dataflow for the discovered and matched services will create the implementation layout of the integrated service. The control flow refers to the order in which Web services operations are invoked and the data flow is the order in which the messages are passed between the Web services operations.

The document containing the description of selected services and flow management details is referred to as "composition scheme" in this thesis.

The level of automation provided in performing selection of services and flow management classifies composition into static, semi-automatic and dynamic. Static composition involves prior hard coding of the service selection and flow management. Performing selection and flow management on the fly, in machine-readable format leads to dynamic composition. In semi-automatic composition, the service composer is involved at some stage.

A motivating example of Web services composition is the classic travel agent problem. In the global village live in, travellers often refer to online solutions to get the best value for money for their itinerary, which might include multi-modal transport and additional services such as accommodation. Most often these services are provided by a number of suppliers that must integrate their efforts to fulfil the customer requirements. One of the most successful examples of such composition is the Galileo International online travel agency [14]. The company uses XML Web services to manage over 45000 travel agencies and small and medium sized enterprises. The Galileo Web services enable

technology development partners and suppliers of air, hotel, cars and cruise services to integrate Galileo's data and functions into their applications via the Internet.

In static composition, developers select these individual services and define the flow management by hand. In automatic composition, intelligent programs or agents decide the suitability of the services with respect to the problem requirements and create the flow management based on the service descriptions. The automatic composition process should have the capability of understanding the Web services descriptions in order to determine the service suitability and to compile flow management. In semi-automatic composition, developers assist the programs or agents during composition process.

Considering the growth of Web services and scale of application services available on the World Wide Web, static Web services composition has the following shortcomings:

- The manual effort involved in static composition makes it cost-prohibitive. For example, in case of the Galileo International example discussed above, manual composition would require hard coding up to 45000 Web services for the composition, which is time-consuming and error-prone exercise.

- Static composition assumes the availability and longevity of Web services. Contrary to this, in the WWW environment, new services are offered and withdrawn quite often.

- Static composition does not address the problem of Business Process Reengineering (BPR) - an important aspect for any organization. BPR involves re-configuration of business processes to adapt the new challenges faced by an organization, i.e. competition or evolution of business rules over time period. This motivates the need for more flexible Web service composition

Automated composition can offer following benefits:

- Automated composition can accommodate an increased number of Web services and possible combinations of such Web services.

- Automatic Web services composition can support highly adaptive systems, where services are automatically added or removed from the composition scheme.

- Automatic composition can take the human developer out of the composition process creation, thus reducing the product-to-market cycle and subsequently the production cost.

The automation of Web services composition necessitates the description of Web services capabilities in a machine understandable format. This ties in with the semantic web premise of annotating the traditional World Wide Web to make it computer-interpretable, user-apparent and agent-ready.

1.3. Semantic Web

1.3.1. Semantic Web Architectures

The Web was invented by Tim Berners-Lee amongst others, a physicist working at CERN. The Semantic web is perceived as the extension of current World Wide Web (WWW), defined as follows [15]:

"The next generation WWW is a Web in which machines can converse in a meaningful way, rather than a web limited to humans requesting HTML pages."

The fundamental premise of the semantic web is to extend the web's current human-oriented interface to a format that is comprehensible to software programs. For instance, in a future scenario of the Semantic Web, intelligent agents should be able to set up an appointment between a patient and the doctor, looking at both timetables, and then finding the best way to the clinic without the patient having to interfere in the process. Hence, the user would only have to specify the appointment requirements and the semantic agent will complete the task on its own.

This example depicts the basic idea of semantic web which is a web in which remote machines can converse with each other in a meaningful way, rather than a web limited to humans requesting HTML pages. The approach adopted by the Semantic Web to achieve this is formalized in terms of layers built on top of XML. XML is a mark-up language which allows user-defined tags and provides almost forty simple data-types. This facilitates the structuring of Web pages by defining complex information as shown in the figure below.

```
<Travel Regions>
        <International>
                <FromCountry> India </FromCountry>
                <ToCountry> USA </ToCountry>
        </International>
```

```
        <Domestic>
                <FromCountry> India </FromCountry>
                <ToCountry> India </ToCountry>
        </Domestic>
</Travel Regions>
```

Figure 2 Defining Tags with XML

The information defined with XML can be parsed and displayed using style sheet languages i.e., XSLT. Use of XML eliminates the limitation of HTML, as the developer has more freedom in defining web pages and is not limited with the simple HTML tags. However, structuring of information by XML is still restrictive, as the syntax does not permit defining relationship between different terms. For example, in above code snippet, using XML one cannot define the relationship between Domestic Travel and International Travel. This shows that XML alone provides only syntactical support and has no notion for the meanings required for achieving the goal of the Semantic Web. However, Semantic Web uses the structuring capability of XML to achieve relationship between different terms or concepts.

Figure 3 describes the cake layer approach adopted for semantic web. The XML-based Resource Description Framework (RDF) [16] and Web Ontology Language (OWL) [17] are the specifications from the W3C (World Wide Consortium) to add semantics. These specifications provide language expressiveness and simulate human reasoning.

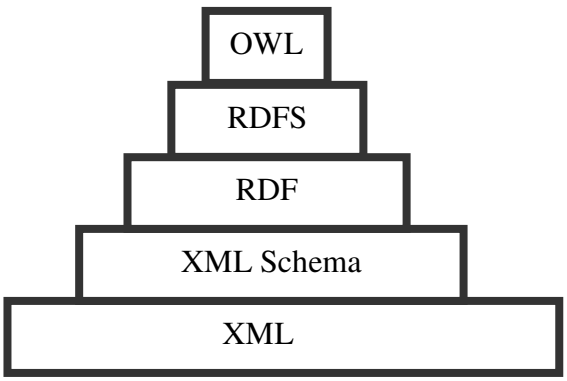


Figure 3 Layered Technologies for Semantic Web

The standard of interest to the Web services composition problem is OWL. OWL uses and extends RDF to specify ontologies. Ontologies are based on OWL which enlarges the possibilities of XML and RDF to introduce meanings. For example, the relationship between concepts “International” and “Domestic” is possible to define as shown in Figure 4. OWL defines Domestic as a subcategory of Travel Domain while disjoint with

the International. Some examples of ontology terms are: class, subClassOf, domain, range, and individual as well as different types of properties like object or data properties.

```
<owl:Class rdf:ID="Domestic">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TravelRegions"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <TravelRegions rdf:ID="International">
      <owl:disjointWith rdf:resource="#Domestic"/>
   </owl:disjointWith>
 </owl:Class>
```

Figure 4 Ontology describing relationship between concepts

To summarize, ontologies are like dictionaries where the meaning of concept can be described in form of unambiguous semantic descriptions. In this way, ontologies define common specifications of domain-related concepts. Other aspect of ontologies is that the reasoner can be designed to interpret these conceptual meanings or derive deduction from the semantic description making the solutions program based and computer-interpretable.

1.3.2. Semantic Web meets Web services

The semantic web technology is an integral part of approach to Web services composition. The logic behind this argument can be traced back to the impact semantic web has on the field of Information Retrieval (IR) [18] or search technology. The IR technology is of paramount importance to organizations due to the growth of computing that has resulted into digitization of personal, commercial and recreational information. This growth requires a technology like IR that mines data to find out relevant information [19]. The goal of IR technology is to understand a request and find relevant information.

The current generations of IR technology and their implementations, search engines, rely on analysing the text in these information sources to matchmake it with the text or keywords in the user query. Some search engines perform a full-text search while others search into some portion of the information sources depending on the algorithms they operate on.

The complete success of these search techniques remains hampered by the fact that they rely on free-text search [20], hence while cost-effective to perform, these search techniques can return irrelevant results as it primarily relies on the recurrence of exact words in the text in the information sources. The inaccuracy of the results increases with the complexity of the query. For example, if you are looking for information on search engines and naturally type "Search Engines" in the Google™ search engine, then the engine returns some good results that provide details on "kind of search engines and how they work", however when we add keywords "to find the impact of search engines on commerce and recreational activities", this disorients the Google search engine and it returns results on "the search optimization techniques (techniques to deal with optimizing websites for search engines)" and something as off tangent as "recreational activities of Wetumpka Area Chamber of Commerce and site on geospatial framework for the Coastal Zone in United States".

Any significant contribution to the accuracy of matchmaking results can be achieved only if the search engine can "comprehend" the meaning of the data in the information sources. For instance, if the search engine can understand that commerce is an act of buying and selling and recreational are activities done for fun or time pass. In the scientific journal article [15] of May 2001, Tim Berners Lee, James Hendler and Ora Lassila introduced concept of "Semantic Web" that precisely targets the problem of making data from the information sources comprehensible to the search engines and ultimately computers.

The same logic applies to Web services discovery engines where the semantic web is applicable to the problem of automated Web services discovery and composition to solve the problems of accuracy. In addition to providing a tool for addressing the problem of Web services discovery, the semantic encoding of web services offers the opportunity of automating Web services composition, as a rich, semantic web representation language can provide machine understandable descriptions for interpreting service capability. For example, richer semantics can support greater automation of service selection and invocation, automated translation of message content between heterogeneous interoperating services, automated or semi-automated approaches to service composition, and more comprehensive approaches to service monitoring and recovery from failure [21]. To meet this need, researchers have been developing languages, architectures and related approaches; the resulting body of work

goes under the heading of semantic Web services [22]. Semantic Mark-up for Web services (OWL-S) [21], Web Service Modelling Ontology (WSMO) [23] and Web Services Semantics (WSDL-S) [24] are such main Semantic Web services efforts. The details of some of these efforts are outlined in the literature survey chapter of this thesis.

1.4. Research Direction

1.4.1. Motivation

Despite the evident popularity of Web services as a secure distributed computing paradigm and the value-added dimension that composition adds to it, the practical adoption of the technology is still to gather the expected pace. The main thesis of this research work is based on the theory that assistance with the facilitation of the composition process to the service providers and the composers plays a major role in encouraging the adoption of the Web services technology.

The facilitation to be provided to the service developers and providers can be considered in terms of the minimum effort they have to make to subscribe their services to composition schemes. Two possible scenarios are:

- The application service is not yet published as a Web service, in which case a blue-print is required to build a Web service wrapper that plugs the application to the composition interface.
- The service provider has exposed the application as Web service that has a specification and format conceptually similar but syntactically different from what the composition interface expects. Ideally here the service provider should not be asked to re-write the Web service, but some work-around is suggested to overcome the mismatch.

The composition techniques can be judged based on how seamlessly they allow the service providers to take part in the composition for the above scenarios.

For the service composer, which can be a human developer or intelligent program/software agent, the facilitation constitutes automating as many steps as possible in order to build and program the composition logic. These steps include:

- Matchmaking services to required solutions.

- Implementing execution flow management in the matchmaking process results in composition of services.
- Automatic integration of alternative services.
- Overcoming mismatches in the service descriptions as transparently as possible.

The literature in the area of Web services composition reflects the fact that in search of automation the focus of work has been transferred from giving priority to the composition participants to the application of various existing formal methodologies to solve composition problem, often at the expense of the practicality of the solutions. Hence, the aim of this work is to pursue a pragmatic vision of contributing towards the efforts of making the composition process as transparent as possible to all the composition participants. This should allow developers and users to perform everyday chores using Web services without being worried about behind the scene technical details.

1.4.2. Application Example

To highlight the type of problems we are aiming to address and the facilitation required in solving such problems, we here present an application example based on travel agent [25] for Web services composition. We believe that travel agent is an ideal application of web services composition where travel agent has to deal with number of sub-domains under the travel domain, i.e., bus, rail, airline and hotel etc and there are already existing travel domain applications in abundance that could be converted in Web services and can take benefit of dynamic discovery and composition mechanism. Here we present a scenario that present the role of service participants and depicting how dynamic Web services composition benefits each of them.

The service requestor initiates service request. We assume that the requestor would like to provide inputs in terms of constraints and preferences on the outputs and results they receive. For example,

Inputs *(Name, Expected Departure Date, Expected ArrivalDate, No of Passengers, Departure City, Arrival City)*

Constraints & Preferences:

Provision of a travel package with Airline and Hotel

Output currency must not be USD.

Output currency must be in GBP.

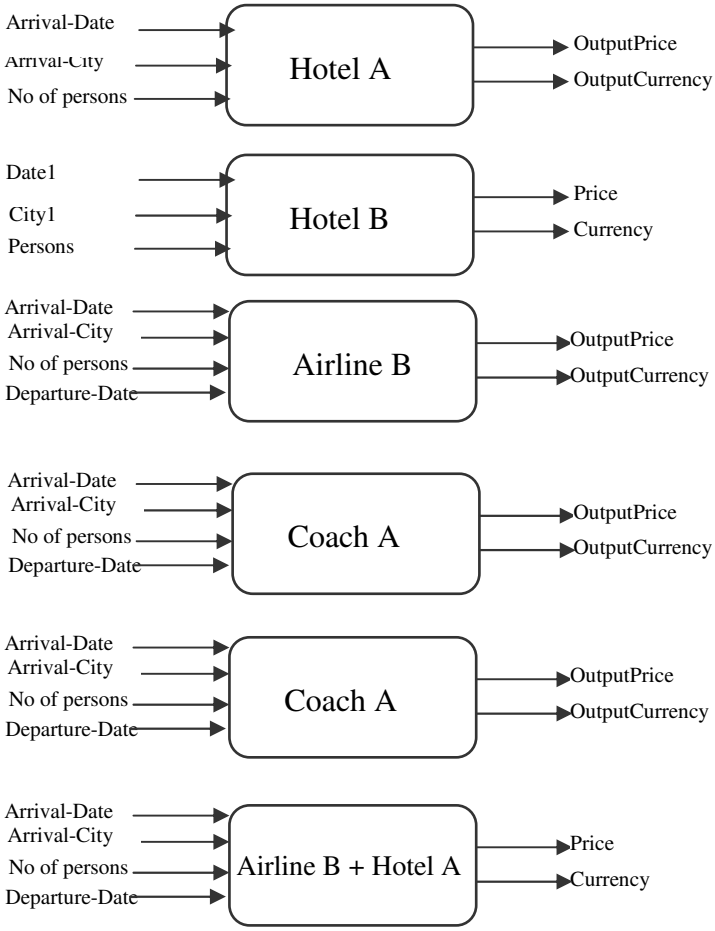
The execution speed of the service must be 3 seconds.

I do not particularly like British Airways.

I get sick in bus, so please do not include bus in the results.

Outputs *(Price, Currency)*

The service provider may want to be part of a composition by providing their service to the composer or to a generic travel service registry. Below are a number of service descriptions from the providers of various domains. Note the variation on the service descriptions.



The composer will be a travel agency that takes requestor's request and finds suitable service(s). In dynamic web services composition, rather than having a fixed list of Web services that travel agency always accesses, the agency would like to instead

dynamically discover Web services at each transaction. This allows the travel agency to avoid having a pre-negotiated agreement with each Web service. The ultimate goal of the intelligent framework is to satisfy user request and facilitate each of the participants in the process of achieving required results.

We would like to mention that our working example is intentionally made simpler than it is required for practical cases. This has been done in order to keep simplicity of presentation. In practice there can be more alternative services available with more parameters for the service for example, parameters such as travel itinerary, routes to avoid, number of rooms, departing and returning timings and so on.

1.4.3. Research Questions

The chief research question this thesis tries to answer is:

> *"How can we develop an intelligent framework that utilizes semantic web for automated Web services discovery and composition and provides automation to the composition participants in a transparent manner?"*

In order to be able to answer this question, we define a set of research questions (RQ) that addresses the problem in detail.

RQ1: Web services composition is mainly a task performed by human developer, how can this task be automated using software programs?

RQ2: Workflow-based techniques are a popular and widely adopted option for application integration/Web services composition. Can semantic technologies inject the required intelligence to aid the workflow techniques in achieving more dynamic, perhaps automated service composition?

RQ3: Investigation of problem solving methodologies that represents a viable approach for solving the problem of automatic Web services composition problem.

RQ4: Selecting the appropriate implementation technology from the abundance of standards available.

RQ5: The main thesis of this research work is based on the theory that assistance with the facilitation of the composition process to the service participants (service requestor,

provider and the composers) plays a major role in encouraging the adoption of the Web services technology. This research shall address question of the facilitation by providing assistance to the service participants in their respective tasks in the composition process.

RQ6: What are the criteria for evaluating the provided functionality compared to that offered by other frameworks?

1.4.4. Problem Statement

The main objective of this thesis is to investigate and provide an intelligent framework for automated Web services discovery and composition. The framework shall provide tools and methodologies that alleviate the burden of dynamic Web services composition from the participants of the framework, namely service provider, service composer and service requestor.

1.4.5. Proposed Solution

We have approached the problem through designing and implementing a prototype system for dynamic Web services composition. The following assumptions and considerations were made:

- In this research, we have argued for the importance of considering the execution values for semantically-described non-functional Web services parameters in decision making regarding Web service adequacy for the task. This is because the service behaviour is impossible to predict prior to execution and can only be generalized if such execution values are stored and reasoned for deciding service capability. AI planning and Intelligent Agent based reasoning methods offer rule-based reasoning methodology rather than experience-based. Hence, we used Case Based Reasoning method that allows capturing experiences and reasoning based on them.

- We have implemented a Semantic Case Based Reasoner (SCBR), which captures Web service execution experiences as cases and uses these cases for finding a solution for new problems. The search considers domain-specific criteria and user preferences to find Web services execution experience that solved a similar problem in the past.

- The initial version of the framework assumed that the case library that holds Web services execution experiences, contains suitable cases for every possible problem. This assumption is not always satisfied considering the vast number of problems and problem parameters. Moreover, the framework also needs to deal with situations where the match-making indicator (aggregate degree of match) of final results is below the domain-specific expected match-making indicator set by the domain administrator. The framework also required to deal with negative user feedback, where the matched services are not acceptable to the user. To address these limitations we extended the implementation with a CBR process - case adaptation, sometimes also refereed as REVISE phase in the CBR theory. The process of adaptation is applied in our framework when the available cases cannot fulfil the problem requirements, so matchmaking is attempted by adapting available cases. This process looks for prominent differences between the retrieved case and the current case and then applies formulae or rules that take those differences into account when suggesting a solution.

- The final solution advocates an exhaustive knowledge-based substitution approach to adapt the functional and non-functional attributes of the candidate case to the requested solution before suggesting more complex and computationally taxing AI-based planning-based transformations that integrate the service profile of a number of cases to deliver candidate solutions.

1.5. Research Methodology

The research methodology for this project was based on the following research activities: literature survey, requirement analysis and refinement, incremental development and evaluation.

1. Literature survey

- The research involved extensive literature survey in the fields of Web services, semantic web, web services composition and Artificial Intelligence based problem-solving methods. The literature survey was carried out to ensure the originality of work and to avoid the repetition of existing work done in the field.

- We studied two categories of Web services composition approaches: the first category largely endorsed by the industry, borrows from business processes'

workflow management theory to achieve the formalization necessary for describing the data flow and control in the composition scheme. The second category mainly promoted by the research community, aspires to achieve dynamic composition by semantically describing the process model of Web service and thus making it comprehensible to reasoning engines or software agents.

- We studied workflow techniques based on BPEL, WS-CDL and BPML together with the semantic web services description languages like OWL-S and WSMO.

- We also studied number of AI methodologies that can be utilized in the procedure of composition to inject level of intelligence. In particular, we focused on AI Planning, Constraint Satisfaction Problem (CSP), Case Based Reasoning (CBR), genetic algorithm, agents and software synthesis.

- The literature survey was an iterative activity through out the PhD where survey was an important input parameter to requirement analysis and refinement for this project. Thus similar way, requirements also triggered the need for carrying out literature survey at the various phases of project cycle.

2. Requirement analysis and refinement

- Like many research problems in computer science, the methodology, tools and specifications that required to fulfil our motivation and answer research questions in this project were analyzed and refined.

- The analysis and refinement was in light of the required facilitation to be provided to the service participants and also to automate the process of discovery and composition.

3. Incremental Development

- Development of a practical solution based on a hybrid approach that merges the benefit of practicality of use and adoption popularity of workflow-based (BPEL-based) composition, with the advantage of using semantic description to aid the composition participants in automatic discovery and interoperability of the composed services.

- Development of a Semantic Case Based Reasoner (SCBR), which captures Web service execution experiences as cases and uses these cases for finding a solution for new problems. The search considers domain-specific criteria and user preferences to find Web services execution experience that solved a similar problem in the past. The reasoner addresses the problem of Web services discovery and matchmaking

- Extending the discovery and matchmaking mechanism to cater for web services composition. Developing an intensive knowledge-based substitution to adapt the functional and non-functional attributes of the candidate case to the requested solution and planning based transformation to integrate the service profile of a number of cases to deliver candidate solutions.

4. Evaluation

The evaluation of the framework is in two categories: qualitative and quantitative.

- The qualitative evaluation answers the research questions we had outlined in our motivation and contrasts them to what we have achieved in this research.

- For the quantitative evaluations, we evaluate our Web services discovery and matchmaking framework on precision and recall along with execution time performance.

1.6. Thesis Structure

This chapter introduced the background topics related to Web services composition. In summary, we argue that XML-based Web services and XML-built Semantic Web are the driving technologies behind automatic application services composition. The composition achieved in this way has the potential to assist the service participants and to automate service discovery and composition process tasks.

Chapter 2 consists of a literature survey focusing on existing Web services composition approaches. The industrial standards based on workflow management theory and research efforts based on semantic web are addressed.

Chapter 3 reviews the prominent workflow based standards for composition and those that use semantics. The limitations and advantages of such efforts are discussed and a

framework is presented that utilizes semantics within the static web services composition standard – BPEL.

Chapter 4 discusses the Case Based Reasoning (CBR) methodology for modelling dynamic Web services discovery and matchmaking. The problems encountered during development and their solutions have been identified. Experimental results are discussed.

Chapter 5 explores solution case adaptation to address limitations of the framework described in Chapter 4. The process of case adaptation is applicable when the available cases cannot fulfil the problem requirements, so matchmaking is attempted by adapting available cases. The chapter outlines process of adaptation to address the limitation of SCBR regarding limited intelligence and extend the framework for Web services composition. The resultant framework of XSCBR is presented in this chapter.

Chapter 6 is devoted to the implementation and evaluation of the XSCBR framework for Web services discovery and composition. This chapter also represents results of the experiments.

Chapter 7 summarises the contribution of the thesis and critically analyses the achieved results and suggests the directions for further research.

2 Chapter Two: Literature Survey

This chapter presents a literature survey of current Web services composition approaches. The study shows that these approaches fall under two categories. The first category, largely endorsed by the industry, borrows from business processes' workflow management theory to achieve the formalization necessary for describing the data flow and control flow in the composition scheme. The second category, mainly promoted by the research community, aspires to achieve dynamic composition by semantically describing the process model of Web service and thus making it comprehensible to reasoning engines or software agents.

The chapter reviews the above approaches to analyze their impact on the application of Web services composition.

2.1. Workflow management theory based approaches

Workflow is the movement of documents and/or tasks through a work process. More specifically, workflow is the operational aspect of a work procedure: how tasks are structured, who performs them, what their relative order is, how they are synchronized, how information flows to support the tasks and how tasks are being tracked [26].

Workflow management systems are a class of information systems that make it possible to correlate people's work and computer applications. Such systems deal with the control flow (invocation sequence of applications) and data flow (information flow between applications) while control flow is important for achieving overall system objective, data flow is essential for the successful operation of individual applications.

In the information systems domain, workflow has been used since the 1970's for the office automation systems [27]. This work has lead to identifications of workflow patterns for control and data flow. Table 1 outlines basic workflow patterns [26]:

Table 1 Workflow Patterns

Category	Type of patterns	Details
Control flow patterns	Sequence	Execute activities in sequence
	Parallel Split	Execute activities in parallel
	Synchronization	Synchronize two parallel threads of execution
	Exclusive Choice	Choose one execution path from many alternatives
	Simple Merge	Merge two alternative execution paths
Data flow patterns	Task Data	Data elements can be identified by tasks which are accessible only within the context of individual execution instances of that task.
	Block Data	Block tasks (i.e. tasks which can be described in terms of a corresponding sub-workflow) are able to define data elements which are accessible by each of the components of the corresponding sub-workflow.
	Scope Data	Data elements can be defined which are accessible by a subset of the tasks.
	Multiple Instance Data	Tasks which are able to execute multiple times within a single workflow case can define data elements which are specific to an individual execution instance.
	Case Data	Data elements are supported which are specific to a process instance or case of a workflow. They can be accessed by all components of the workflow during the execution of the case.

One of the applications of workflow management in information systems domain is to address the Business Process Management (BPM) problem. Business process can be considered as workflow of business activities to carry out business goals [28]. The examples of business activities for customer order fulfilment business process are: customer placing an order, checking account status, verifying order and despatch. Using Workflow management, BPM deals with achieving the integration of these individual applications.

Business processes can have scope within inter and intra organization relations. Enterprise Application Integration (EAI) is the BPM solution to achieve intra-organization business applications integration, while Business-to-Business (B2B) integration software addresses the problem for inter organization business application integration. Traditional EAI and B2B integration solutions are very complex, proprietary and presume many details about the participating applications making them tightly coupled. For instance, these solutions assume the use of homogeneous service interfaces and implementation technology, which is a substantial limitation considering that different organizations will make independent decisions about what technology to use for the construction and deployment of their parts; these decisions made over time accrete different hardware and software technologies [29]. Tightly coupled systems are difficult to manage and re-engineering business rules and requirements in such systems is also challenging. To overcome these limitations, business applications are now being developed using Web services while the BPM problems (EAI, B2B) are being addressed with the workflow based integration of Web services, mainly to utilize SOA based Web services features [30].

The main industrial standards to achieve workflow based integration of Web services are WS-BPEL[1] (Web Services Business Process Execution Language, shortened to BPEL) [31], WS-CDL (Web Services Choreography Description Language, shortened to CDL) [32] and BPML (Business Process Modelling Language) [33]. The service description specification WSDL plays a major role in achieving web services integration in these composition specifications as they take advantage of the fact that WSDL describes how to communicate with a given Web service and includes details such as the definition of available operations, variable formats, service URI and messaging formats. The workflow-based process model for these approaches also addresses requirements for describing flow-management in composition, handling business transaction with roll-back facility, state management for business interaction support, and also handling exception and errors. The category of process model and the extent to which these features are provided differentiates these standards.

The following sections outline two prominent workflow-based industrial standards for Web services composition.

[1] WS-BPEL version 1.1 , WS-CDL version 1.0

2.1.1. Composing services using BPEL

The BPEL specification - enhances and replaces the existing standards Web Services for Business Process Design (XLANG) [34] from Microsoft and Web Services Flow Language (WSFL) [35] from IBM. The specification uses workflow management as a process model to achieve the control and data flow formalization for WSDL-defined data and operations. All the participant services in a BPEL process are modelled as partners. The WSDL files of such partners are required to create BPEL process. The partners contribute to the total processing capability of the BPEL process. BPEL process also has its own processing capability for dataflow, control flow, data manipulation, fault and event handling and state management. The significance of the BPEL architecture is that the process itself is published as a Web Service. This composed BPEL service can be treated as a single Web service and can be used for further composition hence facilitating recursive composition.

Facilitation provided to the service participants

In order to evaluate the facilitation provided to the service participants we consider a scenario based on travel agent service, which manages the reservation of airline and hotel for a customer trip. The travel agent can be implemented as BPEL process, which can be a composition of four Web services: *AirFrance* service, *AirUSA* service, *HotelRating* service and *HotelService* service. The process logic for the travel agent is: "to check the availability of flight service from two competing airlines *AirFrance* and *AirUSA*, make flight reservation, and then retrieve hotel ratings from the *HotelRating* service at the destination city and make the reservation using HotelService Web service at the selected hotel".

For a new service provider to make their service available for the above composition service they need to provide minimum functionality consistent with the business logic outlined by the travel agent which is essentially the composer. Let's assume a new *AirUK* flight service for travel agent composition, the *AirUK* service provider has the following options:

a) If the *AirUK* application is not exposed using a Web service, a wrapper Web service with a compatible WSDL file can be created without modifying existing application. BPEL execution engine uses Web Services Invocation Framework [36]

for the Invocation of such non web-services. This provision is a useful assistance to the service provider which can re-use their legacy systems and can take part in composition efforts.

b) If the *AirUK* provider has a service already available with conceptually similar but syntactically different parameter structure then the BPEL specification provides no form of assistance to the provider. The scenario similar to *AirUK* has relevance in the real-world applications and the omission to address them is a major drawback for BPEL specification.

Considering the case of the service composer who for the most part encounter problems in parameter mismatch during the flow management, i.e., a service operation has different output format from the input of next service operation in the flow logic, BPEL in its current form delegates the responsibly to the service composer to address such parameter mismatch.

From a service requestor point of view, the travel agent BPEL process could be published using JSP technology. This way the service can be retrieved using simple web page or WSDL file for the composed Web service can be retrieved from the public UDDI registry. In such B2C interactions it is totally transparent from the end-user that the service is a Web service with the possibility of composition of multiple Web services or could be implemented on heterogeneous platforms using heterogeneous programming languages. However, there is a limited level of language expressiveness available to the service requestor to outline the constraints and preferences on the outputs and quality of service parameters.

To conclude this section, BPEL is widely-used specification for composing intra-organization Web services. The business analysts and developers can collaborate and can compose enterprise Web services manually using BPEL. The composition is hard coded and the developers should have the explicit knowledge of all the details of participating business services which is a major limitation considering the growth of Web services within and outside organizations.

2.1.2. Composition using WS-CDL

The BPEL process model deals with B2B integration from a single party viewpoint i.e., the requirement specified for the travel agent scenario discussed here is from the

viewpoint of travel agent business logic. Contrary to the BPEL process model, real world B2B integrations are peer-to-peer as opposed to being centralized, where the collaborating business applications agree to provide certain functionality in receipt of complimentary functionality from other business applications highlighting the requirement for a description language documenting peer-to-peer viewpoint since natural B2B integrations are peer-to-peer collaborative relationships and not governed by a single party. The W3C recommendation WS-CDL[2] from W3C Web services choreography working group confirms aforementioned conclusions that more work on BPEL is required to make it adoptable for B2B integration [32] .

WS-CDL is a description language where the first activity of the B2B integration partners is to describe the collaborative functionality. This description document is considered as a contract and each party can implement their own part. The WS-CDL document describes common and complementary behaviour of all the parties involved, making the viewpoint global and peer-to-peer [33]. For example, under WS-CDL process model, the travel agent is no longer the overall controller of the integration of the travel service as the agent and the service providers have to be involved and agree on the composed functionality and ordering of the activities in the WS-CDL document. The other aspect of WS-CDL process model is that the internal business logic of each party remains hidden from the business partners. i.e., for the travel agent application after receiving price quote from all airlines can have internal business logic for air line selection based on some criteria totally hidden from other partners as the external detail described in WS-CDL document is just an operation to make reservation at particular airline.

Facilitation provided to service participants

Service composer designs the global interface WS-CDL file to be adhered by participating parties. Therefore the composer does not have to deal with individual service providers and can easily accommodate individual services once providers adhere to the global interface.

WS-CDL is still a descriptive language but can play the role similar to WSDL to create stub files so that each party service provider can have blue print of what they are supposed to implement [37]. This approach has considerate benefit when the integration

[2] WS-CDL was a draft version when this research was carried out.

takes place between large numbers of Web services. Overall, CDL is designed to address the requirements for B2B integration and compliments BPEL. Consequently, CDL and BPEL together address the problem of BPM by facilitating static composition as the selection of services and decision on flow management is done a priori.

2.1.3. Adoption of Workflow-based approaches

Despite offering static composition, commerce and industry remain loyal to the workflow-based composition for integrating services within the enterprise and for forging B2B collaboration. The success of BPEL and CDL in the business community can be attributed to number of factors. Firstly, the standards are built on the top of tried and tested workflow management theory, making it ideal to model business processes' interaction. The second factor is that BPEL and its derivatives are now mature standards that provide a gamut of features for business processes, such as transaction processing, support for state management with the use of call backs and correlation sets, provision for exception handling, compensation fault processing features that are vital for the long running and fault vulnerable business transactions.

The adoption of BPEL as the Web service composition technology of choice has been reflected in the enthusiasm at large software houses in providing or including BPEL composition tools in their Enterprise Application Servers, for instance Oracle Application server [38], Microsoft BizTalk server [39], and stand-alone tool from IBM, BPWS4J [40].

2.2. Semantic Web-based Composition

The commercial institutions are focusing their efforts on standardizing the static composition techniques in preparation for their wider adoption amongst the business community. In contrast, the research community efforts concentrate on exploiting semantic web for the semi-automatic and automatic composition of Web services.

2.2.1. Semantic Web services

With respect to automation, the limitation of workflow-based approaches is that they rely on WSDL based description for the Web services selection. WSDL is a static interface described using simple XML grammar that has no notion of machine interpretable semantics. The problem of automatic Web services discovery and

integration can benefit from the semantic web machine readable descriptions. The fundamental premise of the semantic web is to extend Web's currently human-oriented interface to a format that is comprehensible to software programmes. Applied to Web services composition, this can lead to the automation of services selection and execution.

The WSDL file of Web services describes the operations provided, request message format required for invoking operations, and the format of response messages produced by the Web services. The interpretation of these details results in the understanding of the service capability. The automation required for the service composition can be achieved by describing the WSDL elements semantically, thus allowing software agents to reason about the service capability, and make all the decisions related to the composition on behalf of the user or developer. The decisions include the selection of appropriate services, their actual composition and close examination of how they meet the criteria specified by the user. In contrast, in the static composition approach, the user or developer manually interprets the requirements for the required composition and the available service capability or functionality and makes decisions regarding how services can be interweaved to make a value-added service.

The WSDL specification is part of the base Web services protocol stack and has been already widely accepted and implemented to describe Web services. Taking this into consideration, the general scenario will be to annotate individual WSDL elements with corresponding OWL elements. OWL-S [21] is such ontology specification for describing Web services semantically. OWL-S ontology provides a mechanism to describe the capability of Web services in machine-readable form, which makes it possible to discover and integrate Web services automatically.

2.2.2. Semantic Mark-up for Web services: OWL-S

OWL-S defines three interrelated subontologies, known as the profile, process model and grounding. In brief, the profile is used to express "what a service does", for the purpose of advertising, constructing service requests and matchmaking; the process model describes "how it works", to enable invocation and composition; and the grounding maps the constructs of the process model onto detailed specifications of message formats, protocols and so forth [21]. Figure 5 outlines these subontologies.

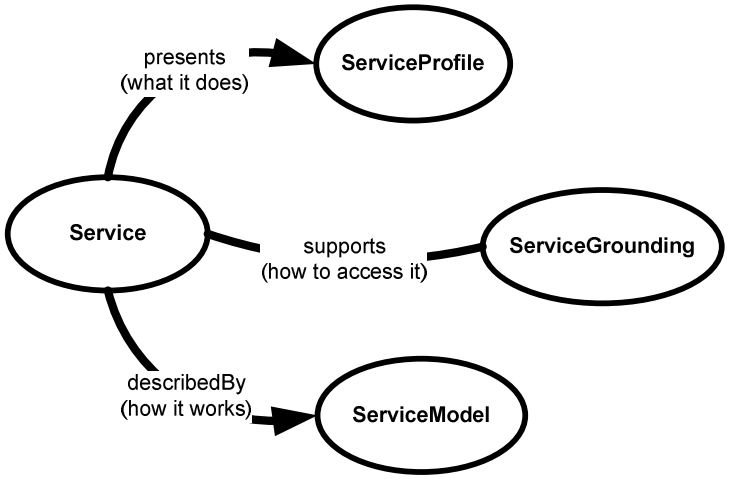


Figure 5 OWL-S subontologies

Service

The service class acts as an organizational point of reference for OWL-S descriptions. Each Web service description will provide a single instance of *Service* with corresponding values for *presents*, *describedBy* and *supports*. The respective ranges of these properties are *ServiceProfile*, *ServiceModel* and *ServiceGrounding*. Each of these represents different level of information regarding the Web services. These classes are introduced as follows.

Service Profile

The service profile is the advertisement of web services by describing what it actually does. Apart from incorporating UDDI-like elements (taxonomy, category, human readable description of service) a service profile has the notion of IOPE (Input, Output, Precondition, Effects), where the input and output are OWL elements describing expected Web service input and generated outputs. Furthermore, since a service may require external conditions to be satisfied, and it has the effect of changing such conditions, the profile describes the preconditions required by the service and the expected effects that result from the execution of the service. For example, a selling service may require as a precondition a valid credit card and as input the credit card number and expiration date. As output it generates a receipt, and as effect the card is charged. Such semantic descriptions assist the discovering party to make sure of their choice, by interpreting what inputs need to be provided to invoke the service, what conditions need to be fulfilled to invoke the service and what will be the output and

effect of the invocation. Using such descriptions software agents can visualize the effect of service execution before actually executing it.

Service Model

The service model assists the service requestor in service executions, as the *process* subclass of `ServiceModel` describes the possible interactions requestor can make with the service. As shown in Figure 6, processes can be atomic, simple or composite. An atomic process is a description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response. A simple process is similar to atomic except it is abstract and can provide multiple views of the same process. A composite process can be decomposable into atomic or other composite process and can be described using the rich semantics of a service model which supports control-flow and data-flow patterns similar to workflow patterns.

```
<owl:Class rdf:ID="Process">
  <rdfs:comment> The most general class of processes </rdfs:comment>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicProcess"/>
    <owl:Class rdf:about="#SimpleProcess"/>
    <owl:Class rdf:about="#CompositeProcess"/>
  </owl:unionOf>
</owl:Class>
```

Figure 6 OWL-S Process

The service model re-assures the requestor about his choice as the software agents can read the process model descriptions and can interpret working of Web service; hence can decide the applicability of the solution.

Service Grounding

A WSDL document contains the description of the invocation details for the Web services. OWL-S enhances the WSDL based service description to accommodate semantics hence the invocation details in WSDL need to be mapped to the semantic description in OWL-S. OWL-S achieves such mapping with `Grounding` component. In Figure 7 the dotted line indicates the possible mappings between OWL-S and WSDL.

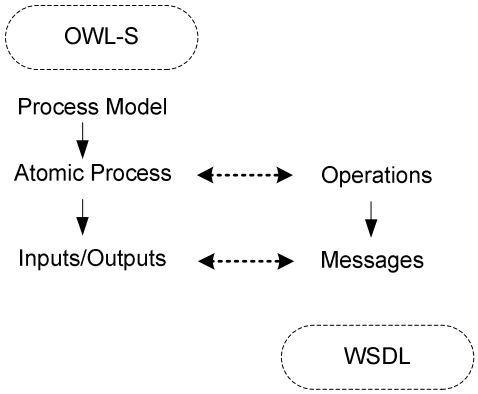


Figure 7 Mappings between OWL-S and WSDL

According to these mappings, an OWL-S process corresponds to WSDL operation. The set of inputs and the set of outputs of an OWL-S atomic process each correspond to WSDL's concept of message. More precisely, OWL-S inputs correspond to the parts of an input message of a WSDL operation, and OWL-S outputs correspond to the parts of an output message of a WSDL operation.

The OWL-S based approach facilitates the meaningful searches with the advantage of (IOPE) in profile and process based service model hence user can perform in-depth analysis of multiple services to perform a specific task.

2.2.3. Reasoning about the Service Semantics

Ontology-based descriptions provide a mechanism to describe Web services functionality and the information useful for composition to be encoded in unambiguous machine understandable form. In order to perform the automated composition, an intelligent layer is essential that can interpret semantic descriptions and can order, combine and execute Web services to achieve the desired functionality or user goals. In other words, the intelligent layer should comprehend the descriptions in order to decide the possible services and build flow management for those services.

The semantics based approaches can be categorized based on the intelligent layer employed to achieve Web services discovery and composition. AI planning, software synthesis, agents, constraint satisfaction problem and case based reasoning are some of the methodologies employed as intelligent layer.

Artificial Intelligence Planning

This section discusses the relevancy of AI planning for the Web services composition problem and presents the literature survey on the subject.

Planning is a task of discovering a sequence of actions that can achieve a goal [41]. A planning problem can be described as a five-Tuple problem ***(S,s0,G,A,T)*** where ***S*** is the set of all possible states of the world, ***s0*** denotes the initial state of the planner, ***G*** denotes the set of goal states the planning system should attempt to reach, ***A*** is the set of actions the planner can perform in attempting to reach a goal state, and the transition relation ***T*** defines the semantics of each action by describing the state (or set of possible states if the operation is non-deterministic) that results when a particular action is executed in a given world state.

Web services composition is similar to planning problem evident from the following mapping.

S is the set of possible Web services, i.e. Web services available from the service registry
s0 is the initial state where some or none services are pre-selected for composition
G is the composition of Web services which satisfies the user requirements.
A is the Web services operations (I) or preconditions (P) available to planner to reach from the initial to goal state
T is the outputs (O) and effects (E) of invoking Web services operations.

AI planning-dependent approaches use IOPE based OWL- S profile and process models to achieve required automation for the Web services composition. For example, if one starts with composition as goal (some desired outputs and effects), and matches it to the outputs and effects of a Web service (modelled as process), the result is an instantiation of the process, plus descriptions of new goals to be satisfied based on the inputs and preconditions of that process. The new goals (inputs and preconditions) then naturally match other processes (outputs and effects), so that composition arises [21].

Consistent with the above theory, Wu *et al.* [42] utilize DAML-S based descriptions, the previous version of OWL-S with SHOP2 planner [43]. The SHOP2 is a Hierarchical Task Network (HTN) planner that creates plan by task decomposition - a process in which the planning system decomposes tasks into smaller and smaller subtasks, until primitive tasks are found that can be performed directly. The authors stress similarity

between the concepts of task decomposition in HTN with the process decomposition in DAML-S.

Sirin *et al.* in [44] describe another approach which couples OWL reasoner with AI planner to reason about the world state (effects and pre-condition) during planning. The reasoning is achieved by describing pre-condition and effects of the Web services using OWL. Peer *et al.* in [45] follows similar approach but argues that the diversity of Web service domains can best addressed by a flexible combination of complementary reasoning techniques and planning systems. Authors present a tool that transforms Web service composition problems into AI planning problems and delegates them to the planners most suitable for the particular planning task. The tool uses the planning domain definition language (PDDL) [46], a language supported by a wide range of planning engines as required transfer format.

In similar spirit, McIlraith *et al.* [47] use GOLOG (logical programming language) for the planning based Web services composition. GOLOG [48] is a high-level logic programming language, developed at the University of Toronto, for the specification and execution of complex actions in dynamical domains. The GOLOG based system models services as actions with IOPEs and uses GOLOG procedures (modelled as OWL-S composite processes) to generate sequences of Web services customized to user's preferences and constraints.

The semantic web community draws on AI planning, which for over three decades has investigated the problem of how to synthesize complex behaviours given an initial state, an explicit goal representation, and a set of possible state transitions [49]. However, the main drawback of the AI planning techniques is the difficulty in dealing with incomplete information, in Web services composition problems the extensional definition of the initial world does not specify all knowledge relevant to the planning task [45]. For instance, in an e-commerce application, the travel agent may not know which web services offers which products, but it needs this information to achieve its goal of buying a product.

Software synthesis

Software synthesis refers to the problem of creating complex software system from individual software components. The approaches modelling Web services composition

as software synthesis problem view atomic services as software components and composed services as synthesised complex software system.

The work of Matskin *et al.* [50] models problem description and the available service's descriptions into Structured Synthesis Program (SSP) [51] – a software synthesis technique which supports input-output specifications with the possibility of extraction of action sequences. The approach supplies the problem statements to the SSP synthesiser with the available service lists and allows SSP to prepare a plan to reach from problem descriptions to a sequence of actions to be performed in order to achieve a viable solution.

Consistent with the work presented above Rao *et al.* in [52] discusses the use of the software synthesise formalism: Linear Logic (LL) for Web services composition. The implementation translates Web services description into LL axioms which are fed to LL prover to generate proof or plan for Web services composition.

These efforts suggest a seamless mapping between software synthesise and service composition, however they treat each service as an atomic entity without inspecting the internal process model and therefore lacks the ability to measure full capacity of services.

Agents

Web services are compositional, independent software components similar to agents. In addition agents are also social, reactive and capable of reasoning [53]. This autonomous and reasoning capability of agents makes them suitable to apply for the problem of automatic Web services composition.

To benefit from agent features, a variety of approaches convert web services to work as agents where one of the options for conversion can be as wrapper mechanism. Buhler *et al.* in [54] apply similar approach which creates agents from Web services using composition language. The agents created in this manner have the reasoning capability derived from the DAML-S descriptions making interaction possible between agents to decide if they can collaborate to fulfil the ultimate goal of composition. Knoblock *et al.* in [55] outlines similar approach, where they have developed tool for web services-to-agent conversion and uses hierarchical constraint system to perform integration.

The work by Richards *et al.* documented in [56] applies and extends Agent Factory - an automated facility for composing software agents, to use Web services as agent components. Their implementation use the DAML-S profile models to provide descriptions of the components at the conceptual level for the discovery and the grounding model to provide the descriptions at the implementation level for the integration.

Although software agents have being researched since 1977 [57], an inherit limitation is that the autonomous nature of agents requires extra safeguards so that agents do not overstep their jurisdiction. Another limitation is the necessary conversion from Web services specification to the agent platforms such as AgentCities which can be computationally expensive.

Case Based Reasoning

Experience based learning using CBR is a relatively old branch of artificial intelligence and cognitive science and is being used as an alternative to rule-based expert system for the problem domains, which have knowledge captured in terms of experiences rather than rules [58]. Case based reasoning for Web services were initially documented in [59], where the developed framework uses CBR for Web services composition. In their approach, the algorithm for Web services discovery and matchmaking is keyword based and has no notion for semantics. This affects the automation aspects for Web services search and later for composition. A similar approach described in [60] proposes an extension of UDDI model for web services discovery using category-exemplar type of CBR, where web services are categorized in domains and stored as exemplar [61] of particular domain. Their implementation of CBR reasoner facilitates UDDI registry by indexing the cases based on the functional characteristics of Web services. However, the approach does not take into consideration the importance of non-functional parameters in service selection and the use of semantics at CBR level is peripheral as they primarily use the UDDI based component for service discovery. UDDI is text-based leaving little scope for automation.

There is also a number of existing approaches which apply CBR for workflow modelling. [62] proposes an approach to support workflow modelling and design by adapting workflow cases from a repository of process models where workflow schemas are represented as cases and are stored in case repositories. The cases are retrieved for a

problem which requires similar business process to solve the problem. The description and implementation language of the framework is based on XML and its main focus is on assisting workflow designer in creating business process flows. Similarly, [63] presents an adaptive workflow management system based on CBR and targets highly adaptive systems that can react themselves to different business and organization settings. The adaptation is achieved through the CBR based exception handling, where the CBR system is used to derive an acceptable exception handler. The system has the ability to adapt itself over time, based on knowledge acquired about past execution experiences that will help solve new problems.

These approaches fail to take advantage of the main feature of CBR that is storing past experiences of current problem for solving future problems. Web services execution experiences can be represented as cases and once stored can be utilized to serve service requests.

Constraint Satisfaction Problem

Constraint Satisfaction Problem (CSP) [64] is a powerful and extensively used AI paradigm. CSP involve finding values for variables subject to restrictions on which combinations of values are acceptable. The approaches that utilize CSP take advantage of the fact that constraints on selection parameters play a major part in discovery and composition of Web services. In accordance with this analysis, authors in [65] argue that Web service composition in real life requires not only planning information, but also additional information requests with constraints, which can be met by scheduling tasks jointly. The authors suggest a combined architecture of planning and CSP for a basic problem-solving engine to automate Web service composition giving an entire framework of intelligent Web services for users.

Another approach as documented in [66] relies on CSP for solving Web services composition problem where authors represents a constraint driven Web service composition tool in METEOR-S framework, which allows the process designers to bind Web services to an abstract process, based on business and process constraints and generate an executable process. The METEOR-S project utilizes semantics with existing Web services standards of WSDL, UDDI and BPEL to support publication, discovery and composition of semantic Web services.

The main advantage CSP offers is its inherent ability to consider constraints; which is essential to serve granular service requests. However, CSP alone is not sufficient to address the composition problem as the methodology does not support planning-like work-flow management. Nonetheless, when used with other workflow or planning based architectures, CSP can be considered as an attractive approach to Web services composition.

2.2.4. Potential Facilitation to the composition participants

Despite the enthusiasm of the research community about the semantic web, there is still some way to go for creating a unifying framework facilitating the interoperation of intelligent agents or reasoning engines attempting to make sense of semantic Web services. However the workflow based approaches address here-and-now practical problem of Web services composition while dynamic Web services composition approaches holds better future potential that can serve a great range of business domains. Automatic Web services composition has the potential to reduce development time and effort for the development of new applications. This is due to automatic re-configuration of changing or unavailable services in the integration.

Semantics assisted dynamic composition can serve all business domains for the possible B2B, EAI and B2C integrations. A user can specify parameters for the successful composition and the composition can be performed at the run-time. The automatic Web services composition solution can address the problems of identifying candidate services, composing them, and verifying closely that they satisfy the request.

The service providers will be able to participate in the composition to their benefit with minimal effort as the development effort will be significantly reduced, as the human developer will be taken out of the composition loop.

2.3. Evaluation of Composition Techniques

For our research objectives, we have chosen the following criteria to study existing Web services composition approaches.

1. Service matchmaking

Using this evaluation criterion we compare various approaches based on how the service matchmaking is performed. The possible options are discovery using WSDL, UDDI, free-text or OWL-S (previously DAML-S) profile and process.

Workflow-based approaches use WSDL files to interpret the capability of a service coupled with the communications with the service provider or manual analysis of service parameters. AI planning, CSP, and agent-based approaches use different algorithms that utilize semantic web services profiles to matchmake with semantically-encoded problem requests. The CBR based approaches are so far using UDDI to matchmake web services.

2. Composition

We use this criterion to compare existing approaches to evaluate them based on how they employ intelligent layers to achieve composition of Web services.

Workflow-based approaches use web services workflow languages such as BPEL and WS-CDL to outline the workflow of Web services. AI planning-based approaches utilize AI planner to form composition plans using existing planners such as SHOP2 [48] or GOLOG [43]. The CSP based approaches utilize existing standards WSDL, UDDI and BPEL to achieve the required composition. The CBR based approaches use bespoke XML based workflow languages to write composition schema. The Agents-based approaches model web services as agents so that the problem of web services composition translates to agent collaboration problem so that it is possible to utilize existing agent-infrastructure for composition.

3. Automation

The automation criterion is used to measure the level of automation achieved by various Web services composition approaches in the process of service discovery, composition and execution.

Most of these approaches support execution of composition schemes by providing execution engines, i.e., BPEL approaches use Oracle BPEL PM execution engine or IBM BPWS4J, AI planners use OWL-S execution engines similar to the OWL-S API provided by the University of Maryland.

Workflow-based approaches are static web services composition approaches involving manual intervention for discovery and composition of services. Semantic web based approaches achieve varying degree of automation in the process of composition (automatic discovery, semi-automatic composition).

4. Transparency

This criterion measures how transparent the process of composition (discovery, integration and execution) is from the composition participants. For workflow-based approaches, end-user is transparent from the fact that the service presented to them in response to their request is a composed service, however the provider and composer has to work closely to integrate services in the workflow hence making the process opaque to them.

For AI planning-based approaches, the service requestor is transparent to the intelligent process of composition; however the process is semi-transparent to other participants. For example, the composer needs to be involved in the process of domain knowledge development and maintenance while tools assist them in converting semantic web services processes into planner domains. This knowledge is supplied to the planner in terms of operators and methods of services in order for planner to build composition plans. The service provider has to provide semantically enabled service but is transparent from the process of composition. Similarly, other semantic web based approaches offer complete transparency to end-users while requires some level of attention from service providers and composers.

5. Extensibility

The extensibility criteria measure how extensible particular approach is to adapt new mechanism or to add new functionalities. For example, workflow based standards can be evaluated based on whether they can include semantics to solve semantic issues in service selection [66][67] and [68] are the approaches that seek the answer and determine that extension is possible for service matchmaking; however a BPEL-based integration mechanism is tightly coupled by nature and offers limited level of extensibility. Various other approaches are extensible as they are already adapted from the existing AI methodologies.

6. Expressiveness

The problem of dynamic Web services composition requires a greater level of expressiveness for describing services and for describing search criteria of services. Hence, the functional parameters of IOPE are sometimes not sufficient to achieve the goal of automation and require non-functional descriptions of services; for example, a Web service can be selected based on the Quality of Service (QoS) it provides. We also analyze the expressiveness in terms of the support service requestor gets in order to describe service criteria closer to natural language; for example, preferences and constraints on various output and other non-functional parameters.

The BPEL specification has no scope to accommodate non-functional parameters beyond IOPE (Input, Output, Precondition, and Effect) due to the absence of provision for syntactical non-functional parameters in the specification.

The OWL-S specification supports set of non-functional properties: service name, text description, quality rating. The specification also has provision for the other non-functional properties using the `ServiceParameter` from `ServiceProfile`. These non-functional properties are described in the service profile part and are explicitly formalized using OWL. The approaches that utilize OWL-S (AI Planning, software synthesis, CSP, software agents) as semantic web services specification exploit these provisions at varying degree for formalizing non-functional parameters. However, we notice that there is no provision or modelling support for allowing service requestor to describe their request in greater detail and are just limited to semantics of required service (parameters and parameter types).

The Universal Description, Discovery and Integration (UDDI) define a set of non-functional properties for a service provider identified by a `businessEntity`. The set of non-functional properties contains: the address, the phone numbers, and the email addresses of the service provider. Additionally to non-functional properties some other information (metadata) about the service is available like for example the service category (using taxonomies such as UNSPSC). Approaches that utilize UDDI (CBR based approaches) are utilizing some of these non-functional parameters to provide a limited level of expressiveness.

7. Scalability of composition

Composing a large number of services can incur significant overhead on the response time to the end user. In a real-world scenario, end users will typically want to interact with many services; for example, if we consider the classic holiday booking scenario where enterprise applications invoke chain of possibly several hundred services [69].Therefore, one of the critical issues is how the proposed approaches scale with the number of services involved. In BPEL, multiple service composition is somewhat tedious because XML files start to grow offering the approaches relying on BPEL as final composition scheme limited scalability (CSP based approach). OWL-S has similar issues and is propagated to the approaches that rely on using OWL-S process as final composition scheme (i.e., AI planning, software agent). Approaches that utilize bespoke XML schemas for final composition scheme (i.e., software synthesis approaches output synthesized XML schemas) also face similar challenges.

8. Knowledge utilization

Semantic Web is utilized to capture and reason knowledge within organizations and the WWW. However because of the distributed and open nature of the Web, these ontologies can be expected to contain conflicts and semantic overlap; different ontologies would describe (parts of) the same domain in a different way, because of differences in the point of view of the different people who have developed the ontologies. This clearly relates to any approaches that apply semantic web to Web services composition. For example, service requestor and service provider might use different ontologies to describe conceptually similar concepts; similarly composer might need to deal with providers using different sets of ontologies.

The knowledge utilization criteria evaluates various approaches based on whether they provide means to mediate various ontologies and knowledge sources while achieving Semantic Web based discovery and composition.

Table 2 summarizes the comparison of existing Web services composition approaches based on the aforementioned criteria.

The comparison also focuses on the methodology each approach uses to achieve matchmaking and composition and how selected methodology affects the prospects of automation and transparency.

Table 2 Comparing Intelligent Layer approaches to Web services composition

	Workflow based Approaches	**Semantic Web based Approaches**				
Criteria	BPEL based Web services composition	AI planning based approaches	Software synthesis based approaches	CBR based approaches	CSP based approaches	Software Agent based approaches
Service matchmaking	Using WSDL and choreography interfaces	OWL-S/ DAML-S profile matchmaking		UDDI based	OWL-S profile templates, BPEL abstract processes	DAML-S profile matchmaking
How does it affect automation and transparency prospects?	No consideration of semantics in service descriptions hence prospects for automation and transparency is affected.	Consideration of semantics in service descriptions makes automation possible; however accuracy of matchmaking process has scope for improvement		No consideration of semantics in service descriptions hence prospects for automation and transparency is affected.	Consideration of semantics in service descriptions makes automation possible; however accuracy of matchmaking process has scope for improvement	
Composition	Using workflow patterns in BPEL or WS-CDL	Using AI planner	Using software synthesise methods (i.e., SSP, Linear Logic)	XML based workflow language	Converting abstract BPEL process to executable BPEL process	Composition as agent collaboration or interaction
How does it affect automation and transparency prospects?	Lack of semantics in workflow process model limits prospects for automation.	Exploitation of semantics in process model with planning techniques leads to semi-automation	Exploitation of semantics only in OWL-S profile but not in process model leads to limited automation	Lack of semantics in workflow process model limits prospects for automation.	Lack of semantics in workflow process model limits prospects for automation.	Exploitation of semantics only in OWL-S profile but not in process model leads to limited automation

	Workflow based Approaches	**Semantic Web based Approaches**				
Criteria	BPEL based Web services composition	AI planning based approaches	Software synthesise based approaches	CBR based approaches	CSP based approaches	Software Agent based approaches
Level of automation	Predefined, static workflows, Automatic Execution	Automatic matchmaking, semi-automatic composition, execution using grounding	Automatic matchmaking, semi-automatic composition, execution using grounding	Static matchmaking, semi-automatic composition, automatic execution using workflow engine	Automatic matchmaking, Semi-automatic composition	DAML-S Web services as agent components and composition as agent collaboration or interaction
Transparency	End-user transparent, provider and composer aware of the process	Transparent to service requestor, semi-transparent to composer and provider	Transparent to service requestor, semi-transparent to composer and provider	Transparent to service requestor, semi-transparent to composer and provider	Transparent to service requestor, semi-transparent to composer and provider	Transparent to service requestor, semi-transparent to composer and provider
Extensibility	Extension is possible for service matchmaking, However integration mechanism tightly coupled hence limited extensibility	Extensible (for example, to include knowledge, non-functional properties)	Extensible to include workflow models	Extensible	Extensible to adapt any workflow standards (BPEL, OWL-S).	Applicable to agent platform only.
Expressiveness (Non-functional parameters)	No support for functional parameters	Exploits provisional unspecified support for non-functional parameters from OWL-S, no support for requestor search criteria	Exploits provisional unspecified support for non-functional parameters from OWL-S, no support for requestor search criteria	Limited support using UDDI specification.	Exploits provisional unspecified support for non-functional parameters from OWL-S, no support for requestor search criteria	No support
Scalability of solution	Difficult to manage scalability	Difficult to manage scalability	Difficult to manage scalability	Difficult to manage scalability	Difficult to manage scalability	Difficult to manage scalability
Knowledge utilization	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported

2.4. Conclusions

This chapter surveyed two prominent categories of Web services composition approaches. The first approach, largely endorsed by the industry, borrows form business processes' workflow management theory to achieve the formalization necessary for describing the data flow and control in the composition scheme. The second approach, mainly promoted by the research community, aspires to achieve more dynamic composition by semantically describing the process model of Web service and thus making it comprehensible to reasoning engines or software agents.

The comparison made in this chapter has shown that workflow based approaches are preferred by organizations as here-and-now and practical, albeit static, composition technique that robustly supports their business needs; while dynamic Web services composition approaches holds better future potential that can serve a great range of business domains. In such kinds of composition participating services can be external and public. The user can specify parameters for the successful composition and the composition is performed at the run-time. The solution addresses the problems of identifying candidate services, composing them, and verifying closely that they satisfy the request.

As the result of this literature survey we concluded that despite the enthusiasm of the research community about the semantic web, there is still some way to go for creating a unifying framework facilitating the interoperation of intelligent agents or reasoning engines attempting to make sense of semantic Web services independent of human developer.

3 Chapter Three: Bridging Gap between Workflow and Semantics based Web Services Composition

The previous chapter discussed prominent Web service composition approaches. This chapter discusses the advantages and limitations of workflow and semantics-based approaches and outlines a hybrid approach that takes advantage of both by introducing semantics to workflow-based composition.

Despite the enthusiasm of the research community about the semantic web, there is still some way to go before creating a unifying framework facilitating the interoperation of intelligent agents or reasoning engines attempting to make sense of semantic Web services.

In large, efforts to facilitate automatic composition web description through semantic description have been progressing in parallel, but also in isolation, to developments in workflow-based standards (specifically BPEL) preferred by the commercial organizations. These organizations prefer a here-and-now and practical, albeit static, composition technique that robustly supports their business needs, to immature, research-biased, dynamic composition techniques that are more focused on the automation factor, rather than business-specific requirements.

The hybrid approach discussed in this chapter concentrates on exploiting industrial standards with the possibility of using semantics, before attempting a full-fledged semantics-based solution. This approach has the merged benefit of practicality of use and adoption popularity of workflow-based composition, with the advantage of using semantic description to aid both service providers and composers in building the composition scheme and adapting new Web services to it.

3.1. Introduction to Business Process Execution Language (BPEL)

The BPEL specification enhances and replaces existing standard XML-based extension of Web Services Description Language (XLANG) [34] from Microsoft and Web services Flow Language (WSFL) [35] from IBM. BPEL uses workflow management as a process model to achieve the control and data flow formalization for WSDL-defined data and operations. All the participant services in BPEL are modelled as partners (see Figure 8). The WSDL files of such partners are required to create a BPEL process. The partners contribute to the total processing capability of the BPEL process. A BPEL process also has its own processing capability for dataflow, control flow, data manipulation, fault and event handling and state management. The significance of the BPEL architecture is that the process itself is published as a Web service. This composed BPEL service can be treated as a single Web service and can be used for further composition hence facilitating recursive composition.

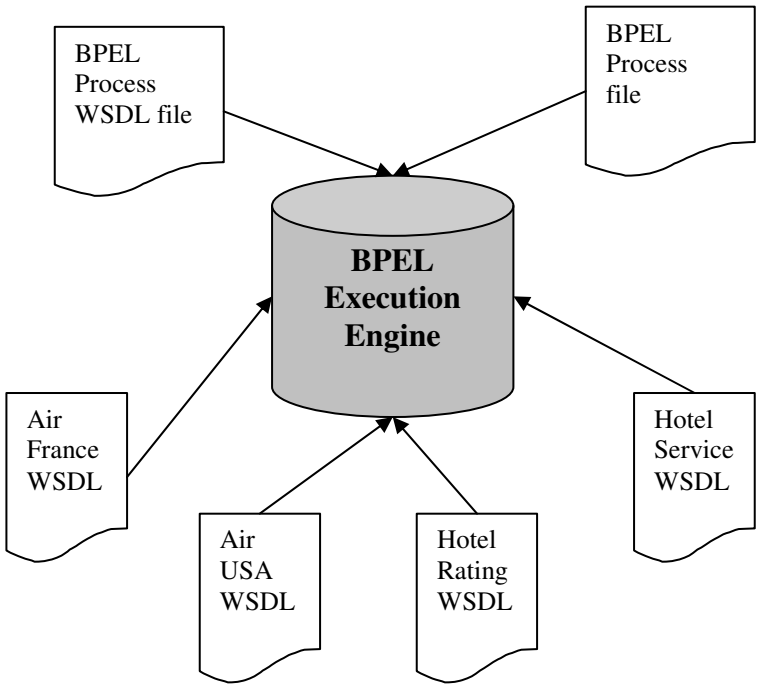


Figure 8 BPEL based Web services composition

The following is a Web service composition scenario implemented using Oracle BPEL process Manager [38]. This particular implementation of BPEL provides a graphical user interface to design business processes. The scenario is based on a travel agent process, which manages the reservation of airlines and hotels for the customer trip. The travel agent is implemented as a BPEL process, which is the composition of four Web

services depicting fictional businesses: `AirFrance` service, `AirUSA` service, `HotelRating` service and `HotelService` service.

The process logic for the travel agent is:

a) Check the availability of flight service from two competing airlines *`AirFrance`* and *`AirUSA`*;

b) Depending on the user request make flight reservation ;

c) Retrieve hotel ratings from the *`HotelRating`* service for the hotels at the destination city;

d) Make the reservation using *`HotelService`* Web service for the selected hotel.

Travel Agent Example

BPEL is built on top of WSDL; hence WSDL files of partner business services are required for the composition process. This fact is described in BPEL using *`partnerLinkType`*. The *`portType`* of such Web service defines the role of partner in the composition. Figure 9 shows *`AirFrance`* and *`AirUSA`* Web services as partners and the role they play in the composition using portTypes (i.e. fr: is the unique identifier for the *`AirFrance`* WSDL file).

```
<plnk:partnerLinkType name="airFrancePLT">
  <plnk:role name="AFcheckServices">
     <plnk:portType name="fr:AirFrance"/>
 </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType   name="airUSAPLT">
  <plnk:role name="AUcheckServices">
    <plnk:portType name="usa:AirUSA"/>
   </plnk:role>
</plnk:partnerLinkType>
```

Figure 9 Describing Partners in BPEL

Figure 10 illustrates the sequence diagram for the travel agent process where 1.1.a and 1.1.b are two activities for checking the availability of flight between source and destination city, performed in parallel. The BPEL syntax for this using <flow> to

achieve parallel execution is shown in the Figure 11 where both invocations are executed in parallel.

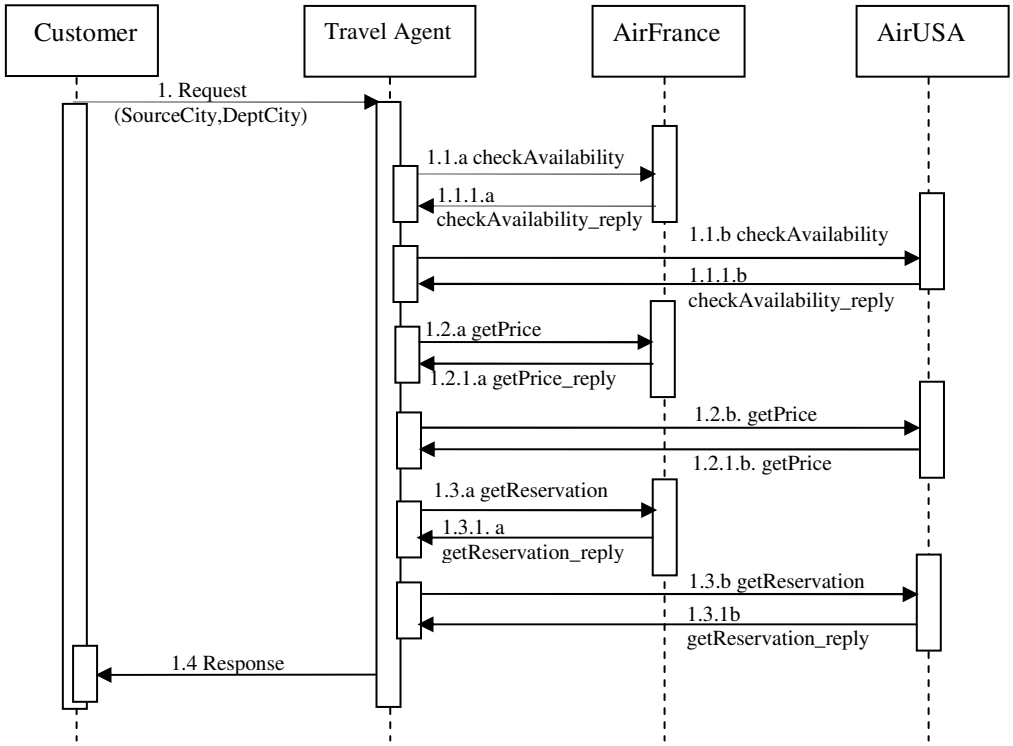


Figure 10 Sequence Diagram for the travel agent composition

```
<flow>
        <invoke name= “invokeAirFrancecheckServices”
                partnerLink = “AFcheckServicesPL”
                portType="fr:AirFrance"…….>
        <invoke name= “invokeAirUSAcheckServices”
                partnerLink = “AUcheckServicesPL”
                portType="fr:AirUSA"…….>
</flow>
```

Figure 11 Concurrency using <flow>

Similarly other operations for checking the possibility of reservation are performed on *AirFrance* and *AirUSA,* and reservation is made after comparing the price (activities 1.2a, 1.2b, 1.3a, 1.3b in Figure 10. The payment details are omitted to keep the example simple. Figure 12 shows the code where the user has specified the cheapest flight reservation in their preference.

```
<switch name="comparePrices">
        <case condition="bpws:getVariableData
```

```
                ('compInfo','PriceAirUSA') &lt; bpws:getVariableData('compInfo','PriceAirFrance')
                ">
                <invoke name= "AUinvokegetReservation" </case>
        <otherwise>
                <invoke name= "AFinvokegetReservation"
                <partnerLink= "AFgetReservationPL" ……..>
        </otherwise>
</switch>
```

Figure 12 Selecting the cheapest AirLine using <switch>

The implementation of travel agent example illustrates the expressiveness of BPEL as Web services composition language. This chapter uses the above described travel agent case study for the discussion.

3.2. Hybrid Framework for Web services composition

3.2.1. The Implementation Scenario

This research work uses classic travel agent problem as the implementation scenario for the composition tool. The implementation of the tool is based on the composition of real-world services from the airline businesses while dummy services were used for hotel and car rental business domains. None of the air line domain applications interface to users through a Web service, hence Web service wrappers were developed on the top of their HTTP portals, and they were then subscribed to a local UDDI and made available for the composition. For instance, wrappers were developed for three airline services: EasyJet (http://www.easyjet.com/), WizzAir (http://wizzair.com/) and FlyBmi (http://www.flybmi.com/) portals. The parameters and fieldnames in particular Web services are maintained the same as on the web portal.

In hybrid approach, the service composer builds a BPEL-based scheme for the composition of services belonging to specific application domains; it is then the responsibility of the service providers to adapt their Web services, if necessary, to the domain interface of the composition scheme. The advantage to the service composer is the ability to recompile and fire the composition with different domain-specific Web services with minimal effort. For instance, travel agent application composes services belonging to three domains: airline, hotel, and car rental. The travel agent pre-specifies the functionality (domain interface) that it expects from each participant, for example price quotation for the user specified flight details. A large section of information

engines and e-commerce services which integrate different Internet-based services through a unifying access interface fall under the same category; for instance loan providers (loan assessor, banks, insurance companies) and shopping robots.

The following sections explain how the domain-interface is specified and how it is exploited to facilitate the seamless dynamic composition of Web services based on the BPEL approach.

3.2.2. Specification of the domain of services

Central to the idea of the grouping services in a domain is the presentation of a domain-interface of the functionality expected from the service by the service composer in a standard, unambiguous format that is comprehensible by the software programs rather than the human developers. See Figure 13.

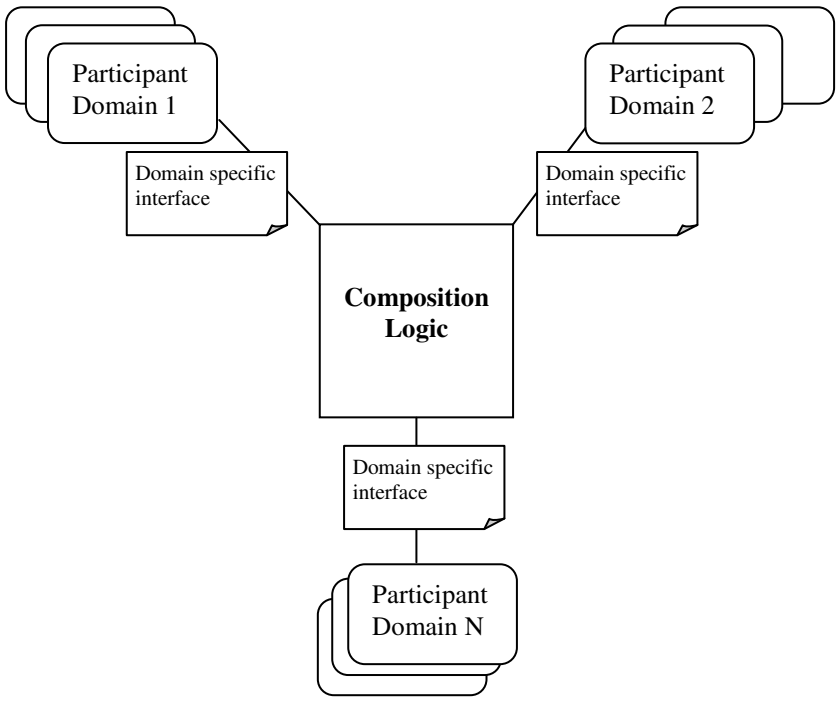


Figure 13 Specification of Domain

The BPEL execution relies on the WSDL syntactical standard that can be used for defining the expected functionalities from a participant Web service for particular domain. The problem with WSDL is that it is a syntactical standard that is developed for human developers rather than program based automation. Hence the tool uses ontologies defined with OWL, to describe the domain-interface depicting expected

functionality for a particular domain. In the tool, WSDL files are accompanied with a semantic description of the service parameters expressed in OWL ontology. This allows the description of expected functionality to be inferred in unambiguous form. Figure 14 illustrate the application of the above solution to the travel agent example

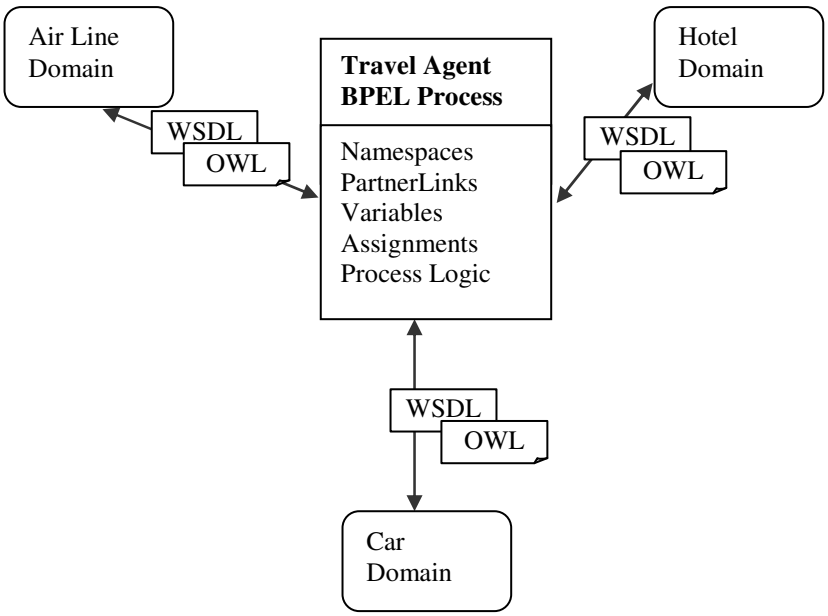


Figure 14 Domain specific composition

A segment of such owl-wsdl domain interface for the airline domain is shown in Figure 15 and Figure 16. The WSDL file complex-type `FlightQuery` of Figure 15 has been mapped into OWL class `FlightQuery` of Figure 16; hence an OWL reasoner can apply the class relationship based inference to verify that the mapped message type contains all the required elements.

```
<wsdl:definitions  targetNamespace="http://travelagent.ntu.ac.uk/AirLineDomainService">
<wsdl:types>
        <complexType name="FlightQuery">
                <sequence>
                        <element name="noOfAdults" type="xsd:int"/>
                        <element name="departure-date" nillable="true" type="xsd:dateTime"/>
                        …
                </sequence>
</complexType>
```

Figure 15 Domain specific interface- WSDL file

```
<owl:Ontology rdf:about="http://localhost/ntu/ac/uk/2005/
TravelAgent/AirLineDomain.owl">
</owl:Ontology>
<owl:Class rdf:about="http://localhost/ntu/ac/uk/2005/
onto/travelquery.owl#FlightQuery">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="http://localhost/ntu/ac/uk/2005/onto/travelquery.owl#noOfAdults"/>
   <owl:someValuesFrom>
    <rdfs:Datatype  rdf:about="http://www.w3.org/2001/XMLSchema#int"/>
   </owl:someValuesFrom>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="http://localhost/ntu/ac/uk/2005/onto/travelquery.owl#departure-date" />
   <owl:someValuesFrom>
    <rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#dateTime"/>
   </owl:someValuesFrom>
  </owl:Restriction>
 </rdfs:subClassOf>
```

Figure 16 Domain specific interface - OWL file

If a new domain-related Web services is to be created, the domain-interface files can be used to create a new Web service that adheres to the functionality expected by the service composer. Otherwise, the service provider needs to edit the ontology file to overcome any mismatches in the service descriptions (parameters and method names). In this case, the ontology can bridge the semantic mismatch provided that conceptual meaning remains the same. Figure 17 describes an ontology file provided by one of the candidate airline service to overcome semantic mismatches with the travel agent domain interface. The ontology file documents the fact that `departureFlightDate` element of this airline description is conceptually similar to the element `departure-date` in the Figure 16.

Figure 17 Ontology file for EasyJet Airline service

```
<owl:Ontology rdf:about="http://localhost/
ntu/ac/uk/2005/EasyJet/easyjet.owl">
```

```
</owl:Ontology>
<owl:DatatypeProperty
rdf:about="http://localhost/ntu/ac/uk/2005/EasyJet/easyjet.owl#departureFlightDate">
  <owl:equivalentProperty>
  <owl:DatatypeProperty
rdf:about="http://localhost/ntu/ac/uk/2005/onto/travelquery.owl#departure-date">
  </owl:DatatypeProperty>
  </owl:equivalentProperty>
</owl:DatatypeProperty>
```

3.2.3. Dynamic Pool for Domain-Specific Web services (DPDWS)

In the second phase, the tool attempts to integrate the domain-specific Web services into a dynamic pool, where the services can dynamically plugged-in and out of the composition scheme, without the need to re-code the composition logic. As explained in the previous section, the prerequisite for domain membership is the availability of a WSDL file describing the service functionality and an accompanying ontology file, ensuring the compatibility of the service parameters to the domain interface.

Domain membership verification

Domain membership verification module verifies the membership of Web services to a particular domain and ultimately the composition scheme. The module verifies the above-mentioned prerequisite according to the following steps (the airline domain is exemplified):

1. Parse the WSDL and corresponding OWL files of the candidate Web services against the domain interface to check all the possible mappings between what is expected and what is provided by the candidate service. If the candidate service description file - WSDL has different format to the domain description file, the supplied ontology is searched for a mapping for this mismatch. If the ontology file has the required mappings, the mappings are stored for future use when the actual composition with this service takes place. For instance, the membership module stores valid mapping *departure-date-> departureFlightDate* for EasyJet service.

2. If the service parameters match semantically, make the service available within the AirLine DPDWS (Figure 18), i.e. declare the service as composition-ready; this

involves storing a reference for the service with the composition-necessary details: target namespace, mappings between required-provided elements, operation name with corresponding portTypes, message names and message types. The verification module also create partnerLink name, partnerLink type and partnerLink role based on the service name for this service. Table 3 describes such possible information. These details are used when the actual composition is carried out.

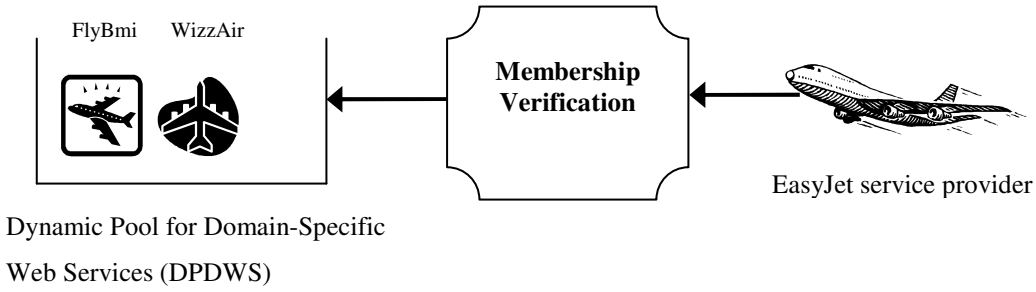


Figure 18 Membership verification module for the Dynamic Pool for Domain-Specific Web services (DPDWS)

Table 3 Information stored by Membership Verification module

Store	**mismatch1** (FlightQuery => departureFlightDate, FlightQuery=>departure-date)
	messageName getEasyJetFlightsRequest
	operation Name/portType CheckReservation/ EasyJetPortType
	Namespace http://travelagent.ntu.ac.uk/EasyJetFlightService
Create and Store	**Namespace prefix** ejet
	Variables inputEasyJetAir => getEasyJetFlightsRequest
	partnerLink name EasyJetPL
	partnerLinkType EasyJetWSLink
	partnerRole EasyJetWSProvider

Figure 19 is the snapshot of airline domain membership verification module, which implements domain membership algorithm and is designed using Jena [70], Pellet [71] ontology reasoner, DOM XML parser and the Java technology. The only input required from the service provider is description and ontology files and the tool takes care of

making the service composition-ready by following the membership verification algorithm.

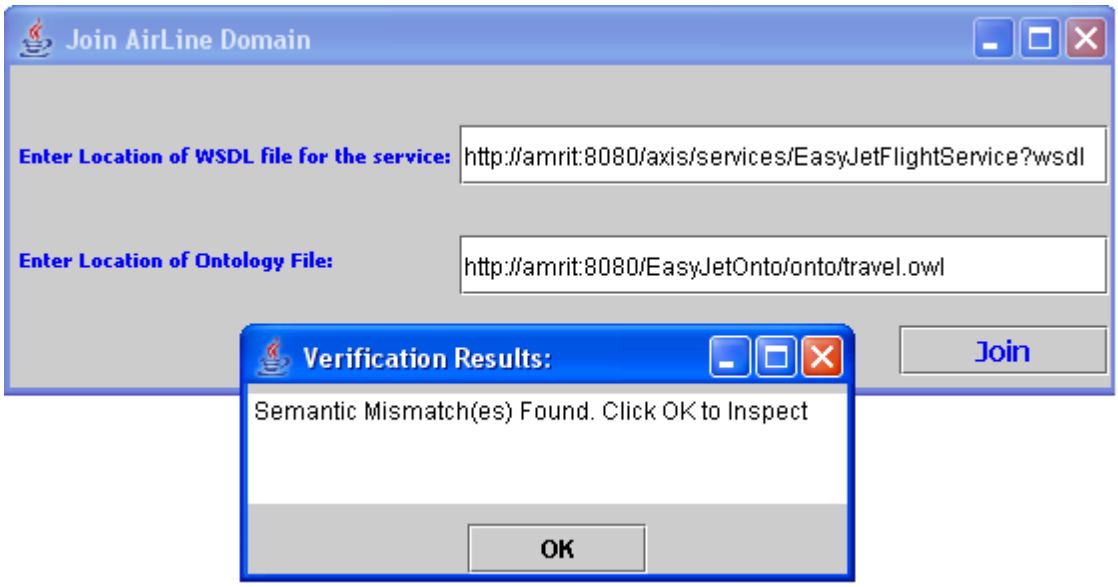


Figure 19 Membership Verification

The next section details the mechanism for automating dynamic selection of Web services from the dynamic pool and their integration into the composition scheme.

3.2.4. Dynamic BPEL-based service composition facilitated by DPDWS

In the tool implementation, dynamically adding a Web service from the domain pool constitutes placing an instance of the service in the composition scheme file. For example, to add the functionality of retrieving a price quote for a specific journey by the easyjet airline service, the travel agent service composer will have to add the following instance to the relevant execution segment of the BPEL composition file:

```
[<invoke name partnerLink="EasyJetPL"
portType="ejet:EasyJetPortType"    operation="checkReservation"
inputVariable="inputEasyJet" outputVariable="outputEasyJet"/>]
```

Such integration is automatically performed by dynamic composition tool. Hence, the BPEL process file does not have to be manually edited and recompiled to integrate alternative Web services into the composition scheme.

Table 4 shows how a BPEL process can be created with the programming-based tool. This implies that the BPEL process file can be created dynamically with the inclusion of the new services from particular domain. This tool can create the service references by reading the WSDL file and can add them throughout the composition scheme, making

the creation of process file automatic and the resultant composed service execution ready. This makes the scenario shown in Figure 20 possible where services from the domain can be plugged in and plugged out automatically.

Table 4 Process file creation with Java

Required composition function	Corresponding tool method
Add partnerLinks for the airline service with particular values for the new service	public String setParetnerLinks(Document bpeldoc, String prefix, String partnerlink_name, String partnerlink_type, String partnerlink_role)
Set the process logic for the AirLine service by placing partnerLink, which has price Check operation.	public void setPriceCheckInstance (Document bpeldoc,,String invar, String outvar, String portType, String operation, String partnerlink_name)

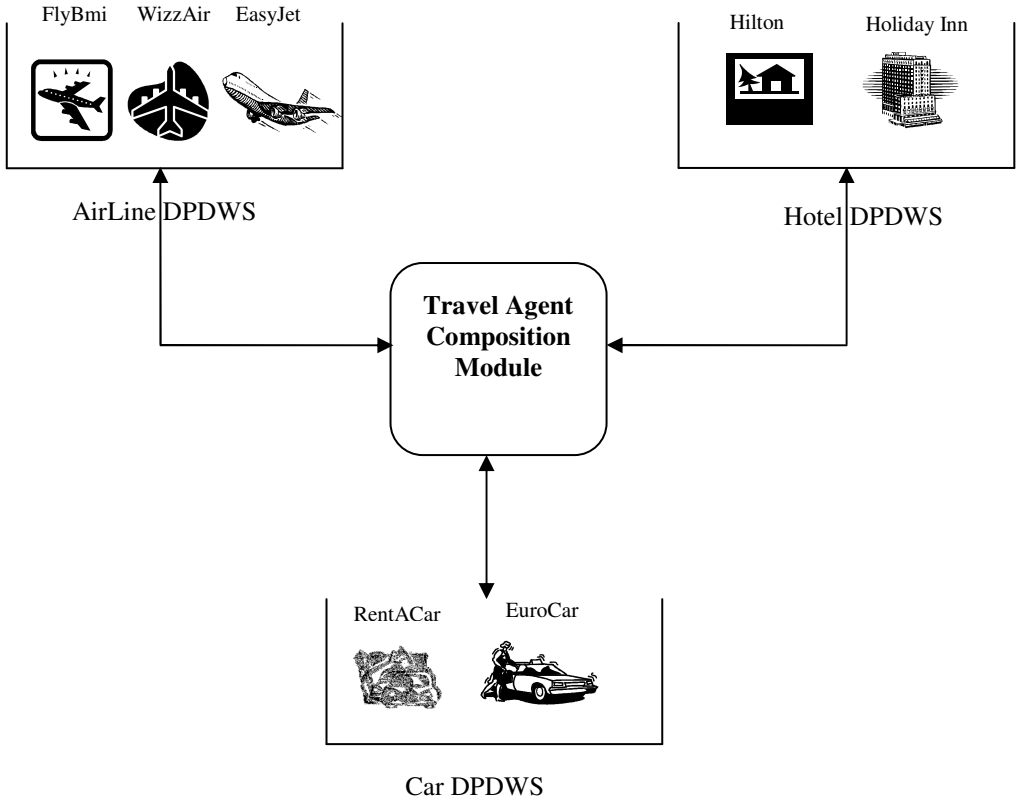


Figure 20 Travel agent composition facilitated by DPDWS

The target BPEL execution engine for the tool is Oracle's BPEL Process Manager [38]. It is worth to mention that this particular implementation of BPEL also requires two additional files to be input with the BPEL process file: a service wrapper WSDL file that contains information to make the service a partner in the business process and a BPEL configuration file that identifies the location of the wrapper file and binds it with a particular Web service `partnerLink`. For each new service participating in the

composition, the *bpel.xml* file is modified to include new service. The tool creates the process file, wrapper file and adds the entry in the *bpel.xml* file making the process files composition ready and with the inclusion of newly added service.

Following is the algorithm for DPDWS-facilitated composition, which creates BPEL process file automatically allowing the services dynamically selected from the domain.

Algorithm for DPDWS facilitated domain specific Composition.

In the implementation, the composition module is always initiated with a default skeleton file that contains the composition scheme of default Web services. For example, composition can be initiated with the easyjet Web service from airline domain, hilton Web service from the hotel domain and Rent-A-Car Web service from the car domain. The tool performs the following steps for facilitating the dynamic composition of alternative domain-specific Web services.

1. On the selection of an alternative domain service, a new BPEL composition scheme and other configuration files required by the BPEL execution engine are generated. This is achieved as follows:

   i) The reference for the alternative service is selected from the membership verification module. This will include all the details pertaining to the service and required by composition module such as *partnerLink* details, namespace, and prefix. Next the semantic mappings are retrieved from membership verification module and used them wherever applicable during the process logic.

   ii) The new service namespace is added to the root element of the newly created BPEL composition scheme.

   iii) *PartnerLink*s are added for the new service.

   iv) The messages of the Web services are mapped to the BPEL process variables; the variable names are generated automatically. Steps ii-iv use the reference details created during membership verification module.

   v) The process logic for the new service is composed from the created service instances. This includes the addition of the service instance at all the places where the composition logic for a particular domain is defined in the default skeleton BPEL process file. Examples of such instances can be invoking the service,

assigning responses to intermediate variables and passing them for particular operations etc.

2. The newly generated BPEL composition scheme is validated.

3. A service wrapper file is created with the partner link information defined for the service reference and including a pointer to the location of WSDL file within the wrapper file.

4. Finally the `partnerLink` details are bound with the service wrapper file location and the existing `bpel.xml` file is modified to reflect the integration of the new service.

The composition module algorithm is implemented using Java technology and DOM XML parser. Figure 21 illustrates the admin interface of the composition tool. The locations of process (BPEL skeleton file) and configuration files are necessary for the initialisation of the tool. The list of available services to each DPDWS is dynamically populated with the membership verification module detailed earlier. The service composer can select any possible combination of service from domains for composition and new process file with configuration files are automatically created and the composed service is fired if required.

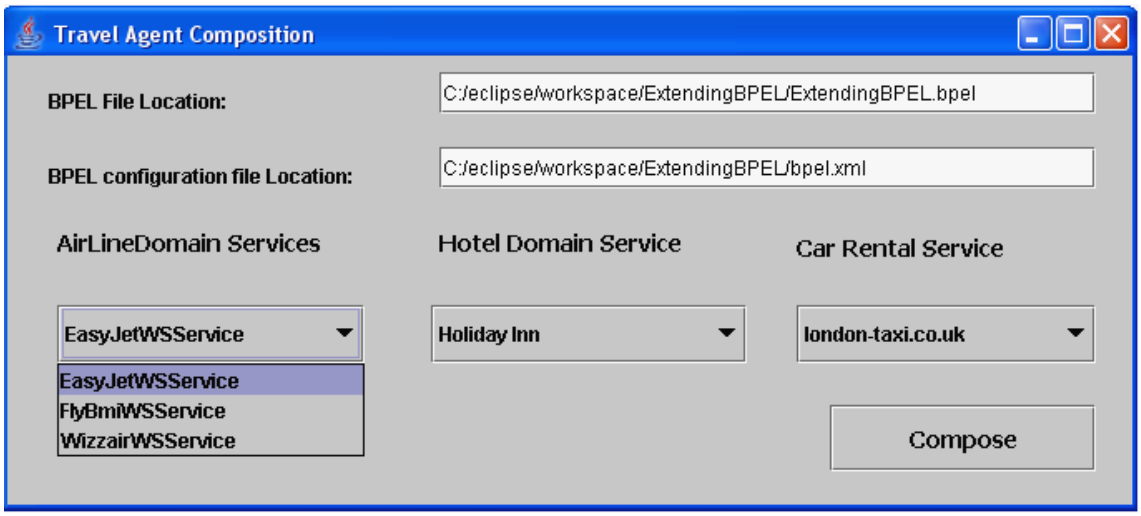


Figure 21 Travel Agent Composition

In the tool implementation, the composition module is initialized with the default skeleton file (See Appendix A, Table 24). When a different domain service has been selected for the composition, the composition module retrieves the information for the new service from the membership verification module. For instance, to replace the

default *WizzAir* service with *EasyJet* service in the composition scheme, here are few examples on how the verification and composition modules collaborate to bind the new service:

- The name space now represents *EasyJet:* Namespace = *http://travelagent.ntu.ac.uk/EasyJetFlightService*
- In place of expected *FlightQuery* ➔ *depature-date* element, the service will expect *FlightQuery* ➔ *departureFlightDate*.
- The unique prefix for this service becomes *ejet* and *partnerLink* name = *EasyJetPL*
- Variable for the *messageType getEasyJetFlightsRequest* becomes *inputEasyJet*

This information is feasible to generate and retrieve considering the domain specific implementation of the composition module and restriction imposed on the service participants. The composition module then takes the default BPEL file and replaces the instances of new service by following the unique prefix identifier of the existing service in the composition scheme. Refer in Appendix A.

The approach explained in this section demonstrates the use of domain-specific services combined with lightweight semantics to alleviate the cumbersome and time-consuming task of manually compiling a BPEL-composition scheme each time a new service is added to the composition scheme. This is very important particularly when the underlying composition logic rarely changes.

3.3. Related work

In recent years, the research community have realized that the union of semantics with business standards can be helpful in automating composition tasks.

Akkiraju *et al.* in [24] presents such semantic-based approach which uses semantic annotations within WSDL file, to facilitate service discovery and selection. The hybrid approach discussed in this chapter differs in that it uses ontologies in combination with WSDL to describe the service fields and to incrementally describe any mismatches in the service provider's service. The logic implementing the association between domain specific WSDL fields and domain-specific ontology elements is handled using the

ontology reasoner and is pre-defined and hard coded in the membership verification module. The membership module scans the service participant's ontology files for equivalent properties.

Mandell and McIlraith adopt [67] similar approach but propose a bottom-up approach for Web services interoperation in BPEL4WS; they use OWL-S based descriptions for runtime binding of service partners. The Implementation collects the OWL-S profiles into a repository and exploits the profile semantics to query partners for desired properties. This approach allows selecting partners at run-time otherwise selected at design-time according to BPEL process model. Their implementation includes SDS (Semantic Discovery Service) module, which works as a broker for the semantic discovery. SDS sits between the BPEL process engine [40] and Web services partners. In the framework BPWS4J, in place of passing requests to hard-coded, pre-selected partners directs them to SDS, which in turn locates service partners providing required properties. This approach uses semantic web Technology for automatic, meaningful service selection. However, the problem of actually automating the composition process is not addressed, as the composition logic is built manually for inclusion of partner services.

The hybrid tool considers the composition from the service composer's perspective. The service composer categorizes the possible service partners into domains and makes the domain specific interface (WSDL+OWL) available to the service providers. This interface serves as the prerequisite for joining particular domain. Hence, the tool is based on top-down approach that declares the expected requirements first and then populates domains with compatible services; unlike [67], which uses OWL-S profiles for selecting service partners based on service descriptions. The tool also allows creating a general re-usable programming framework for selecting services from particular domain and composing them automatically.

Traverso and Pistore in [68] present an AI planning based technique to convert semantic (OWL-S) web service process models into executable BPEL4WS processes. The implementation translates the OWL-S profile models into partially observable state transition systems, which are utilized for generating plans to reach the goals for composition. Their approach uses semantics at the composition level and takes advantage of the expressiveness and executable nature of low-level BPEL processes. The approach targets the composition of services to be automatic, while service

discovery and selection is manual. The hybrid tool also uses semantics at the composition level; however it exploits the BPEL process creation mechanism combined with domain concept to implement an automatic composition programming tool rather than using planning techniques. The implementation allows selection and removal of service partners in the composition to be automatic.

3.4. Summary

The aim of the research effort in this chapter was to create a tool that alleviates the burden of dynamic Web services composition. The argument is that despite the evident popularity of Web services as a secure distributed computing paradigm and the value-added dimension that composition adds to it, the practical adoption of the technology is still hindered by the knowledge and effort required for the compilation of the composition process and the manual adaptation of new and existing web services to it.

After critical analysis of current approaches to Web services composition, the conclusion was that there is scope for developing a practical and current solution that merges the benefit of practicality of use and adoption popularity of workflow-based (BPEL-based) composition, with the advantage of using semantic description to aid the composition participants in automatic discovery and interoperability of the composed services.

The main premise of the approach is to aid the service composer in building a generic BPEL-based scheme for the composition of services belonging to specific application domains, and assist the service providers in adapting their application services to the composition scheme. Web services join the BPEL composition scheme by subscribing to a specific domain interface.

In the tool, the domain functionality described in WSDL-XML grammar is accompanied by a semantic description of service parameters expressed in OWL ontology, allowing the description of the expected domain functionality in an unambiguous form and catering for any mismatches in the Web services description. A domain membership verification module was developed that allows the service providers to adapt their application services to the domain interface and making them with minimal effort.

Once a domain Web service is declared composition-ready, the dynamic composition tool transparently integrates the Web service into the BPEL process file, i.e. it is automatically added to a pool of dynamic Web services for this domain. The chapter describes the algorithm for dynamic population of the domain pool with Web services, thus allowing the service composer to effortlessly select any possible combination of services from the composition domains and fire the composed service.

3.5. Limitations of the workflow-semantics hybrid approach

The BPEL specification solves the immediate problems industry is facing regarding the use of Web services for enterprise application integration. However, in its present form the specification overlooks the possibility of binding the service participants and performing flow management on the fly, hence only specifies how the service composer can perform both activities manually. As demonstrated in this chapter with the DPDWS tool, enriching BPEL specification with semantics achieved automatic selection of the Web services with prior-agreed interfaces. The hybrid approach presents a practical solution to a current problem. However, the approach could only achieve limited automation to the composition as elaborated below:

- The main contribution of this approach is to utilize semantics in the BPEL specification to provide dynamic selection of the Web services participants at runtime with the use of semantics processing as a middleware. However this does not take full advantage of the semantic description capability, as the use of semantics is limited to the Web services functional parameters. The non-functional parameters play a significant role in deciding service suitability for particular task; for example, a Web service can be selected based on the Quality of Service it provides. The main problem is that the BPEL specification has no scope to accommodate non-functional parameters beyond IOPE (Input, Output, Precondition, and Effect) due to the absence of syntactical notation in BPEL.

- In order to automate Web services composition, two problems have to be resolved: automatic discovery and selection of Web services and automatic compilation of flow management for the selected services. The hybrid approach addresses the Web service discovery problem, but relies on the flow management provided by the BPEL process model hence on the understanding of service composer to design the flow management.

To summarize, the use of semantics with workflow-based composition is going to involve the human developer at some stage whether it is at the level of domain subscription or compilation of the composition scheme. Hence, the provided facilitation is restricted.

The root of the problem is related to building the process model on top of WSDL, which is an XML grammar. Using XML one cannot define concepts or relations between concepts, which is the most important factor for the intelligent reasoning required for the automation. The issue related to the current discussion is the use of non-semantic grammar for the composition specification. For the composition engine to provide automatic discovery and flow management, the process model needs to have the consideration of the semantics in the specification. The addition of semantics within an XML centric standard like BPEL will not achieve the sought-after automation as that would require an intelligent reasoner which can interpret the semantic description. The following chapter will introduce research efforts to develop an intelligent semantic-web based reasoner based on the AI theory of Case Based Reasoning (CBR).

4 Chapter Four: Semantic-Driven Matchmaking and Discovery of Web Services using Case Based Reasoning

The automated discovery of adequate Web services is the pre-requisite and core feature for achieving dynamic Web services composition. This chapter presents an approach that utilizes Case Based Reasoning (CBR) methodology for modelling Web services discovery and matchmaking problem. The framework uses OWL semantic descriptions extensively for implementing both the components of the CBR engine and the matchmaking profile of the Web services.

4.1. CBR for automated Web services discovery and composition

The accuracy of service selection is critical to the success of the composition process and largely relies on assessing the capability of a service in accordance to the service composition request. In this research, the concept of considering "runtime behaviour of services" to improve the accuracy of Web services discovery is proposed. The argument is that the existing semantic and non-semantic Web services composition approaches do not consider run-time behaviour of Web services in order to assess service suitability for the service request. For example, semantic approaches that rely on OWL-S profile for discovery compare service descriptions for the service request and existing services in registry in terms of whether the offered service has similar inputs and outputs with similar data or object types to the service request, and if it has then the service is considered a potential solution. These approaches can satisfy coarse-grained service requests that consists of a simple singleton query such as book purchase services, airline booking services or sensor reading services; however these approaches cannot satisfy fine-grained service requests such as finding a book purchase service that charges in USD or finding an airline that travels from Milan and charges in EUR or finding a sensor service that has reliability of 0.9.

Figure 22 exemplifies the argument about consideration of run-time behaviour of Web services in service selection. As shown in Figure 22, in existing approaches, for a new service request the descriptions are matched with the available service descriptions. For instance, to find a service that provides flight to the German city *Bonn* and charges in *USD*, the existing approaches match service descriptions of (*OutputCurrency*, *To_City*) with the existing services in the service registry. Although for the candidate Web services it is highly likely that service descriptions are semantically similar, the run-time execution values can vary significantly. This variation is expressed in the values for such functional and non-functional parameters constituting domain-specific knowledge. This domain-specific knowledge can provide valuable guidance for decision-making process regarding service adequacy for the task. This is because service run-time behaviour is difficult to presume prior to service execution and can only be formed based on the experience with the service execution.

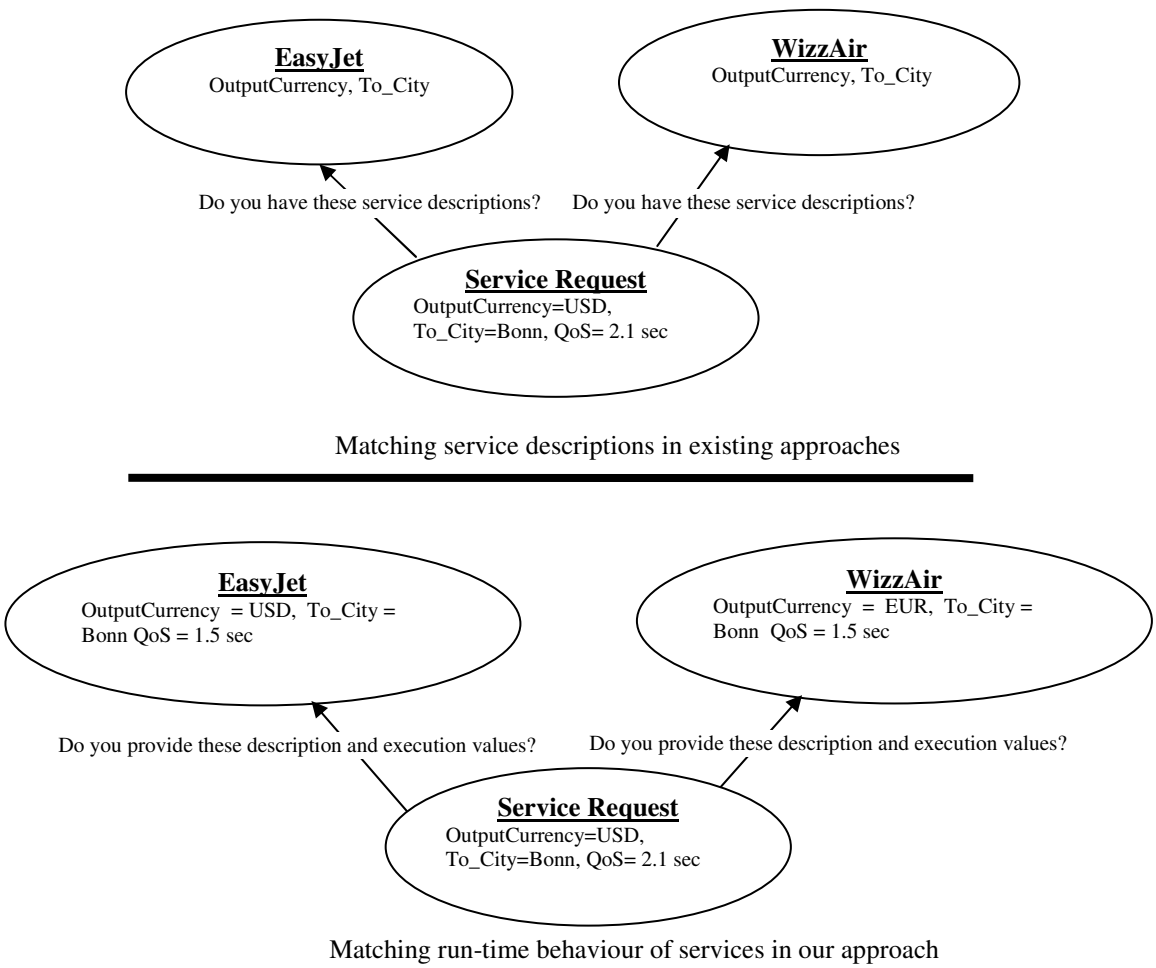


Figure 22 Matching service descriptions v/s service run-time behaviour

As shown in the Figure 22, in the proposed approach considering the execution values of Web services in service selection is advocated. For instance, service request in

addition to functional parameters (*OutputCurrency*, *To_City*) can include non-functional parameters such *QoS*, and compare services in registry with their execution values such as there exists a past run-time experience with *EasyJet* where the service charged in *USD*, the destination city for travel was *Bonn* and QoS of the service was 1.5. The accuracy of automatic matchmaking of Web services can be further improved by taking into account the adequacy of such past matchmaking experiences for the requested task.

Therefore, there is a need for a methodology that uses domain-specific knowledge representation system for capturing the Web services execution experiences and reason based on those experiences. Case Based Reasoning [72] provides such methodology as its fundamental principle is that experience formed in solving a problem situation can be applied for other similar problem situation. An added benefit of reasoning about past execution experiences can be the analysis of aggregate service behaviour over time. For instance, more precise conclusions can be drawn about the service reliability by analyzing its QoS execution experiences over a period of time.

This chapter presents a Semantic Case Based Reasoning (SCBR) framework, in which reasoning for service discovery and matchmaking is based on a set of previous experiences or cases described using semantics.

4.2. Overview of Case Based Reasoning

The CBR technology was developed in 1977 based on the research effort of Schank and Abelson. In [73], they proposed that our general knowledge about situations is recorded in the brain as scripts that allow us to set up expectations and perform inference. CBR's fundamental premise is that situations recur with regularity [74] i.e. experience involved in solving a problem situation can be applied or can be used as guide to solve other contextually similar problem situation. The reasoner based on CBR hence matches the previous experiences to inspire a solution for the new problems. The processes involved in CBR can be represented by a schematic cycle as described in Figure 23 and comprising of four phases [74].

- RETRIEVE the most similar case(s); this phase requires case retrieval methodology to find cases with similar experience.
- REUSE the case(s) to attempt to solve the problem; this phase requires a case matchmaking methodology to identify similar cases in order to reuse those cases.

- REVISE the proposed solution if necessary, this phase requires case revision methodology to adapt existing cases to fit new problem request.
- RETAIN the new solution as a part of a new case; this phase requires case representation to be defined and cases to be indexed and stored.

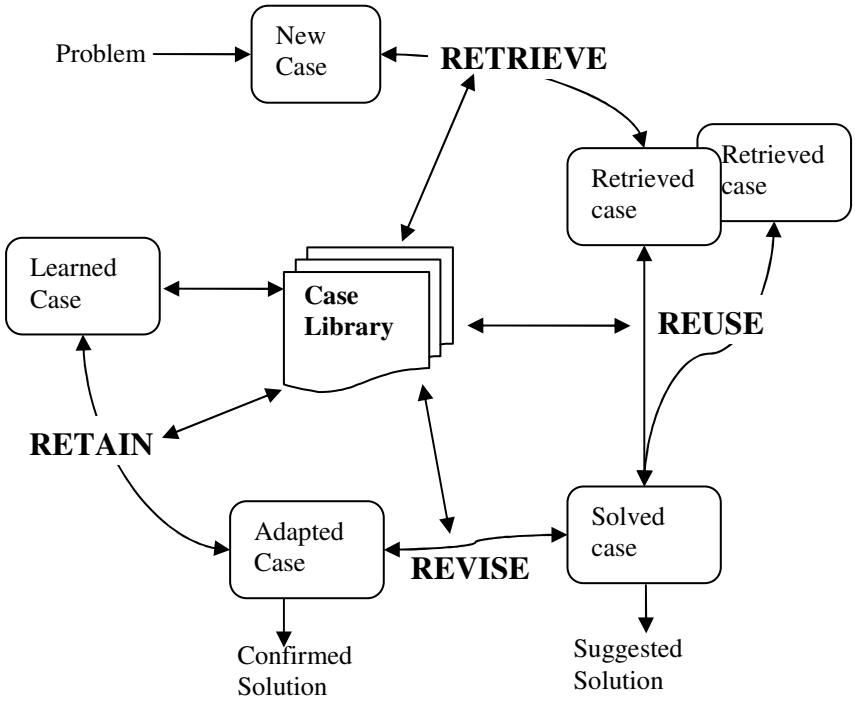


Figure 23 The CBR Cycle

Following figure describes the main stages in CBR reasoning to achieve the aforementioned four stages in the CBR cycle.

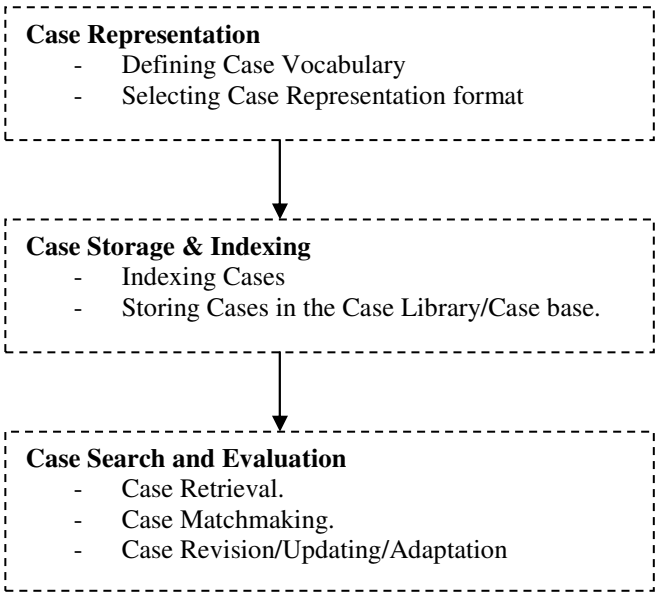


Figure 24 CBR methodology

4.2.1. Case Representation

Case is a core component of CBR system and can be defined as a contextualized piece of knowledge representing an experience [74]. It contains the problem, a description of the state of the world when the case occurred, and the solution to this problem. When a reasoner is created, the elements of the case are defined according to the context. For example, the city of departure or the output currency could be some elements to represent a travel experience as a case. Case vocabularies are thus developed for each reasoner, to define what knowledge needs to be captured. Hence, case vocabularies are the labels or the representation schemas defining knowledge. These vocabularies need to be organized in modular or structured fashion to make them recognizable by the CBR reasoner; hence various representation styles for case representation exist.

4.2.2. Case Storage and Indexing

A case worthy of storage contributes to the reasoning process by representing a potential base solution for new problem situations. Such cases need to be indexed and stored in the case library or case base, so that reasoner can retrieve them for reasoning. The process of searching entire case library is computationally expensive and indexing cases and searching cases based on indices allows frameworks to efficiently find a solution as indexing process effectively reduces number of cases to be investigated. Apart from efficiency the purpose of indexing cases is relevance, i.e. to retrieve contextually relevant cases to the new problem.

4.2.3. Case Search and Evaluation

Whenever a new problem needs to be solved, case library is searched for the cases that can provide potential solution. The first phase of the search is case retrieval, and uses indexing to retrieve cases that are contextually similar to the new problem. The next phase is matchmaking where the retrieved contextually similar cases are further matched or investigated to verify if a solution to prior problem situations can be applied to the problem in-hand. If the system does not find an adequate match, then the combined contextual knowledge of relevant cases is applied to solve the problem, this phase is called adaptation. On success, adapted cases are entered in the case library. On failure, the situation leading to failure is entered in the case library, which serves as a guide to the CBR reasoner to avoid future failures in similar problem situations. The

inconsistencies encountered during the evaluation are recorded as cases and are termed case revision.

4.3. Modelling Web Services Discovery and Composition Problem into CBR Problem

CBR maps naturally into the Web services composition problem as it is possible to model the search and adaptation methodology in CBR as Web services discovery and composition mechanism. Figure 25 illustrates how CBR modelling can be applied to the problem of Web services discovery and composition problem. In the SCBR framework, Web services execution experiences are modelled as cases where the cases are the functional and non-functional domain specific Web services properties described using semantics. In this modelling, the case library will be the storage place for such execution experiences and is identical to Web service registry in that it stores Web services references, but unlike registries case libraries also describe runtime behaviour.

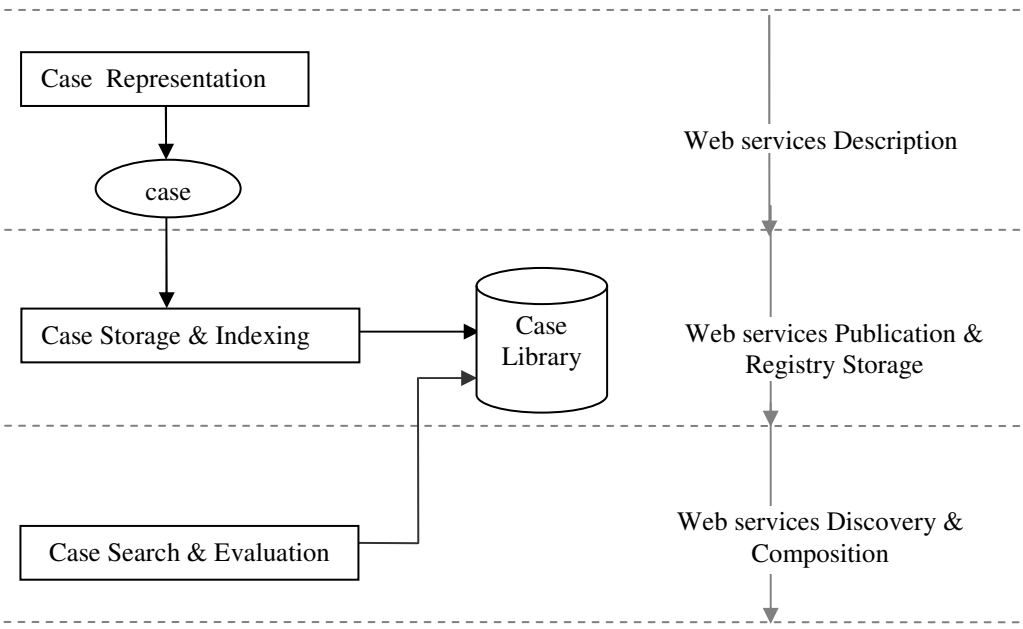


Figure 25 Mapping Web services composition problem to CBR

The process of case search is divided into the matchmaking and retrieval sub processes. The retrieval process is similar to Web services discovery problem in that both mechanisms seek to find potential Web services for the current problem. The case matchmaking process is similar to Web services matchmaking as both processes attempts to select acceptable Web services from the retrieved Web services by the retrieval phase. The process of case adaptation which is applicable when the available cases cannot fulfil the problem requirements and the process is carried out by adapting

available cases, is similar to Web service composition, as the composition is applied when available services are not sufficient in meeting the requirement for the problem.

The apparent compatibility confirms thesis of this research that the CBR methodology is well suited to build automatic Web services composition frameworks. This chapter explores utilization of CBR to model the Web services discovery and matchmaking problem. Chapter 5 deals with the problem of service composition.

4.4. Use of Case Based Reasoning for Web Services Matchmaking

4.4.1. The Framework Architecture

In the SCBR framework, there are two main roles: case administrator who is responsible for case library maintenance by entering or deleting cases from the library and case requestor who searches the case library to find solution for the problem and is similar in role with Web service requestor. Figure 26 illustrates the schematic diagram for the framework.

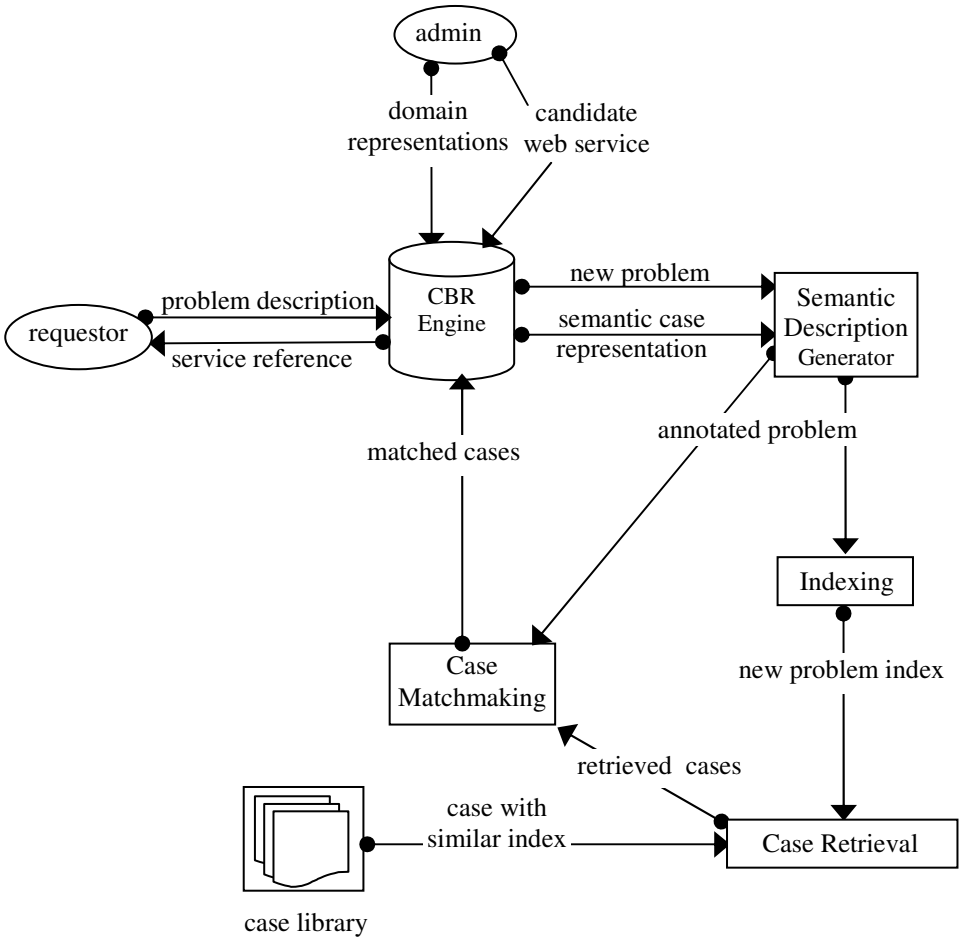


Figure 26 Architecture of the SCBR framework

The framework allows the Web service requestor to provide problem description and search for Web service that meets the requirements. The dynamics of the framework operation is as follows:

1. Initially, the administrator populates the repository with semantic case representation formats for specific application domain. This representation is used to semantically annotate both the user requests for suitable services and the execution experiences of Web services for the specific domain.

2. The SCBR Engine is the first entry point for the web service requestor, who can use the user interface to input the problem requirements and as a final result receives Web service references with other details. After receiving the problem description, SCBR starts search for finding suitable services that matches the request.

3. At this stage, the engine passes new problem description and the custom semantic case representation format to the semantic description generator module, which annotates the new problem according to the representation format

4. The annotated problem is then passed to the indexing module, which computes the suitable index for the new problem and passes the index to the case retrieval module.

5. The case retrieval module queries the case library for cases with the similar indexes. Output at this stage will be the cases, which have similar index to the current problem and these retrieved cases are passed to the next stage.

6. The case matchmaking module takes retrieved cases and the annotation of problem description from the semantic description generator module, and outputs matched cases.

7. The CBR engine receives these matched cases and extracts the Web services details from the solution part of the case.

8. The CBR engine returns Web services details to the service requestor.

4.4.2. Benefit of utilizing semantics for service discovery

Web Ontology Language (OWL) is utilized for constructing ontologies in this framework. From a computing science point of view, ontology represents an area of knowledge that is used by people, databases, and applications that need to share domain

information. Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them.

Applied to Web services retrieval, the semantic annotation of Web services creates a conceptual understanding of the domains that the services represents, enabling software agents, i.e. search engines, to make more intelligent decisions about the relevance of the services to a particular service request. For example, when searching the jUDDI free-text based Web services search engine for some travel web services relevant to London, it seems relevant to use keywords 'travel service to London'. However, the jUDDI search engine returns 1 service out of possible 10 relevant services, with returned results including `London Underground Web service`, primarily because the string "London" is part of the service name.

The use of the semantic web in Web services retrieval is likely to improve the computer's understanding of the domain objects and their interactions. The goal is to make the machine understand that London is a city, and that it is an English capital and there are number of transport mediums available departing from and arriving to the city of London.

The ontology relating London to City concept should be able to retrieve all the services execution experiences where departure city is London. To attain such expanded results, the data needs a better structure, so as to make sense for a machine that City are attached to Travel and can be either departure city or arrival city. Here, the semantic web is likely to bring a structure that integrates concepts and inter-entity relations from different domains, such as City, Travel, and Transport in relation to the query above.

4.4.3. Semantics for Case Representation and Storage

The most common use of ontologies is the reconciliation between syntactically different terms that are semantically equivalent. Applied to CBR case descriptions for Web services, ontologies can be used to provide a generic, reasoner-independent description of their functional and non-functional parameters. Moreover, ontologies can be used to further index and structure cases with key domain features that increase the efficiency if the matchmaking process. For instance, it is possible to add a feature to the travel domain ontology to indicate whether a trip is domestic or international.

In the framework, ontologies are also used to describe the rules of the CBR reasoning engine which streamlines the intercommunication between the Web service, user request, and the case library.

This section provides details on how we use CBR modelling to address the Web services discovery and matchmaking for specific application domains, exemplified by the classic travel domain problem where a user (Web service requestor) searches suitable Web service for a planned travel trip.

Case Vocabularies

In CBR theory, the first step is to define all the elements contained in a case and the associated vocabulary that represents the knowledge associated with the context of a specific domain. This vocabulary includes functional and non-functional parameters:

1. Functional parameters are the service inputs (e.g. the travel details) and the service outputs or results are (e.g. travel itinerary). The Input corresponds to the request of the user (e.g. date or city of departure) whereas output corresponds to the response given to the user (e.g. price, flight number).

2. Non-functional parameters are constraints imposed by the user (e.g. exclusion of particular travel medium) or preference over certain parameters (e.g. price range, Quality of Service expected). In addition, runtime experiences stored in the case library should also include the solution (e.g. Web service effectively used) and a notion to specify if the solution is acceptable for the end-user. Features that characterise the domain are extremely useful for top-level indexing and can also be included as non-functional parameters.

Case Representation using Frame structures.

After deciding on the knowledge and corresponding vocabulary to be represented as a case, we need to decide how this knowledge can be represented. The proposed approach adopts frame structures for the case representation [75]. In frame structures, frame is the highest representation element consisting of slots and fillers. Slots have dimensions that represent lower level elements of the frame, while fillers are the value range the slot dimensions can draw from. In the implementation, slot dimensions represent case vocabulary in modular fashion while fillers describe the possible value ranges for the slot dimensions.

The frame representations are highly structured and modular which allows handling complexity involved in representation. Moreover, frame structures have a natural mapping to the semantic OWL descriptions language as the semantic net representations largely borrowed from the frame structures [76], which makes natural transition to the semantic web descriptions possible. For example, `slot` in frame structure maps to `Class` in OWL descriptions. Table 5 shows a frame structure for the travel domain case vocabulary.

Table 5 Travel Domain Frame Structure

Slot	Dimension	Filler
Travel Request	City Departure	valid city
	City Arrival	valid city
Travel Response	Price Range	positive double
	Currency	any Valid Currency
Constraints on Goal	On Instance	valid Travel Domain Instance
	On Domain	valid Travel Domain
Preferences	On Price range	positive Double
	On Currency	valid currency
	On QoS parameter	possible QoS parameter(s)
Features	Travel Regions	Domestic/International
Solution	Access Point	pointer to the WSDL file.
Feedback	Experience	success/Failure

The frame structure is used for case representation of Web services execution experiences. The case representation has a notion for describing functional and non-functional parameters, which provides a mechanism for representing higher structured real-life problems. For instance, a real world web services execution problem described in plain English representation: "Find a Trip for single person, Mr Lee; Mr Lee wants to travel from Boston to New York, with price range in total $220, He does not want to travel by road. The dates of Travel will be 27-02-2005 for departure and 01-03-2005 as return date. He prefers to pay in USD. He needs quick results (approximately in 1.5 seconds)" with solution will be transformed as frame as shown in Table 6.

Table 6 Example of a case

Travel Request	City Departure	New York
	City Arrival	Boston
Preferences	On Price range	220
	On Currency	USD
	On QoS parameter	$1.5_{executionDuration}$
Features	Travel Regions	Domestic

Solution	Access Point	http://EJ.com/ws.wsdl
Feedback	Experience	Success

Mapping Frame structure to Ontologies.

The developed framework map the frame structures to ontologies. The rules for such mapping are described in Figure 27. According to this mapping, frame and slot are represented as classes. The relationship between frame and slot is expressed in terms of properties of a frame, where the range for these properties are the slot classes. The dimensions are the properties of the slots. The possible range for these properties is the values the respective filler can derive from.

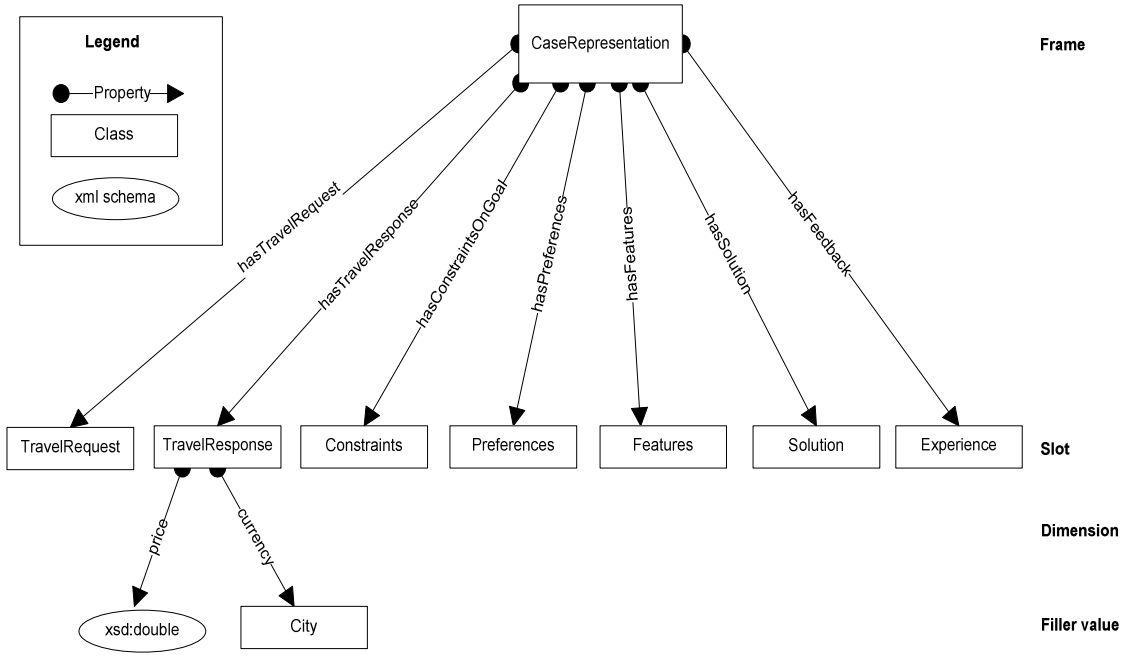


Figure 27 Mapping frame structure to semantic case representation (Travel Domain)

The framework use OWL for representing these ontologies. After applying the mapping, the ontology for the travel domain case representation is created (Figure 27), where `CaseRepresentation` class has: `hasTravelRequest`, `hasTravelResponse`, `hasConstraintsOnGoal`, `hasPreferences`, `hasFeatures`, `hasSolution` and `hasFeedback` object properties. The range for these properties is `TravelRequest`, `TravelResponse`, `Constraints`, `Preferences`, `Features`, `Solution`, and `Experience` classes respectively.

In order to exercise the noble objective of globalization of semantic descriptions, implementation used external ontologies where appropriate [77]. For instance, the `cityOfArrival` is an object property referring to the publicly available ontology

where other useful information about the specific city can be found such as country, the number of inhabitants, etc.

After modelling cases in OWL based semantic descriptions, it is possible to reason using OWL reasoner [71]. Each new case stored in the case library, will be an instance of the ontology class *CaseRepresentation*. This makes it possible to derive inference for the purpose of decision-making, which involves further phases of CBR system. The explicit values expressed in Table 6 have been semantically mapped as illustrated in the following Table 7

Table 7 Semantic Description of case

Travel Request	
City Departure	New York ([City (USA [Country])]) $_{\text{is-city-of}}$
City Arrival	Boston ([City (USA [Country])]) $_{\text{is-city-of}}$
Preferences	
On Price range	220
On Currency	USD ([Currency]) $_{\text{code}}$
Features	
Travel Regions	Domestic ([Travel Regions])
Solution	
Access Point	http://Jetservices.net/UnitedAirLines.wsdl
Feedback	
Experience	Success
Class = [class], Instance = instance ([class]), Property = $_{\text{properties}}$	

4.5. SCBR FRAMEWORK DEVELOPMENT

4.5.1. Case Indexing and Storage

The cases can be indexed based on vocabularies, which should allow retrieval of appropriate cases during the search procedure. For indexing the cases, the framework uses "partitioning case library" method, which is a variation of "flat memory indexing" technique [72]. In this indexing method, case library is partitioned based on certain vocabularies and the new problem is recognized based on the identical vocabularies to decide which partition the problem falls into. In our case study, cases are stored based on vocabulary element *Features* as presented in Table 5, which corresponds to *hasFeatures* property from the CaseRepresentation ontology class. For the travel agent case study, the possible values for this vocabulary (*hasFeatures* property) are either *Domestic* or *International* (pre-defined instances from the *TravelRegion* class) hence indexing will partition case library into two parts. The indexing is

performed based on identifying combinations of features of a case that describe the circumstances in which a reasoner might find the case useful during reasoning. To achieve this it is sufficient to consider single feature in our proof-of-concept work, however the real-world case based reasoning system require depending on the application domain more than one vocabulary term or combinations of vocabulary terms for indexing for this purpose. For example, CLAVIER [78] - a case based reasoner for design and evaluation in the domain of autoclave loading and spatial arrangements, indexes based on the autoclave parts, part layouts, part locations and part orientations.

4.5.2. Case Retrieval

Whenever a new Web service needs to be searched, the problem description involving the functional parameters and non-functional parameters are encoded using the case representation frame structure, i.e. as an instance of `CaseRepresentation` ontology as illustrated in Table 5.

The framework identifies the new problem based on the partition it falls into and then the rest of the matching is applied to cases from that partition only. This corresponds to using `hasFeatures` property value to reason whether the new problem falls under `Domestic` or `International` travel region. Based on the outcome of reasoning, the cases associated with particular partition are further investigated.

4.5.3. Case Matchmaking and Ranking

The case retrieval procedure fetches Web services that are a potential solution to the new problem. The matching process narrows down the retrieved cases to present acceptable solution(s). From the available methods for matchmaking in CBR literature, the framework uses Nearest-Neighbour Matching and Ranking using numeric evaluation function [79]. This method operates as follows:

1. Compare the similarity for each property, between the new problem and the cases retrieved. The method used for comparison depends on the type of property.

2. Quantify the weight of the similarity. A ranking is assigned to each property in accordance with its importance as exemplified in Table 8. To improve the accuracy of matchmaking process, a spectrum of functional and non-functional parameter based matchmaking criteria is employed; hence requiring such novel quantifying

mechanism to measure these parameters individual contribution to the overall of Aggregate Degree of Match (ADoM).

Table 8 Quantifying the Travel Domain case dimensions

Slot	Dimension	Importance (0-1)
Travel Request	City Departure	1.0
	City Arrival	1.0
Constraints on Goal	On Instance	0.2
	On Domain	0.8

For each case retrieved, the similarity degree is computed and the case with the highest score corresponds to the best-match. Similarity takes values between 0 and 1, which is attributed to each property for each retrieved case. The similarity comparison depends on the type of the dimension: data or object.

Object property comparisons

For semantically matching object property value of the new problem and the retrieved cases, the algorithm compares the instances. If the instances match, then the degree of match is 1. Otherwise, the algorithm traverses back to the super (upper) class that the instance is derived from and the comparison is performed at that level.

The comparison is similar to traversing a tree structure, where the tree represents the class hierarchy for the ontology element. The procedure of traversing back to the upper class and matching instances is repeated until there are no super classes in the class hierarchy, i.e. the leaf node for the tree is reached, giving degree of match equal to 0. The degree of match (DoM) degree is calculated according to the following equation:

$$DoM=\frac{MN}{GN}$$

Equation 1 Degree of Match (DoM)

Where MN is the *Total number of matching nodes in the selected traversal path*, and GN is *Total number of nodes in the selected traversal path*

For example, for the request in Figure 28, case#1 will return a degree of match of 0 because no matches are found while traversing the ontology tree until the leaf node is reached. However, for case#2, the degree of match will be 2/3=0.67 as the instances (New Jersey, New York) does not match but the instances of the Country super class

match.

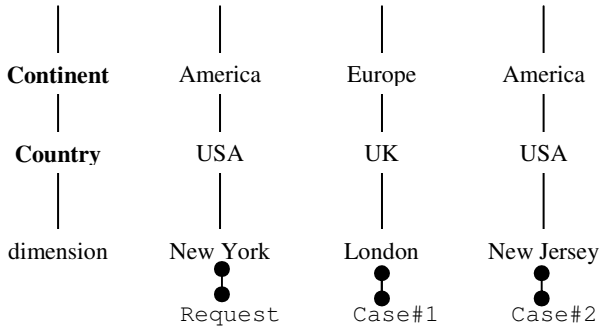


Figure 28 Semantically matching object properties

Data type property comparisons

To compare data type properties, like the price range or the value of QoS (e.g. execution time), the qualitative regions based measurement method is used, the closer the value in a retrieved case is to the value in the request higher the similarity coefficient is.

For each data type property, this formula used is: $|V_r - V_c| \leq X.[V_r|$, where V is the value of the property in the request r or in the retrieved case c and X the factor of tolerance. Thus, a factor of tolerance of 0.9 means the value of the retrieved case should be in ±10% region in relation to the value of the request. The optimum tolerance value is determined by the administrator and can be calculated emperically.

Computing the overall similarity value

Overall similarity is evaluated by computing the aggregate degree of match (ADoM) for each retrieved case according to the following equation:

$$ADoM = \frac{\sum_{i=1}^{n} W_i \times sim(f_i^N, f_i^R)}{\sum_{i=1}^{n} W_i}$$

Equation 2 Aggregate Degree of Match (ADoM)

Where n is the number of ranked dimensions, W_i is the importance of dimension i, sim is the similarity function for primitives, and f_i^N and f_i^R are the values for feature f_i in the new problem and the retrieved case respectively.

The evaluation function sums the degree of match for all the dimensions as computed in the DoM step and takes aggregate of this sum by considering the importance of dimensions.

The accuracy of Web services discovery and matchmaking is dependent on the right combination of indexing, ranking and the existence of adequate cases in the case library.

Although the chosen case study for this work is from the travel domain, the modular, ontology-driven design of framework makes it application-independent and allows its seamless reuse for other application domain.

4.6. Preliminary Implementation

To perform a case study, the SCBR framework for the travel domain is implemented. The implementation of this framework uses semantics extensively to implement both the utility ontologies describing the components of the Case Based Reasoner and the domain ontologies that describe the profile of the Web services in the case library with a semantic representation.

OWL was ontology language of choice and Pellet [71], a Java based OWL reasoner as ontology engine in favour of the more popular Jena [70] as it supports user-defined simple types. Pellet was used to load and verify (type and cardinality) ontology class instances of user requests and candidate cases.

Figure 29 illustrates a snapshot of the GUI developed for the matchmaking framework. The interface allows different options to two types of users: The case administrator, who is responsible for case library maintenance and a case requestor who wants to retrieve Web service for a trip. The implementation provides case administrator privileges in order to perform case maintenance activities: case seeding, rankings and setting up the threshold value, i.e., the acceptable value for matching coefficient. The case requestor can also setup rankings, which will be applicable for a particular session.

While seeding the case library with a new case, the interface assists the case administrator in creating the ontology instances. The main feature of the framework is that the program creating the user Interface uses *CaseRepresentation* class from the ontology (Figure 27) to form the GUI elements. The subsequent properties from the *CaseRepresentation* class and the range for those properties constitute the rest of the user interface. This allows maintaining transparency from the service requestor and hides complexity of the reasoner. For example, one GUI component in Figure 29 shows the mode in which case administrator is assisted in creating the instance of *TravelRequest* class while entering *hasTravelRequest* property of

CaseRepresentation class. The value entered for particular property is validated against the range and cardinality from the ontologies. The framework also makes the possible instances available once they are created. For example, in Figure 29 while entering values for the *TravelRequest*, city instances *Boston* and *New York* are available for re-use. As a result of seeding a new case, framework creates an ontology instance of *CaseRepresentation* class and stores into case library.

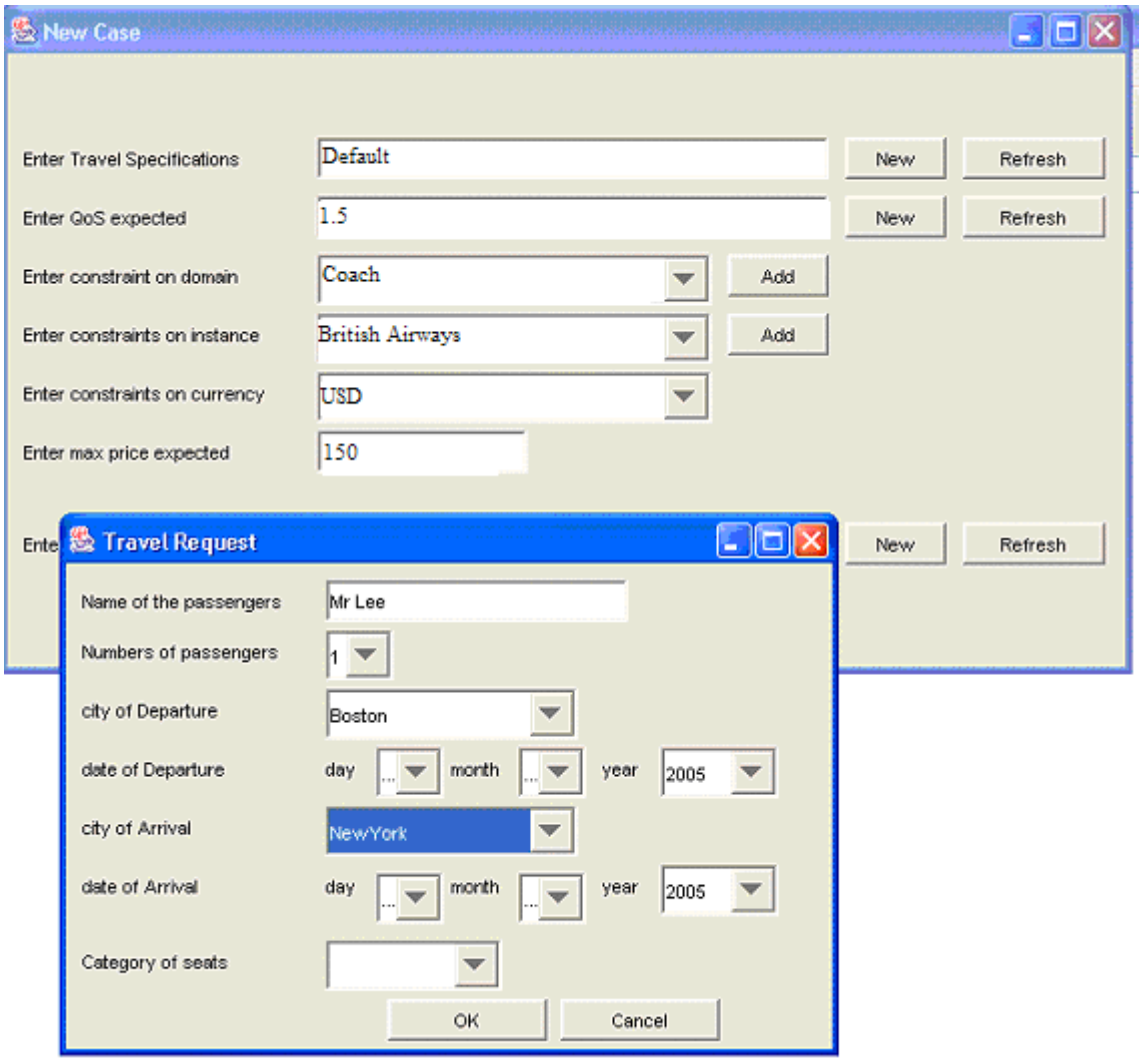


Figure 29 Seeding the case library

For case searching, the framework offers the requestor similar interface to that available for case administrator, and creates semantic description for the new problem parameters. The generated index for such semantically described problem governs the decision regarding which partition the problem falls into and the cases from that partition are retrieved for further matching. This matchmaking procedure is implemented in accordance with the algorithm described in the section 4.5. The result of the match-making procedure displays the case instances, which have similar problem situation to the new problem. The framework also displays the aggregate matching

coefficient associated with such suggested case instances for the requestor to view and make appropriate selection.

4.7. Preliminary RESULTS

At this initial stage of the development, the focus of the experiment was to validate the logic for the matchmaking framework, rather than testing a fully working prototype. In order to consolidate the test process, the experiment applied different rankings against each test case and associated them with a specific profile. The profile represents a group of users that have similar requirements for the travel request. For instance, the business profile stands for corporate users, who have to travel frequently; therefore a high standard of comfort is a significant element of choice. These users also need reliability of services while cheap fare is not critical because firms very often have contracts with travel companies. On the other hand, for regular users represented by personal profile, cost is of paramount importance.

The three other types of users are mainly based on specific comparison properties: the economy profile retrieves cases which price never exceeds a user-defined maximum amount; travel medium profile is specific for constraints on travel domain as well as instances; and the enterprise profile is useful for companies which are interested in using reliable services. The later can be important if contracts between the company and some Web services exist so that they can restrict other services.

Table 9 shows the rankings of profile systems. Example of constraint on domain is traveller's reluctance to use a certain transport such as Air transport; example of constraint on instance is the exclusion of certain airline from the search such as excluding easyjet airline. The Quality of Service parameter is represented as a single parameter, but in this experiment it is expressed as the availability and response time of the service.

Table 9 User Profiles

	Property			
Profile	*Constraints on Domain*	*Constraint on Instance*	*Price*	*Quality of Service*
Business	0.6	0.4	0.1	0.5
Personal	0.4	0.7	0.5	0.2
Economy	0.4	0.2	1	0.1
Travel Medium	1	0.8	0.3	0.2
Enterprise	0.3	0.1	0.2	1

Table 10 Case Instances and Satisfactory measurements

	User				
Case	*Business*	*Personal*	*Economy*	*Medium*	*Enterprise*
CaseInstance#1	0.45	0.37	0.6	0.19	0.22
CaseInstance#2	0.36	0.26	0.21	0.22	0.24
CaseInstance#3	0.22	0.17	0.16	0.27	0.11
CaseInstance#4	0.1	0.11	0.04	0.03	0.05
CaseInstance#5	0.12	0.1	0.13	0.11	0.12

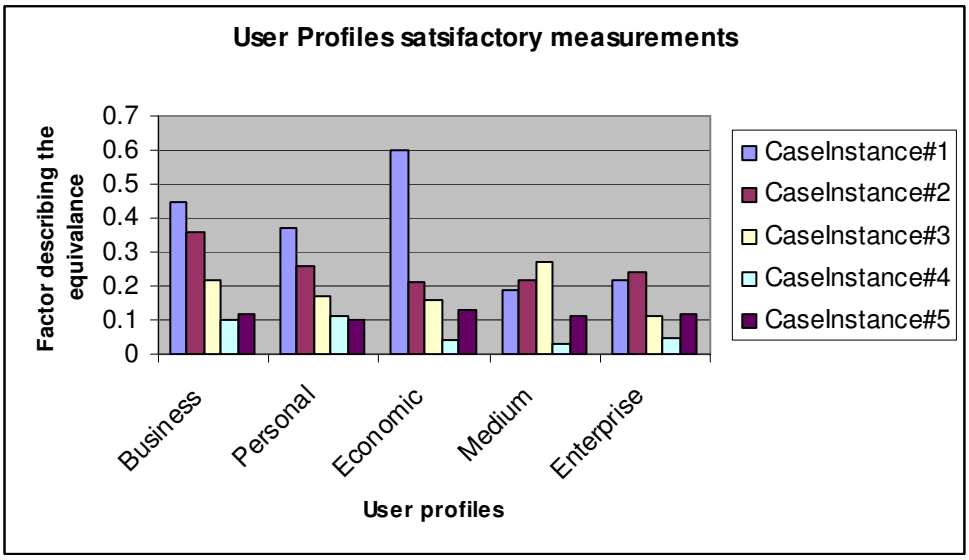


Figure 30 Case Instances and Satisfactory measurements

Table 10 and corresponding graph in Figure 30 highlight the fact that the services can serve different circumstances differently. For example some cases (Web services experiences) such as `CaseInstance#1` present satisfactory results to all users, while case `CaseInstance#3` is more suitable for business category of users than the users from the enterprise profile. According to conducted investigation, there is no similar framework that allows comparing Web services on this granular level by analyzing execution experience of candidate services.

4.8. Related Work

Semantic descriptions are increasingly being used for exploring the automation features related to Web services discovery, matchmaking and composition. In [80] such semantic-based approach is described. The authors use ontology to describe Web services templates and select Web services for composition by comparing the Web service output parameters with the input parameters of other available Web services. A constraint driven composition framework in [66] also uses functional and data semantics with QoS specifications for selecting Web services. In similar spirit,

DARPA's OWL-S (Ontology Web Language for Web services) is the leading semantic composition research effort. The OWL-S ontologies provide a mechanism to describe the Web services functionality in machine-understandable form, making it possible to discover, and integrate Web services automatically. An OWL-S based dynamic composition approach is described in [44] , where semantic description of the services are used to find matching services to the user requirements at each step of composition, and the generated composition is then directly executable through the grounding of the services. Other Approaches use Artificial Intelligence planning techniques to build a task list to achieve composition objectives: selection of services and flow management for performing composition of services to match user preferences. Authors in [47] uses Golog – AI planning Reasoner for automatic composition, while in a similar spirit some other approaches such as [42] have used the paradigm of Hierarchical Task Network (HTN) planning to perform automated Web service composition. These approaches use semantics for automatic Web services discovery, but they overlook the Web service run-time behaviour in the decision-making process.

Experience based learning using CBR is a relatively old branch of Artificial Intelligence and cognitive science and is being used [58] as an alternative to rule-based expert system for the problem domains, which have knowledge captured in terms of experiences rather than rules. However, case based reasoning for Web services was initially documented in [59], where the developed framework uses CBR for Web services composition. In their approach, the algorithm for Web services discovery and matchmaking is keyword based and has no notion for semantics. This affects the automation aspects for Web services search and later for composition. A similar approach is described in [60], which proposes an extension of the UDDI model for web services discovery using category-exemplar type of CBR, where web services are categorized in domains and stored as exemplar of particular domain. Their implementation of CBR reasoner facilitates UDDI registry by indexing the cases based on the functional characteristics of Web services. However, the approach does not take into consideration the importance of non-functional parameters in service selection and the use of semantics at CBR level is peripheral as they primarily use the UDDI based component for service discovery. The UDDI registry based publication and discovery is text-based leaving little scope for automation. The SCBR framework consumes semantics extensively and achieves the automation required for Web service discovery and matchmaking. Use of ontologies also makes framework extensible and reusable.

The application of CBR, semantic web and Web services are common technologies in this effort and the efforts in [82] albeit with different objectives. Their work is based on consuming these technologies to assist the procedure of semantic web services creation using CBR approach, while our main concern is services composition. The authors present INFRAWEBS project to implement Semantic Web Unit (SWU) which is a collaboration platform and interoperable middleware for ontology-based handling and maintaining of semantic web services. The framework provides knowledge about a specific domain and relies on ontologies to structure and exchange this knowledge to semantic web services development process.

There are also a number of existing approaches which apply CBR for workflow modelling. Madhusudan *et al.* in [62] propose an approach to support workflow modelling and design by adapting workflow cases from a repository of process models where workflow schemas are represented as cases and are stored in case repositories. The cases are retrieved for a problem which requires similar business process to solve the problem. The description and implementation language of framework is based on XML and main focus is on assisting workflow designer in creating business process flows.

In similar spirit, [63] represents adaptive workflow management system based on CBR and targets highly adaptive systems that can react themselves to different business and organization settings. The adaptation is achieved through the CBR based exception handling, where the CBR system is used to derive an acceptable exception handler. The system has the ability to adapt itself over time, based on knowledge acquired about past execution experiences that will help solve new problems. The approach discussed in this chapter concentrates on Web services as a unit of computation to take advantage of highly accessible and loosely coupled nature of Web services technologies. The focus is on utilising service execution experiences to best serve user requirements and encode the framework with semantics.

Recent work on Web services discovery by Zaremba *et al.* in [83] have drawn similar conclusion about considering run-time behaviour of services. They realize the limitation of matching static behaviour of services in semantics and non-semantics approaches and propose that service discovery which operates on abstract descriptions of services needs to be further elaborated in order to return results of concrete services satisfying concrete goals. For this purpose they utilize instance data using data-fetching algorithm from the

service provider at discovery-time. The authors use abstract state machine formalism [84] to model the interface allowing scalable interactions with a service provider for specific discovery sessions. However a drawback of interacting with a service during the discovery phase can be a significant communication overhead and in circumstances where the service provider does not provide interface for data-fetching services or does not provide such service at all. In SCBR framework, reliance is on existing service and service interface to capture the knowledge required to evaluate the run-time behaviour of services.

4.9. Limitations of SCBR framework

This section outlines the limitation of the SCBR framework. The limitations mentioned are generic to the concept of using CBR for Web services composition rather than specific to the implementation of our algorithm. It is envisaged that addressing these limitations by extending SCBR framework to cater for generality of purpose and apply CBR adaptation mechanism for composition as explained in the following chapter.

4.9.1. Limited intelligence

The current framework addresses the problem of automatic Web services discovery and matchmaking by annotating Web services execution experiences and storing them into a case library. The search considers domain-specific criteria for the user preferences and represents Web service which solved the similar problem in the past. However, the framework assumes that the case library contains suitable cases for every possible problem. This assumption is not always satisfied considering the vast number of problems and problem parameters. For example, a new problem might contain new circumstances in terms of problem constraints and preferences which were never evaluated in existing cases, hence necessitates evaluating existing cases to match these new circumstances, i.e. in travel domain case study, if user in his service request specifies preference on *Hotel* and *Airline* domains, then the framework has no alternative to address a situation where the case library contains cases that only individually involves *Hotel* and *Airline* domain but not the combination of the two. Moreover, the framework also needs to deal with situations where the aggregate degree of match (ADoM) is below the domain-specific expected degree of match set by the domain administrator and to also deal with negative user feedback, where the matched services are not acceptable to the user.

4.9.2. Extension to Web services composition

Web services composition is essentially a service discovery process, where services are discovered to meet the service request and are integrated when the existing services are not sufficient to achieve the required objectives. Web services composition can also offer new opportunities by providing new, value-added services through facilitating cooperation between existing application services. The approach so far considers utilizing case library to find suitable services (service experiences) and needs to be extended to consider composition to address the situations where the available service execution experiences does not satisfy service request. The extension shall also cater for creating value-added Web services out of existing services.

4.9.3. Expressiveness in case representation

Although the chosen case study for this work is from the travel domain, the modular, ontology-driven design of framework makes it application-independent and allows its seamless reuse for other application domain. However, the work outlined in this chapter lacks an explicit specification of case representation that is domain-independent and can serve as a blue-print to implement any possible domains.

4.9.4. System performance while using universal ontologies

The other enhancement to the current SCBR framework should deal with the response time of the framework. The use of universal re-usable ontology to build and extend our framework can increase the overhead incurred by parsing the semantic descriptions as the accessing ontologies are subjected to network delays and source availability.

4.10. Conclusions

Semantic description of Web services' profiles paves the way for automating the discovery and matchmaking of services since it allows intelligent agents to reason about the service parameters and capabilities. However, the accuracy of such automatic search mechanism largely relies on how soundly formal methods working on such semantic descriptions consume them.

In the second phase of this research work, it was stressed that consideration of the execution values is important for the semantically described non-functional Web services parameters in decision making regarding Web service adequacy for a particular

task. This is because the service behaviour is impossible to presume prior to execution and can be only generalized if such execution values are stored and reasoned upon to assess the service capability. To implement a framework that supports storing and utilizing Web services execution experiences, an experience-based reasoning methodology is required. The AI planning and intelligent agent systems are rule-based reasoning methods and do not support such level of experience-based reasoning methodology. The exhaustive literature survey resulted in identifying Case Based Reasoning (CBR) methodology as a potential solution. CBR allows reasoning based on past experiences of the computational units and is widely used as an alternative to rule-based expert system for the problem domains, which have knowledge captured in terms of experiences rather than rules.

A Semantic Case Based Reasoner (SCBR) was implemented that captures Web service execution experiences as cases and uses them for finding a solution for new problems. One of the main features of this framework is the extensive utilization of semantic web technologies in describing the problem parameters and in the implementation of the core components of the framework: representation, indexing, storage, matching and retrieval. These components are modelled based on ontologies, making the application logic captured within semantic descriptions and addressing the problem of interoperation between independently developed reasoning engines. Without this interoperation, the reasoning engines remain imprisoned within their own framework, which is a drawback, especially that most engines usually specialise in servicing a particular domain, hence interoperation can facilitate inter-domain orchestration. We believe that in this work we took a small step towards standardization at the reasoner level by describing the CBR reasoning model semantically.

In this chapter the preliminary experimental results of SCBR framework was also presented, which informally proved the correctness of the approach by demonstrating the advantages of considering past experiences of Web services and testing them based on a classification of user groups into profiles that have standard set of constraint rankings. The research concluded that there is no similar framework that allows comparing Web services on this granular level by analyzing execution experience of candidate services and is only possible with an experience-based framework such as the SCBR framework.

The semantic approach for modelling CBR reasoner is a promising solution as the

framework achieves required automation and makes reasoner extensible and reusable. In the next chapter the extension of the matchmaking framework for Web services composition to solve framework limitations is presented.

5 Chapter Five: Extending SCBR for Web Services Composition

In the previous chapter, a Web service discovery and matchmaking approach based on case based reasoning was introduced. A general idea of such approach is inspired by the provision of considering past execution experiences of solutions satisfying problems similar to that requested by the end user. The framework was termed as SCBR (Semantic Case Based Reasoner) as it utilizes interpretable semantic conceptualization of domain-specific criteria and user preferences to find Web services execution experience that solved a similar problem in the past.

The previous chapter also highlighted limitations of SCBR framework with regards to limited intelligence and the expressiveness of the case representation. In addition SCBR framework is based on the assumption that the case library contains suitable cases for every possible problem. This assumption is not always satisfied considering the vast number of problems and problem parameters. For example, new problem contains new circumstances in terms of problem constraints and preferences which were never evaluated in existing cases, hence requires evaluating existing cases to match these new circumstances. Moreover, the framework also needs to deal with situations where the aggregate degree of match (ADoM) is below the *domain-specific* expected degree of match set by the domain administrator and to also deal with negative user feedback, where the matched services are not acceptable to the user.

In this chapter an aspect of CBR - case adaptation is explored in order to overcome the limitations discussed above. The case adaptation process is applicable when the available cases cannot fulfil the problem requirements, so matchmaking is attempted by adapting available cases. In this process existing framework is extended with the following:

1. A general case representation format that is applicable to any application domains.

2. Case adaptation is modelled to extend the matchmaking mechanism. The extension will address the scenarios where the available cases are not sufficient to solve new service request. An account of how this will also address the problem of Web services composition is given.

3. A case study based validation of the adaptation algorithms.

In this chapter an extension of SCBR is proposed which is termed as e**X**tended **S**emantic **C**ase **B**ased **R**easoner (XSCBR), to resolve the problem of Web services composition. In section 5.1 the design decisions to overcome limitations of SCBR framework is introduced. In section 5.2 the XSCBR framework for Web service composition based on case adaptation is presented. Finally, conclusions are outlined.

5.1. Design Decisions to Overcome Limitations of the SCBR Framework

In CBR, case is a contextualised piece of knowledge representing an experience. It contains the problem, a description of the state of the world when the case occurred, and the solution to this problem. The solution contains elements to answer the problem. In SCBR, frame structures for describing the elements of a case are adopted and transformed to the OWL ontologies. The case description in SCBR highlights the methodology for using ontologies for case representation; although the exact semantics of case description parameters are left to developer's interpretation, hence making case description domain-dependent and raising developer transparency issues. For example, the case study on travel domain includes `CaseRepresentation` class with `hasTravelResponse`, `hasConstraintsOnGoal`, and `hasFeature` object properties where range for these properties are `TravelResponse`, `Constraints`, and `Feature` classes respectively, however the guideline as to which properties to include in inputs, outputs or other components of case representation (i.e., a generic case representation mechanism) is not addressed in the framework.

Moreover, the solution component of the previous framework only focuses on the physical location of the Web service as it serves the purpose of performing Web services discovery where the user only needs access point of the selected service to utilize service at their end. In this chapter, the emphasize is on the fact that if the existing solutions are not sufficient to solve the current problem, then by using case adaptation we can modify an existing solution so that it fits new problem. This process

will require description of the composition scheme to be included as part of solution component of case representation in order to make necessary changes in the composition scheme.

In this section generic case representation format is outlined which is inline with existing methodology for describing case elements using OWL ontologies and addresses aforementioned requirements on generalization of use and inclusion of solution component.

5.1.1. Modifying Case Representation

The motivation is to specify a generic case representation schema which is applicable to heterogeneous application domains hence to the heterogeneous services in these domains. To achieve this, case representation shall cater for services with different descriptions from that required by the composition. This is due to the fact that in the majority of SOA implementations, service providers have different service description formats to those of the composers hence a domain independent, generic representation will address real world scenarios where providers can have different service descriptions than the expected by the composers, clearly benefiting the SOA community.

The requirement to consider the facilitation provided to the service requestor in case representation is also realized. Existing approaches for Web services discovery and composition lack standard representation for refining user requests. For example, a service requestor does not have the means to specify constraints and preferences on the final results such as output currency must be Euro. The existing approaches do not include elements to specify such granular service requests.

Figure 31 outlines an example of a case representation scheme which will be applicable for web services discovery and matchmaking in heterogeneous domains. In this representation, an organization could provide *CaseService* with a *CaseRepresentation* format. The figure shows the developed ontology for *CaseRepresentation*, where the *CaseRepresentation* class consists of object properties including: *hasInput*, *hasOutput*, *hasConstraint*, *hasPreference* and *hasSolution*. These properties have value range *Input*, *Output*, *Constraint*, *Preference* and *Solution*. An organization specifying their case representation using *CaseService* should adhere to this generic

representation of *CaseRepresentation* class and implements the variable components of the representation in customized manner to encode the domain parameters. Hence case library will be consisting of a variety of service execution experiences consisting of numerous case representations, which suits to real world scenarios.

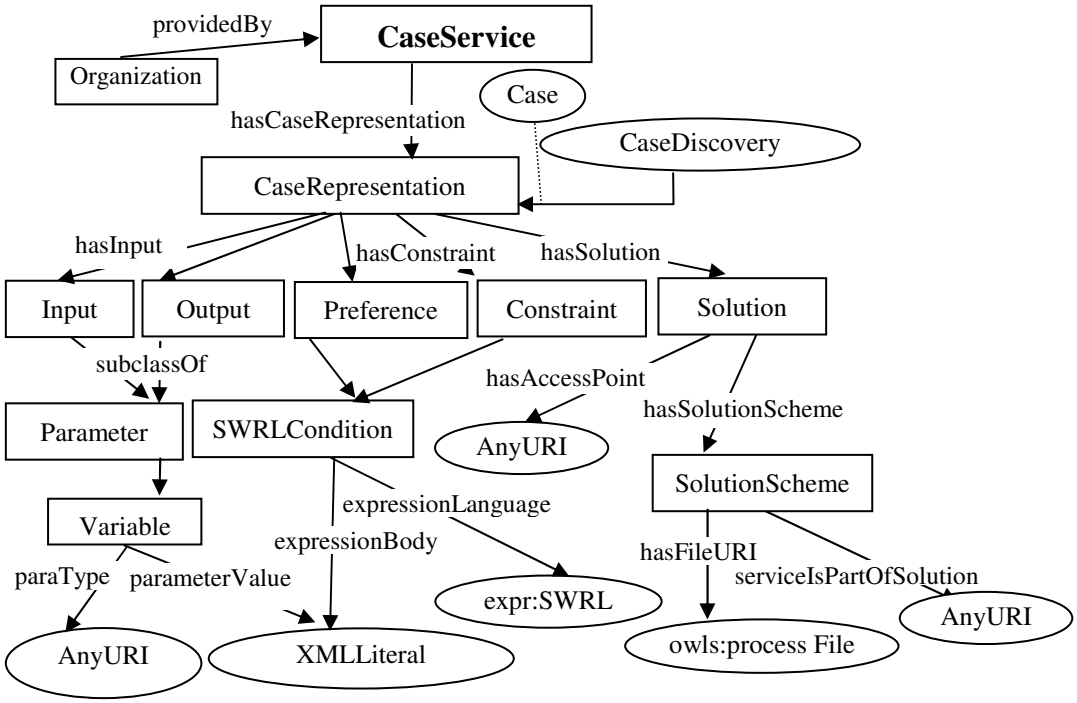


Figure 31 Generic Case Representation

A service provider can map their service inputs and outputs to *Input*, *Output* from *CaseRepresentation* class of a specific organization and can submit service descriptions to composers.

A service requestor can use the *Constraint* and *Preference* components of *CaseRepresentation* to narrow-down their search, we thus fulfil our goal of providing facilitation to service requestor as the service requestor can use these components to query granular level request and is transparent from the complexity of the framework.

Case Representation

The use Web Ontology Language (OWL) for constructing ontologies is continued while the use of Semantic Web Rule Language (SWRL) [85] for defining rules is proposed.

The ontology in Figure 31 for case representation has *CaseRepresentation* class

with object properties including: *hasInput*, *hasOutput*, *hasConstraint*, *hasPreference* and *hasSolution*. These properties have value range *Input*, *Output*, *Preference*, *Constraint* and *Solution*. *Input* and *Output* classes are grounded in *Variable* class while *Preference*, *Constraint* are grounded in *Condition* class, and *Solution* in *SolutionScheme* class respectively.

Some of the properties and descriptions are similar to OWL-S descriptions, as the intention is to extend OWL-S descriptions for fulfilling the objectives of building domain-independent case representation format. OWL-S has been a significant semantic web based web services standard [86] and this work provides backward capability with the OWL-S descriptions. OWL-S specification provides grounding in WSDL hence the service providers with existing services can utilize the OWL-S specification for semantically annotating their Web services. Similar way, they shall be able to use this case representation schema which extends OWL-S description in the area of non-functional parameters and in providing elements to support the service requestor in searching for Web services

The *CaseRepresentation* class has two instances: *Case* and *CaseDiscovery*. *Case* is used for describing various web service execution experiences, while *CaseDiscovery* is used while searching for cases that fulfil user requirements from the case repository. Both use different components of the *CaseRepresentation*: *Case* uses *Input*, *Output*, *Feature*, *Solution* and *Feedback* to store execution experiences. While *CaseDiscovery* uses *Input*, *Output*, *Constraint*, *Preferences* and *Feature* to formalize a search request.

The variable classes *Input* and *Output* are subclasses of the *swrl:Variable* class which achieves variable status by defining parameters using a resource URI as a *ParameterType* and XML Literal as *ParameterValue*. Other variable classes *Constraint* and *Preference* on search are achieved by defining them as *SWRLCondition* using SWRL as description language and *XMLLiteral* to encode such condition. SWRL extends language expressivity of OWL with horn-like first order logic rules. We here re-used publicly available semantic descriptions with namespaces swrl[3] and expr[4]. The framework currently supports conditions defined only in Semantic Web Rule Language (SWRL).

[3] http://www.w3.org/2003/11/swrl#

[4] http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#

Solution and *Feedback* concepts have fixed semantics. Figure 32 highlights semantics for *Solution* components. *Solution* contains an object property *hasAccessPoint* which points to the access point of Web service, which could be a WSDL file or a Web based access point for the service. To formalize the detail of the solution, SCBR framework uses a pointer to an OWL-S process file as the result of *hasSolutionScheme*. In section 5.1.2 the components of OWL-S process model are revisited which are consumed in this framework. *serviceIsPartOfSolution* is an important part of the *Solution* class as it contains the domain based URI for the candidate solution services.

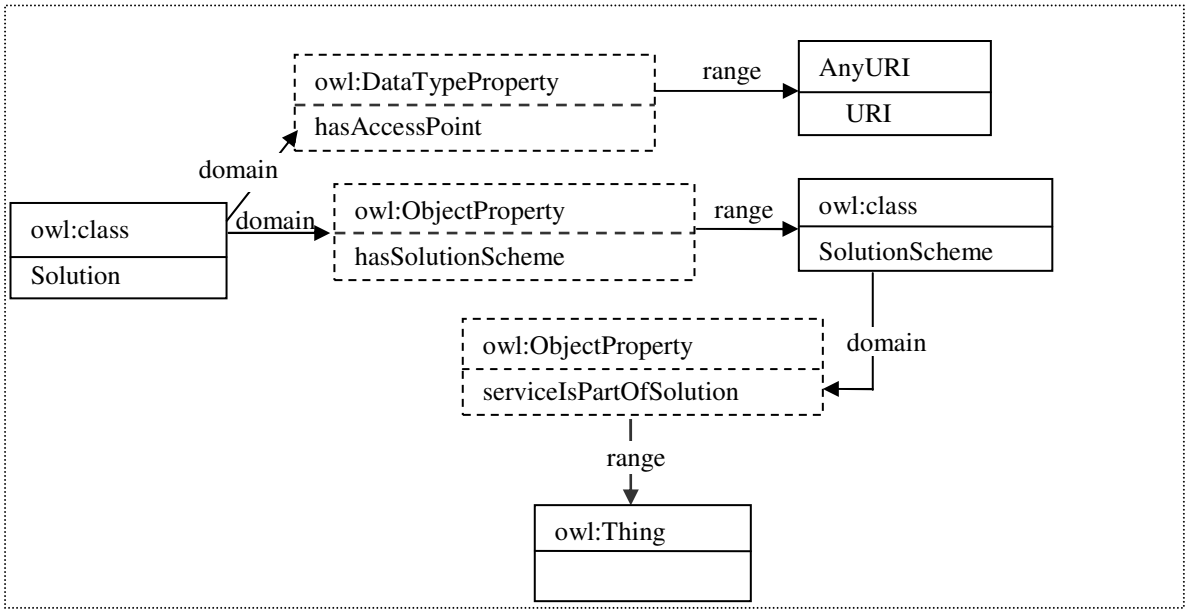


Figure 32 Solution description

Using Case Representation

An example of a semantically-encoded *CaseDiscovery* representation for travel domain is illustrated in Table 11. Note the use of rules to define the constraint conditions. For example, the rule *Constraint on Currency* outlines requestor's constraint by specifying the fact that "If *OutputCurrency* is *y*, *ExpectedCurrency* is *USD,* and *y* is not equal to *ExpectedCurrency* then *y* will satisfy requestor's constraint and could be a legitimate *ResultCurrency*".

Table 11 Example of a Travel Domain case

City of Arrival	<Input:cityArrival "http://localhost/onto/City.owl#Boston"
Date of Arrival	<Input:dateArrival>2007-09-01
Constraint price	parameterValue(ExpectedPrice,200) ∧ parameterValue(OutputPrice, x) ∧ lessthanorequal(x, ExpectedPrice, true) ⇒ parameterValue(ResultPrice, x)

Constraint currency	parameterValue(OutputCurrency, y) ∧ parameterValue(ExpectedCurrency, USD) ∧ notEqual(y, ExpectedCurrency, true) ⇒ parameterValue(ResultCurrency, y)
Constraint QoS	ExecutionDuration(OutputQoS, y) ∧ ExecutionDuration(ExpectedQoS,1.5) ∧ lessthanorequal(y, ExpectedQoS, true) ⇒ ExecutionDuration(ResultQoS, y)

It is important to note that constraint conditions are not only simple equality checks i.e., if one individual is equal to other individual or not, but are also complex in terms that they include mathematical and logical operations, i.e., one value is greater or less than the other etc. Therefore, in case representation, we support rules for describing such complex constraint relationships as rules capture such relationships that are not possible to represent using OWL alone. Apart from their obvious ability to deal with complex relationship, rules also provide more expressive power with respect to properties - for example, allowing one property to be inferred from a composition of others. A well-known example is the assertion that the composition of "parent" and "brother" should imply "uncle"—that is, uncle(x, z) ←parent(x,y) ^ brother(y, z), a relationship that can't be captured using OWL. This kind of relationship between properties is quite common and is certainly useful in applications as varied as medical terminologies and Web service descriptions [87]. Our framework utilizes SWRL for implementing such rules.

To highlight the need and benefit of using rules for conditions, it will be helpful to contrast new case representation format in XSCBR with the previous format of case representation in SCBR. If considering the previous version of case representation, in the absence of rules such constraints were represented as shown in Table 12, where the system uses OWL alone for the descriptions. Interpreting constraints defined in OWL requires customized logic to reason on them, where the reasoner needs to know the logic explicitly. In contrast, when defined with rules it is possible to shift the reasoning burden from reasoner level to the knowledge representation layer or the semantic definition layer. This is to reduce the number of reasoner cycles, and is achieved by using normalized, well-planned ontologies that encode the repetitive logic in the knowledge base. The reasoner cycle is the number of time the ontology reasoner is involved in reasoning tasks.

Table 12 Case Representation specific to travel domain (In previous framework)

Class	Properties	Range
Input (Travel Request)	City of Departure	Any valid city
	City of Arrival	Any valid city
	Date of Departure	Any valid date
	Date of Arrival	Any valid date
	Number of Persons Travelling	Any positive integer
Output (Travel Response)	Currency	Any valid currency
	Price	Any positive double
Solution	AccessPoint	WSDL/Web access point
	hasSolutionScheme	OWLS-process
Features	Travel Regions	Domestic/International

Figure 33 and rule illustrated in Figure 34 highlight the difference in describing a case in SCBR and XSCBR framework. For example, the SCBR framework will store a Case with (attribute, value) pair for the attribute - `PreferenceOnCurrency` = `USD`. In contrast, the XSCBR framework will store preference for currency using `rule-1` (as shown in Figure 34), which encodes logic saying that the case follows rule-1 which fires when the output currency is equal to the expected currency `USD`. Here, in contrast to SCBR framework, the reasoner does not have to remember the value for the attribute `PreferenceOnCurrency` and the `rule-1` will take care of the logic maintaining the constraint on currency. This results into economy of storage space and allows shifting the reasoning burden from reasoner level to the knowledge representation layer.

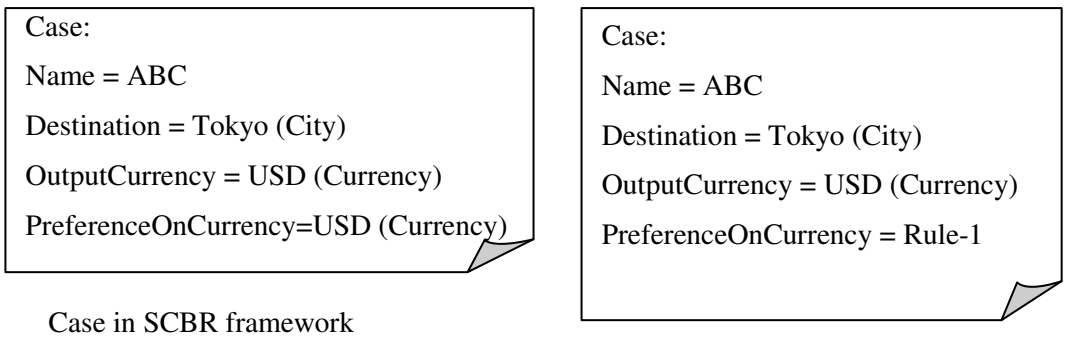


Figure 33 Comparing Cases in SCBR and XSCBR-I

Rule-1

$$\text{parameterValue}(\text{OutputCurrency}, y) \wedge \text{parameterValue}(\text{ExpectedCurrency}, \text{USD}) \wedge \text{Equal}(y, \text{ExpectedCurrency}, \text{true}) \Rightarrow \text{parameterValue}(\text{ResultCurrency}, y)$$

Figure 34 Comparing Cases in SCBR and XSCBR-II

Quality of Service Descriptions for Web services

It is worth to mention here that one of the main components of this framework is the quality of service provided by the services. We consider QoS as an important selection criterion for web services discovery and need to modify QoS descriptions from the previous framework inline with the modifications and improvements in the case representation, especially using rules. An extensive ontology was developed that supports reasoning on QoS. Following is the definition of QoS in XSCBR framework:

Table 13 QoS parameters in XSCBR

Element	Range	Semantics
providedBy	organization	Portal:Organization
Availability	IntervalZero ToOne	$SimpleType(?x) \wedge value(?x, ?y) \wedge greaterThanOrEqual(?y,0) \wedge lessThanOrEqual(?y,1) -> \text{int}\ ervalZeroToOne(?x)$
Reliability	IntervalZero ToOne	$SimpleType(?x) \wedge value(?x, ?y) \wedge greaterThanOrEqual(?y,0) \wedge lessThanOrEqual(?y,1) -> \text{int}\ ervalZeroToOne(?x)$
Reputation	NonNegativeInteger UpTo	$SimpleType(?x) \wedge value(?x, ?y) \wedge lessThanOrEqual(?y,5) -> nonNegativeIntergerUptoFive(?x)$
Execution Price	positiveDouble	$SimpleType(?x) \wedge value(?x, ?y) \wedge greaterThanOrEqual(?y,0) \wedge -> positiveDouble(?x)$
Execution Duration	positiveDouble	$SimpleType(?x) \wedge value(?x, ?y) \wedge greaterThanOrEqual(?y,0) \wedge -> positiveDouble(?x)$

The `organization` in the above ontology could be any organization which certifies or provides detail of the QoS for particular service. It could also be used in a system where the QoS is self-certified by service providers.

To summarize, following are the design decisions taken which contribute to overcoming limitations of SCBR.

- Revising case representation to make it generic and applicable for efficient annotation and discovery. The proposed generic case representation has provision for the facilitations to service participants; especially service requestor. The rules to handle

complex constraint and preferences relationships in case descriptions are introduced. The QoS is considered as an important selection criterion for web services discovery and modify QoS descriptions from the previous framework inline with the modifications and improvements in the case representation, especially using rules.

- The solution component is also semantically presented in order to adapt the solution for new problems whenever required.

The XSCBR framework opt to utilize OWL-S process model as solution schema and here the OWL-S process model is revisited in following section to explore components of process model which are used in XSCBR framework.

5.1.2. Revisiting OWL-S Process model

Favouring OWL-S over BPEL

In a later section, emphasis is given to the fact that if the existing solutions are not sufficient to solve the current problem, then using case adaptation it is possible to modify existing solution so that it fits new problem. This process (case adaptation algorithm) will require a composition scheme or a file description of the composition scheme to be included as part of solution component of case representation. To achieve a semantic workflow specification is required such as OWL-S which is described in OWL. Amending OWL based solution will be consistent with our policy of using semantics for automation as compared to using XML based BPEL composition schemas. The issue related to the current discussion is the use of non-semantic grammar for the composition specification. For the composition engine to provide automatic discovery and flow management, the process model needs to have the consideration of the semantics in the specification. The introduction of semantics within an XML centric standard like BPEL will not achieve the automation. Automating service composition with frameworks like BPEL requires a more substantial evolution, as BPEL simply represents an execution engine for pre-defined process workflows. Therefore, to achieve any reasonable degree of automation, it requires integration of intelligent reasoners that can adapt the workflow in accordance to dynamically changing goals.

In the following section components of OWL-S process model are revisited which are useful in the procedure of adapting solutions.

Overview of the OWL-S process model

The OWL-S process model supports atomic and composite services. An atomic process is a description of a service that expects one message and returns one message in response. A composite process can be decomposed into atomic or other composite process.

$$\text{CompositeProcess} \equiv \text{Process} \sqcap \exists \text{ composedOf.AtomicProcess}$$

An OWL-S atomic process corresponds to a WSDL operation. Different types of operations are related to OWL-S processes as follows:

- An atomic process with both inputs and outputs corresponds to a WSDL request-response operation.
- An atomic process with inputs, but no outputs, corresponds to a WSDL one-way operation.
- An atomic process with outputs, but no inputs, corresponds to a WSDL notification operation.
- A composite process with both outputs and inputs, and with the sending of outputs specified as coming before the reception of inputs, corresponds to WSDL's solicit-response operation.

A composite process can be described using the rich semantic of service model which supports control flow and dataflow patterns similar to workflow patterns.

Control flow

The control flow of these atomic processes within composite process is governed by control constructs such as `Sequence`, `Split`, `Split + Join`, `Choice`, `Any-Order`, `Condition`, `If-Then-Else`, `Iterate`, `Repeat-While`, and *`Repeat-Until`*.

Dataflow

When defining processes using OWL-S, there are many places where the input to one process component is obtained as one of the outputs of a preceding step, short-circuiting the normal transmission of data from service to client and back. This is one type of data flow from one step of a process to another. There are also other patterns; in particular, the outputs of a composite process may be derived from outputs of some of its

components, and specifying which component's output becomes output X of the composite is also a data-flow specification.

Consider the following tableau:

```
I1 input of: { Composite Process CP }: with output O1
 composed of
Step 1: Perform S1
Step 2: Perform S2
where S1 has inputs I11 and I12, and output O11
and S2 has input I21 and output O21
```

Each of these equalities is represented in OWL-S as a *Binding*, an abstract object with two properties: *toParam*, the name of the parameter (e.g., *I21 (S2))*, and *valueSpecifier*, a description of its value. In an effort to provide value specifications in as concisely as possible in a variety of situations, OWL-S specification provides four different types: *valueSource*, *valueType*, *valueData*, and *valueFunction*.

Modifying OWL-S process file (Algorithm *ModifyOWL-S*)

The OWL-S process is used as solution scheme in the XSCBR framework. In the adaptation algorithm, the decision to select the components of an existing solution that needs to be adapted is based on a variety of descriptions and situational parameters, i.e. functional and non-functional parameters. Once the components are identified, necessary adaptation changes are made to the solution of an existing case, which in XSCBR framework is represented by OWL-S process file. The following algorithm *ModifyOWL-S* outlines the methodology to modify OWL-S process files.

Assuming that there exists an OWL-S process which satisfies problem *P* and the process is assumed to be composite process of services S_1 and S_2. If there is a mechanism in place to verify that service S_3 is similar in function and semantic descriptions to S_1 then following is the list of main components that need to be modified in order to create a new process with composition of S_2 and S_3 which will also be able to solve problem *P*.

1. Replace *Import* URLs of S_1 with S_3
2. Replace atomic process belonging to S_1 with the functionally similar atomic process of new service S_3

3. Bindings of old service S_1 have to be replaced with functionally and semantically similar bindings from the new service S_3.

The OWL-S API [88] is used to support reading, modification and execution of the OWL-S process models. The `Input` and `Output` in `CaseDiscovery` section of `CaseRepresentation` class has similar semantics to OWL-S process model functional parameters (`Input` and `Output`), hence the compatibility through grounding exists.

5.1.3. Summary

To summarize, existing OWL-S process model is utilized for modelling adaptation in this framework. The richness of the workflow based patterns supported by OWL-S process model and provision of semantics in the specification itself were the reasons selecting OWL-S. This section outlined extension of OWL-S descriptions to support service requestor in terms of specifying constraint and preference on search. We also outlined how OWL-S process file could be modified with the help of API to insert or remove service reference to satisfy new problem requirements. The following section details the role of knowledge in making decision about service references replacement.

5.2. XSCBR for Composition using Case Adaptation

The following section presents case adaptation [58] as an extension to the SCBR framework for solving the problem of Web services composition.

5.2.1. Introduction to case adaptation

Case adaptation is termed as the REVISE phase in CBR theory and is applicable when the available cases cannot fulfil the problem requirements, so matchmaking is attempted by adapting available cases. Adaptation looks for prominent differences between the retrieved case and the current case and then applies formulae or rules that take those differences into account when suggesting a solution [89].

Case adaptation can be defined by the following formula:

$$C' = \alpha(C)$$

Equation 3 Case Adaptation

Where, C' = new case, C = old case(s) and α indicates adaptation operator.

The adaptation operator indicates the process of identifying and substituting or transforming an existing solution to fit new situations and is used in knowledge-based substitution adaptation.

Knowledge based substitutions

In CBR's matchmaking process previous cases cannot be always reused without making some adaptation to the existing solutions. The reasoning about these changes requires general and domain specific knowledge *(K)* to assist case adaptation. Under this circumstance, the Equation 3 can be reformulated as:

$$C' = \alpha(C, K)$$

Equation 4 Knowledge based Substitution

Planning based transformations

The planning based transformations can be applicable when the available solutions can not fulfil the problem requirements with normal matchmaking and discovery mechanism or by applying minor modifications using substitution based transformation. Under these circumstances, the Equation 3 can be reformulated as:

$$C' = \alpha(C, \rho)$$

Equation 5 Planning based Transformation

ρ indicates the application of planner for transformation, where classical planner handles the task of coming up with a sequence of actions that will achieve a goal.

Figure 35 shows how case adaptation fits in CBR methodology [90].

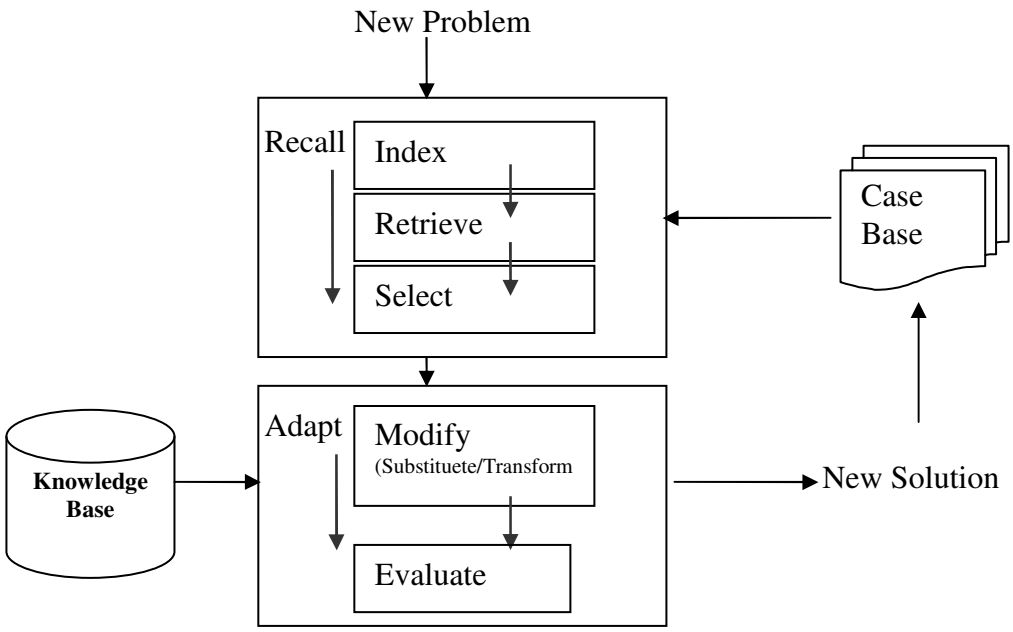


Figure 35 Case Adaptation process

5.2.2. Challenges in case adaptation

One of the major challenges in CBR is the development of an efficient methodology for case adaptation. The problem is so acute that the most effective current strategy for building CBR applications is to bypass adaptation entirely, building advisory systems that provide cases to human users who perform the adaptation themselves. The following discussion elaborates over the complex issues related to the implementation of case adaptation:

Using adaptation, a solution to a new problem results from merging the local solutions from previously solved problems to create a globally consistent solution for the new problem. However, the merging process is difficult since the local solutions typically exhibit conflicts when merged together. Furthermore, local solutions can be characterized by different representations, which further intensify the difficulty of synthesizing the global solution in ad-hoc way [91] .

As documented in this chapter, while investigating application of case adaptation to Web services composition, we came to similar conclusion as the authors of [91], where for some problems merging of local solution spawn a globally inconsistent solution. We have designed a methodology based on Constraint Satisfaction Problem (CSP) to address this challenge and discussed our experience and insights in this chapter.

In current CBR systems, rules are used for encoding adaptation knowledge. However, the ability to define those rules depends on knowledge of the task and the domain that

may not be available a priori [92]. In the XSCBR framework, a knowledge-intensive approach is advocated to automate the process of adaptation in CBR inspired Web services discovery and composition problem. We believe that the aforementioned challenge regarding availability of knowledge applies to any knowledge-intensive approach. The thesis here is that Web services composition is a developer-intrusive problem solving method, the automation of which requires the reasoning about domain-specific knowledge at their disposal. Although the XSCBR framework requires a priori knowledge of domain, the amount of knowledge available incrementally increases through the life cycle of the framework as more cases are added to the case library. In addition, the framework allows rules to be added at any time in the framework.

5.2.3. Case Adaptation in XSCBR framework

In XSCBR framework, when the existing web services experiences in their original form are not sufficient to satisfy current request, the framework attempts to relax the case restrictions under which a solution is acceptable. Figure 36 shows the holistic CBR methodology to achieve Web services composition using the REVISE cycle [62].

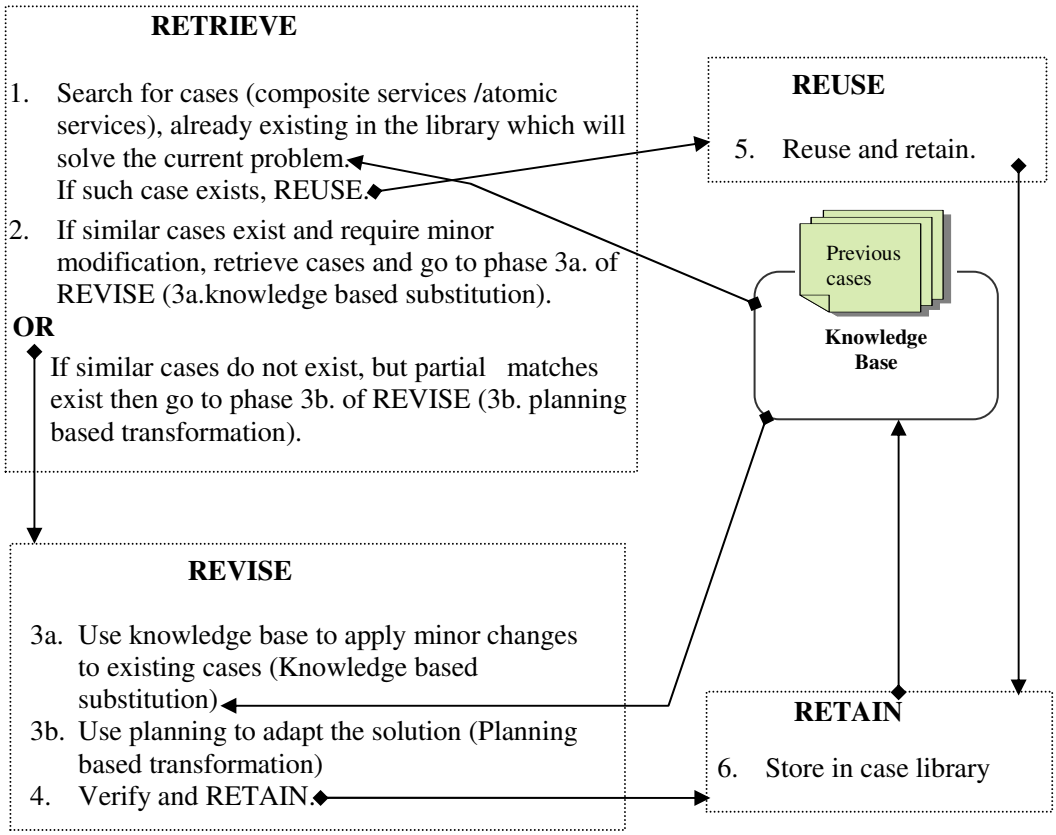


Figure 36 CBR methodology for Web services composition

5.2.4. Knowledge based substitutions in the XSCBR framework

In Web services' matchmaking process previous experiences cannot be always reused without making some changes. Knowledge based substitutions (KBS) is the process which signifies general and domain specific knowledge (K) to model these changes.

In the XSCBR framework, a knowledge-intensive approach is advocated to automate the process of adaptation in CBR inspired Web services discovery and composition problem. We believe that Web services composition is a developer-intrusive problem solving method, the automation of which requires the reasoning about domain-specific knowledge at their disposal. This approach is novel as existing approaches focus only on semantic descriptions of web services, however are oblivious to the fact that in the process of matchmaking candidates for the composition, selected web services might have individual attributes that while matching the explicit goals of the composition might invalidate the integrity of the composition workflow.

In Equation 4, K indicates the influence of general or domain specific knowledge. When applied to XSCBR framework the knowledge should be used for:

1. Targeting situations where exact match is neither available nor possible.

2. To help the reasoner in operating more efficiently by ignoring the unnecessary search and exploration.

3. To make absolutely sure that the only possible solution is transformation (which is an expensive operation involving AI planner or some sort of ultra-intelligent, resource expensive exercise). For example, if for the current problem P, the available cases in the case library are

C_1 (S_1+S_2, F_1),
C_2 (S_2+S_3, F_2)
C_3 (S_1),
C_4 (S_2)
C_5 (S_1+S_2, F_5)

The interpretation of this formalism as follows: C_1 (S_1+S_2, F_1) indicates case which has services S_1 and S_2 as a solution under circumstances defined by F_1. These circumstances could be service description, problem description, constraints and preferences in the problem request P_1.

The successful knowledge based substitution should solve a new problem with P with a possible solution $(S_1 + S_2, F_5)$ by exploring the matching cases C_1 and C_5 first before transforming C_3 and C_4 to find a solution from a scratch.

After identifying the criteria and expectations from knowledge based substitutions (KBS), following section formalizes how knowledge is represented in the semantic web.

Category of Knowledge

The knowledge to be represented can be classified into the following three broad categories:

1. Common sense knowledge

Common sense knowledge describes domain knowledge perceived by every one working on that domain. For example, domain knowledge includes representation of a consistent view of the domain entities and possible relations between them. Using the semantic web, such knowledge is specified using RDF/OWL ontologies.

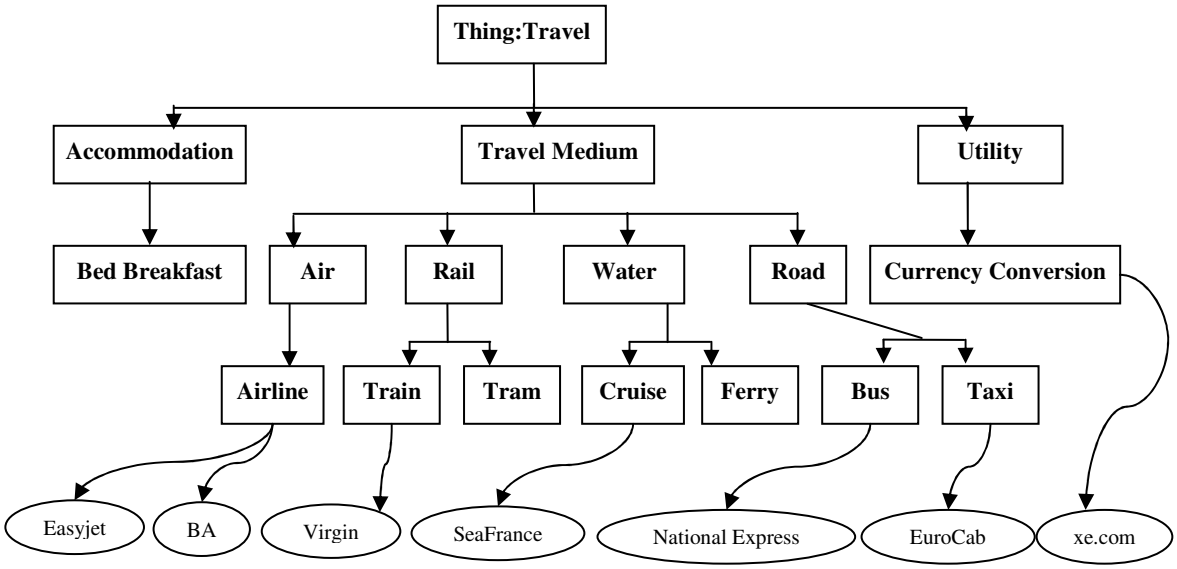


Figure 37 Travel Domain Taxonomy

In addition to domain ontologies, we also recognize the role of hierarchical taxonomies as common sense knowledge required as input to the XSCBR framework. A hierarchical taxonomy is a tree structure of classifications for a given set of objects. For example, in travel domain, taxonomy is vital to the process of matchmaking and discovery of functionally similar services. Figure 37 describes such taxonomy for the travel domain case study. Each joining service or business could be added to this

taxonomy according to the business category the service represents. For example, *EasyJet* and *British Airways* services represent *Airline* category, which is subsequently type of travel medium *Air*.

2. Heuristics

Heuristics represent ad-hoc knowledge about domain. In the XSCBR framework, heuristics are used to compile rules representing ad-hoc knowledge regarding the domain and the tasks involved in service composition. In relation to the Web services matchmaking and composition problem, heuristics are mainly useful to bridge discrepancies between the domain ontologies representing the perspective of service composer and that of service provider. For example, p_1 depicts service composer's perspective of concept c_1, and p_2 depicts service provider's perspective for the same concept with c_2. In this situation, for the successful functioning of composer, heuristic which bridges these interpretations with $p_1.c_1 \equiv p_2.c_2$ is required. In the XSCBR framework we use OWL and SWRL rule language to encode such heuristics. For example, following heuristic in OWL equates perception of city concept *Paris* in ontologies O_1 and O_2.

$$O1.Cities.Paris \equiv O2.City.Paris$$

3. Casual model

Casual model represents casual connections of some type of system or situation [93]. For example, parameter adjustment is a casual model which provides mathematical transformation of various units, i.e. following casual model adjusts parameter in feet with respect to value provided in inches.

$$\text{lengthInFeet}(i, \text{feet}) \rightarrow \text{lengthInInches}(i, \text{inch}) \wedge \text{multiply}(\text{feet}, \text{inch}, 12)$$

Casual model when available helps composer to modify solution of existing problem for the new requirements without needing transformation or even a new service to do so.

Representing knowledge using semantic web technologies

Semantic web provides a rich knowledge representation which allows a domain expert to encode the knowledge required to model the above three categories in order to achieve KBS. The knowledge can be represented in terms of concept relationship

defined by ontologies. For the benefit of the discussion, it is necessary to revisit the following components of semantic web formalism:

a. Taxonomy Relationships (TR)

Taxonomy is the concepts classification system facilitated by semantic web. `Class` and `Individual` are the two main elements of this structure where a class is simply a name and collection of properties that describe a set of individuals. Examples of relationships between concepts at the taxonomy level are `class`, `subclass`, `superclass`, `equivalent class`, `individual`, `sameAs`, `oneOf`, `disjointWith`, `differentFrom`, `AllDifferent`.

For example, we anticipate description discrepancies due to the possibility of heterogeneous service and case representations in our framework. In these circumstances, `TR` could be used to encode heuristic with explicit knowledge that a particular class or property in one ontology is equivalent to a class or property in a second ontology. In this situation, a casual model can be developed which states these facts. For example,

$$O1 \cdot TravelDomain \equiv O2.Travel$$

Where ≡ represents `equivalentClass` and `O1`, `O2` are two different ontologies.Similarly, `TR` element `sameAs` can equate two individuals.

$$O1.Cities.Paris \equiv O2.City.Paris$$

Where ≡ represents `sameAs`, `O1` and `O2` are two different ontologies, Paris (City) and Paris (City) are part of ontology `O1` and `O2` respectively.

The following table shows list of some of the elements which allow defining explicit knowledge.

Table 14 Knowledge Representation - Explicit

Element	Matching Value	Example
`EquivalentClass(≡)`	1	$O1 \cdot TravelDomain \equiv O2.Travel$
`sameAs(=)`	1	$O1.Cities.Paris = O2.City.Paris$
`differentFrom(⊆ ¬)`	0	$O1.Cities.Paris \subseteq \neg O2.City.London$

AllDifferent	0	$O1.Cities.Paris \subseteq \neg O1.City.London$ $O1.Cities.Paris \subseteq \neg O1.City.Madrid$ $O1.Cities.London \subseteq \neg O1.City.Madrid$ Above relationship will make London, Madrid and Paris mutually distinct.

Similarly *TR* could be used to describe knowledge which is not explicit however requires some level of reasoning to derive inference and relationship between two components of semantic descriptions, where matching value will be:

$$M = Dist(S, D)$$

Where, *M* = Matching value

Dist is the function which finds semantic distance between source *(S)* and destination *(D)* concepts.

To evaluate implicit relationships and the matching distance *M*, *subsumption* and *classification* are used to perform semantic tree traversal and compare concepts with respect to the semantic network tree as detailed in retrieval algorithm in section 4.5 of chapter 4. The algorithm compares concepts and if the concepts match, then the degree of match is 1. Otherwise, the algorithm traverses back to the super (upper) class that the concept is derived from and the comparison is performed at the upper class level. The comparison is similar to traversing a tree structure, where the tree represents the class hierarchy for the ontology element. The procedure of traversing back to the upper class and matching concepts is repeated until there are no super classes in the class hierarchy, i.e. the root node for the tree is reached, giving degree of match *(M)* equal to 0.

Table 15 Knowledge Representation - Implicit

Element	Matching Value	Example
Subclass (⊂)	*M=Dist(Airline.C_1, TravelMedium.C_2) =1/3 = 0.33*	$O1 \cdot AirLine \subset O1.Air \subset O1.TravelMedium$
Superclass (⊃)	*M=Dist(Airline.C_1, TravelMedium.C_2) =1/3 = 0.33*	$O1 \cdot TravelMedium \supset O1.Air \supset O1.AirLine$
disjointWith (≡¬)	*M=Dist(Air.C1, Rail.C2) = 0*	$O1.Air \equiv \neg O1.Rail \equiv \neg O1.Road$

b. Rules based relationships (RR):

Semantic Web Rule Language (SWRL) defines rule based semantics using subset of OWL with the sublanguages of Rule Mark-up Language (RuleML). SWRL extends OWL with horn-like First Order Logic rules to extend the language expressivity of OWL. It allows users to write rules to reason about OWL individuals and to infer new knowledge about those individuals. SWRL is built on OWL DL and provides more expressivity than OWL DL alone. However, it shares its formal semantics hence conclusions reached by SWRL rules have the same formal guarantees as the conclusions reached using standard OWL constructs [85].

The use of rules to define complex concept relationship is highlighted with the following example. Let's assume that there is a service provider adhering to different version of taxonomy than defined in the Figure 37, and subscribes to the category *LicencedTaxi.* While matchmaking, the composer will fail unless there is a casual model or heuristic bridging gap between categories of the service *LicencedTaxi* to the categories the composer is aware of. If rule as described below exists, then composer will be able to infer indirect subclass relationship between *Taxi* and *LicencedTaxi* and assigns matching factor of *M, where M = o1.LicencedTaxi* ⊂ o2.Taxi = ½ =*0.5.*

Taxi (? t) ^ Licence (? l) ^ hasLicence (? t, ? l) -> LicencedTaxi (?t)

After identifying how knowledge is represented in the semantic web, the following section formalizes the levels at which this knowledge is applied to achieve knowledge based substitution.

5.2.5. Applying KBS to the Existing Framework

Applied to the current framework, when the existing web services experiences in their original form are not sufficient to satisfy current request, the framework uses KBS for relaxing the case restrictions under which a solution is acceptable. The following section explains the process of utilizing KBS in the existing system. The application of KBS can be envisaged at two levels:

I. Description level

In this category using available knowledge, modification is made to the new problem and the old case descriptions to prepare the XSCBR framework for the new problem

request. For example if the new problem request adheres to a case description D_1 and there is no case with similar descriptions in the case library but there is a case with description D_2 which potentially be similar to the description D_1, then the framework uses knowledge base (KB) to verify if D_1 and D_2 are equivalent. On success, the framework employs normal matchmaking algorithm to find a suitable case which matches to new problem request. Figure 38 illustrates the aforementioned scenario.

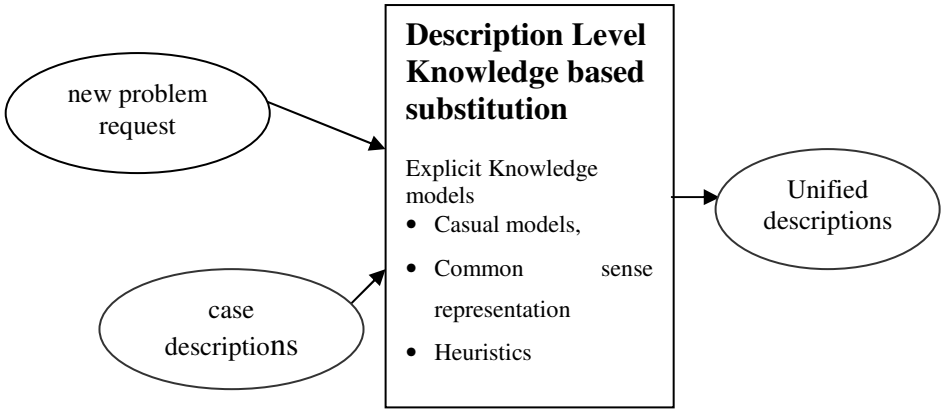


Figure 38 KBS at Description Level

II. Solution level

In this category using available knowledge, modification is made to the solution part of a candidate case (potential solution) to adapt it to a new problem request. Figure 39 illustrates Knowledge Based Substitution (KBS) application at the solution level. KBS is applied when the aggregate degree of match (ADoM) for the matchmaked candidate case C_{cand} is below the expected value for request R. This request's satisfaction problem P is represented by the following specification:

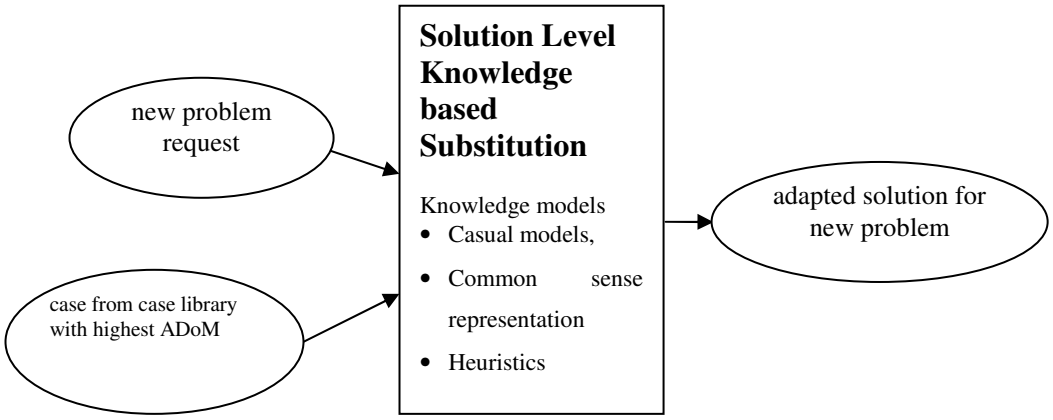


Figure 39 KBS at Solution Level

$P\ (R, C_{cand}, S_{cand}, C)$, where C_{cand} is the candidate case with S_{cand} as solution that has the highest *ADoM* for request R. Lets say, S_{cand} is a solution composed of a finite set of service instances $S_{cand}\ (i_1, i_2, \ldots\ldots\ldots, i_n)$ and C represents finite set of cases in case library $(c_1, c_2, \ldots c_e, \ldots\ldots, c_n)$ then solving problem P involves discovering a set of service instances $(i1_{sub}, i2_{sub}, \ldots\ldots\ldots, in_{sub})$ that individually match the descriptions of the instances in the candidate solution $S_{cand}\ (i_1, i_2, \ldots\ldots\ldots, i_n)$. It is important to note that the substitution service instances $(i1_{sub}, i2_{sub}, \ldots\ldots\ldots, in_{sub})$ can originate from different solution sets.

We apply following algorithm to solve problem P. We term this algorithm *ApplyKBS*.

ApplyKBS: KBS application in XSCBR

1. Apply TR or RR relationships on ontology O to find out service instances that match $(i_1, i_2, \ldots\ldots\ldots, i_n)$; let's say I_{sub} is a set of such matching service instances.

2. The case library contains cases that store runtime behaviour of service instances I_{sub}. These cases are discovered and are marked as *ball park cases*.

3. Apply the SCBR matchmaking algorithm on these *ball park cases* to find out Aggregate Degree of Match (ADoM) for each service instance.

4. The service instances $(i1_{sub}, i2_{sub}, \ldots\ldots\ldots, in_{sub}) \in I_{sub}$, with the highest ADoM are selected as substitution service instances.

5. Apply algorithm *modifyOWLS* $(S_{cand}, (i1_{sub}, i2_{sub,}\ldots\ldots\ldots, in_{sub}), (i_1, i_{2,}\ldots\ldots\ldots, i_n))$ to modify $S_{cand.}$

Evaluating the ApplyKBS algorithm

To continue with the travel domain case study, following considers two scenarios for evaluating the preliminary algorithm described in this section. For example, the system is required to find a solution for following travel request scenarios:

Scenario-1 = "Find a Trip for a traveller Mr Li; Mr Li wants to travel from London to Milan and also wants to reserve a hotel at Milan, he prefers to travel by air but wants to avoid travelling by EasyJet. He prefers to pay in GBP…. (Other requirements are snipped...)"

Scenario-2 = "Find a Trip for a traveller Mr Osman; Mr Osman wants to travel from Paris to Tokyo and also wants to reserve a hotel at Tokyo, he prefers to travel by air but wants to avoid travelling by WizzAir. He prefers to pay in EUR…. (Other requirements are snipped...)"

The matchmaking algorithm discovers that case *CaseEasyJet* satisfies scenario-1 and *CaseWizzAir* scenario-2 however the Aggregate Degree of Match (ADoM) is below expected value. Table 16 applies preliminary algorithm to evaluate the possibility of adapting *CaseEasyJet* and *CaseWizzAir* for the respective travel requests.

Table 16 Evaluating ApplyKBS Algorithm

Algorithm steps	Applying to scenario - 1 (follow Table 17)	Applying to scenario – 2 (follow Table 18)
1. Apply *TR* or *RR* relationships on ontology *O* to find out service instances that match S_{cand} $(i_1, i_2, \ldots\ldots, i_n)$; let's say I_{sub} are such matching service instances.	Applying *TR* and *RR* relationship on travel domain ontology *O* to find service instances matching to *EasyJet* results in *WizzAir, BA* and *EuroLine*.	Applying *TR* and *RR* relationship on travel domain ontology *O* to find service instances matching to *WizzAir* results in *Japan Airline.*
2. The case library contains cases that store runtime behaviour of service instances I_{sub}. These cases are discovered and are marked as ball park cases.	Ballpark cases with the service instances are *CaseWizzAir*, *CaseBA* and *CaseEuroLine*	Ballpark cases with the service instances are *CaseJapanAirLine*
3. Apply the SCBR matchmaking algorithm on these ball park cases to find out Aggregate Degree of Match (ADoM) for each service instance.	describes result of applying matchmaking algorithm on the ball park cases.	describes result of applying matchmaking algorithm on the ball park cases.
4. The service instances $(i1_{sub}, i2_{sub}, \ldots\ldots, in_{sub}) \in I_{sub}$, with the highest ADoM are substitution service instances.	The service instance *WizzAir* with the case *CaseWizzAir* is substitution service instance.	The service instance *JapanAirline* with the case *CaseJapanAirline* is substitution service instance.

Table 17 Scenario-1

	Problem	**CaseEasyJet with highest ADoM** C_{cand}	**CaseWizzAir** **Values**	**DoM**	**CaseBA**		**CaseEuroLine**	
Destination City	Milan	Milan	Milan	1/1	Naples	0.5/1	Milan	(1/1)
Preference Domain	Hotel	NULL	NULL	0/1	NULL	0/1	NULL	0/1
Preference Domain	Airline	EasyJet	Airline(satisfied)	1/1	Airline(satisfied)	1/1	Coach(not satisfied)	0/1
Preference Currency	£	£	£(satisfied)	1/1	€ (not satisfied)	0/1	£ (satisfied)	(1/1)
Constraint Instance	EasyJet	not satisfied	BA(satisfied)	1/1	WA(satisfied)	1/1	EuroLine(satisfied)	0/1
ADoM Result		ADoM not acceptable	4/5 highest ADoM	0.8	2.5/5 second rank	0.5	2/5 third rank	0.4

Table 18 Scenario-2

	Problem	**CaseWizzAir with highest ADoM** C_{cand}	**CaseJapanAirLine** **Values**	**DoM**
Destination City	Tokyo	Tokyo	Tokyo	1/1
Preference Domain	Airline	Airline(satisfied)	Airline(satisfied)	1/1
Preference Domain	Hotel	Hilton Tokyo	NULL	1/1
Preference Currency	€	€	¥	0/1
Constraint Instance	WA	WA (not satisfied)	JapanAirline (satisfied)	1/1
ADoM Result		ADoM not acceptable	4/5 highest ADoM	0.8

The algorithm works for the first scenario where the case with *WizzAir* web service fulfils preferences and constraints from the problem request; however the algorithm fails for the scenario-2, where the algorithm will suggest replacing *JapanAirline* with the *WizzAir* for the travel request. This replacement with *JapanAirline* service leads to inconsistencies as the composed service in the candidate solution S_{cand} expects currency to be €, while the *JapanAirline* deals with currency ¥.

The fact that the variable *currency* depends on the variable airline *instance* (i.e. change in airline will affect the output currency) needs to be documented. This observation raises a new challenge of encoding variable dependency as the framework should use such dependency relationships to make sure that while adapting existing solution for the new problem request it does not violate any of the previously satisfied constraints. Therefore, some mechanism is necessary to maintain the integrity and consistency of the framework in order to prevent scenarios where contradicting constraint causes inconsistency as described in while applying knowledge base substitution.

As discussed earlier Web services composition process is an intelligent decision making process and the automation of which requires the reasoning about domain-specific knowledge at their disposal. Defining dependency relationship between domain variables represents such domain specific knowledge. The knowledge required is important from cognitive modelling perspective, as a step towards understanding how humans adapt cases when they reason from prior episodes. Such definition of variables and their dependency is termed in the framework as Domain Dependency Module (DDM).

The evaluation of the algorithm raises the following challenges:

1. How to define relationship between variables?

2. How to measure the impact of such related constraints and reflect that when the replacement occurs? (i.e. when changing the airline, how to reflect that on the fare currency?

3. At what stage in the matchmaking process do we use domain-dependency verification as it is fair to assume that existing solutions (prior to substitution) are consistent?

Defining variable dependency in XSCBR framework

An exhaustive surveyed of relevant literature was conducted to find methodologies that consider domain variables and dependency between variables. Such methodology shall also support constraints on domain variables in addition to the dependency constraints so that new system can fit with our existing framework.

One of the methodologies researched was *functional dependency* which has been part of the database technology for very long time. The functional dependency is dependency between database variables, for example, if relation *R* that has two variables *A* and *B;* then *B* functionally dependent on the variable *A* if and only if for each value of *A* there is no more than one value of *B* is associated.

Closure is an extensively used concept for the detection of such functional dependency in database technologies [94]. The main limitation of *Closure* with respect to the Domain Dependency Module is that it has no provision for considering other constraints apart from dependency, i.e., one of the constraints apart from dependency the SCBR framework requires is value constraints where variable can only have certain value from a restricted domain of values.

In the XSCBR framework, defining constraint between variables and depicting dependency on variables as part of these constraints is modelled as a Constraint Satisfaction Problem (CSP) [64]. Following section introduces CSP problem and provides the justification for it's use in the Domain Dependency Module.

Introduction to Constraint Satisfaction Problem

Constraint satisfaction problem is a powerful and extensively used AI paradigm. CSP involves finding values for variables subject to restrictions on which combinations of values are acceptable.

Formally speaking, CSP is defined by a set of variables $Z=\{X_1, X_2, \ldots, Xn\}$, and a set of constraints $C = \{C_1, C_2, \ldots., C_m\}$. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an

assignment of values to some or all of the variables, $\{X_i = v_i; X_j = v_j, \ldots\}$. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints [64].

CSP problem could be modelled as a graph called CSP graph. CSP graph is a representation of CSP where the vertices are variables of the problem and the edges are constraint between variables. Vertices are refereed as *nodes* and edges are called *arcs*.

A *Node* represents the domain variables, while *arc* represents the relationship between the variable nodes. A relationship could be of type dependent $\rightarrow$, independent $\Updownarrow$, incremental etc. The relationship could be formalized using different constraint operators.

A widely used example to show application of CSP is map colouring problem. A map colouring problem can be stated as: "Given a map with N regions bordering each other and M colours that can be used to colour each region. The problem is whether there is an assignment of one of the colours to each region such that two neighbours (regions that share at least one border) have the same colour." If we assume N=4, M=3 then,

Here, variables are Z = $\{w, x, y, z\}$

Domain for the variables are $D_w = D_x = D_y = D_z = \{r, g, b\}$

Constraint on the variables are $C = \{w<>x, w<>y, x<>y, x<>z, y<>z\}$

CSP graph for this problem is described in the following figure

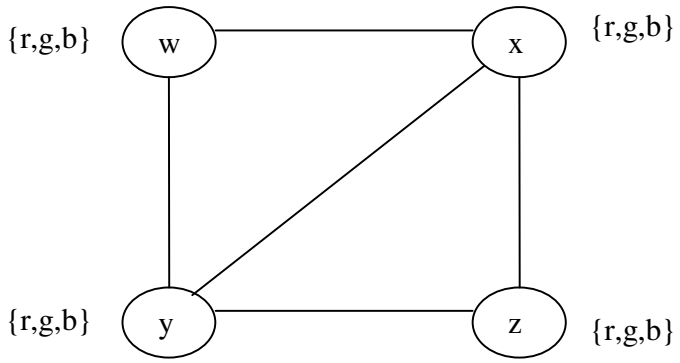


Figure 40 CSP graph for Map coloring problem

Commonly used techniques to solve CSP problems examine two types of consistency to solve problems [95].

a. Node Consistency
b. Arc Consistency

Node consistency ensures that every component or variable satisfies its domain constraint. Hence, for every variable X, values $x \in Domain[X]$, satisfy constraint on X. For example, region variable `w` in colouring problem should have values from red, green or blue.

To maintain Arc consistency: For every variable X, $x \in Domain[X]$ and for all variables Y, there needs to be a value $y \in Domain[Y]$, such that relationship C(X, Y) is satisfied by $\{X \leftarrow x, Y \leftarrow y\}$ and such value Y is called support for x. If X does not receive support from one of its neighbours then X is inconsistent. For example, in map colouring domain value of `y` has to be {`red` *or* `green` *or* `blue`} however `y<>w` meaning that `y` could not have same values as `w`, hence if `w` occupies `green` then `y could have` either `red` *or* `blue`. Arc consistency makes sure that related nodes are consistent. To find consistent solution for the problem the CSP graph has to be node and arc consistent making assignments consistent or legal.

Semantic description for Domain Dependency Module

Figure 41 illustrates a partial view of the DDM description for the travel domain where the dependency relationship between domain variables currency, solution, QoS and domain are described. The descriptions is limited to a binary CSP graph, where binary (two) variables are always directly related as this will be sufficient describing variables in XSCBR framework. For example, in the Figure 41, the directional-arrows in the graph describe dependency directions, for instance `currency` is dependent on `solution` variable and `solution` variable is dependent on the `domain` variable.

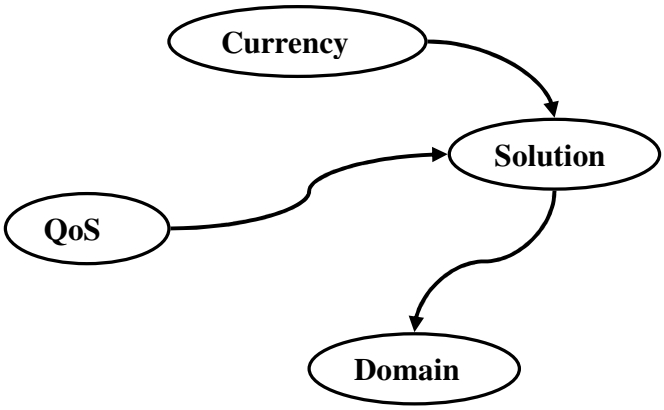


Figure 41 DDM for Travel Domain

When applied to web services composition problem where the framework requires a formal methodology to describe the constraint on the variables and a way to formalize the consistency criterion on variables, the definition of CSP fits as follows (travel domain exemplified):

Variables are = $\{w = currency,\ x = solution,\ y = domain,\ z = QoS\}$

The domain for the variables is $Dw = \{Any\ currency\ apart\ from\ P\}$

$D_x = \{Any\ solution\ apart\ from\ Q\}$

$D_y = \{Any\ travel\ domain\ apart\ from\ R\}$

$D_z = \{Any\ double\ but\ at\ least\ S\}$

Constraint on the variables are $C = \{w \rightarrow x,\ x \rightarrow y,\ z \rightarrow x\}$

Category of constraints

The type of constraints which are possible to be defined using CSP could be broadly defined in these following categories (Figure 42), which we call constraint behaviour.

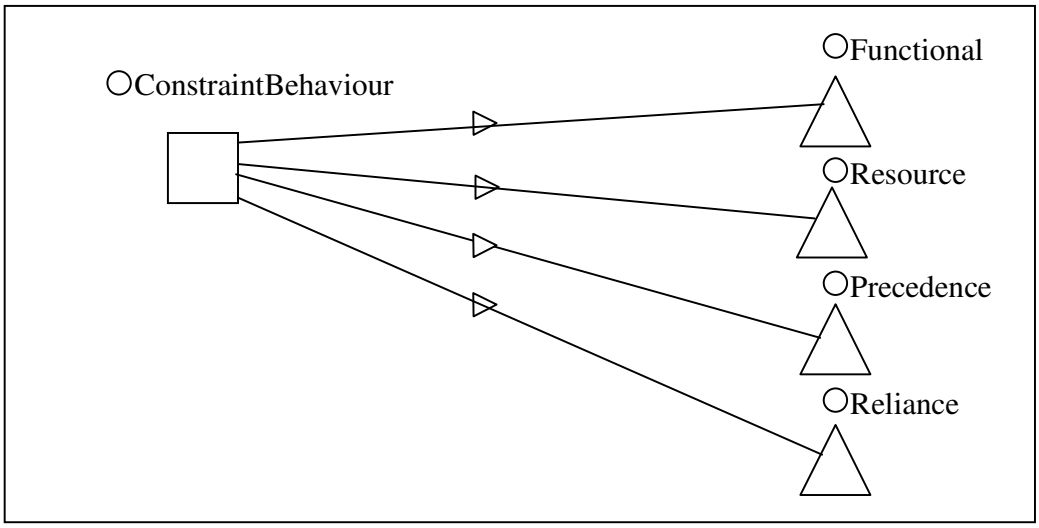


Figure 42 Constraint behaviour definition in DDM

The survey of the existing literature revealed that there are no semantic descriptions that provide ontology for describing CSP problem. Hence ontology was created for the CSP descriptions which covers *Functional*, *Resource*, *Reliance,* and *Precedence* behaviours applicable in various domains and also modelled this ontology in generic fashion making it possible to extend or reuse.

This research work primarily focused on the *Reliance* behaviour of CSP in order to address dependency relationship between domain variables and to explore this particular constraint behaviour in detail.

Reliance constraint behaviour

Arc constraints are *Reliance* constraints or are in terms of dependency relationship, when variables in the systems have relationship $X \rightarrow Y$, implying that if values of X changes then value of Y also changes.

This is particular to interest and is conceptualized in XSCBR framework. The DDM ontology was created that concentres on *Reliance* constraint behaviour (Figure 43) specific to Web services composition problem, while in DDM description we provide base for the other types of constraint behaviours to make it reusable for other technology domains.

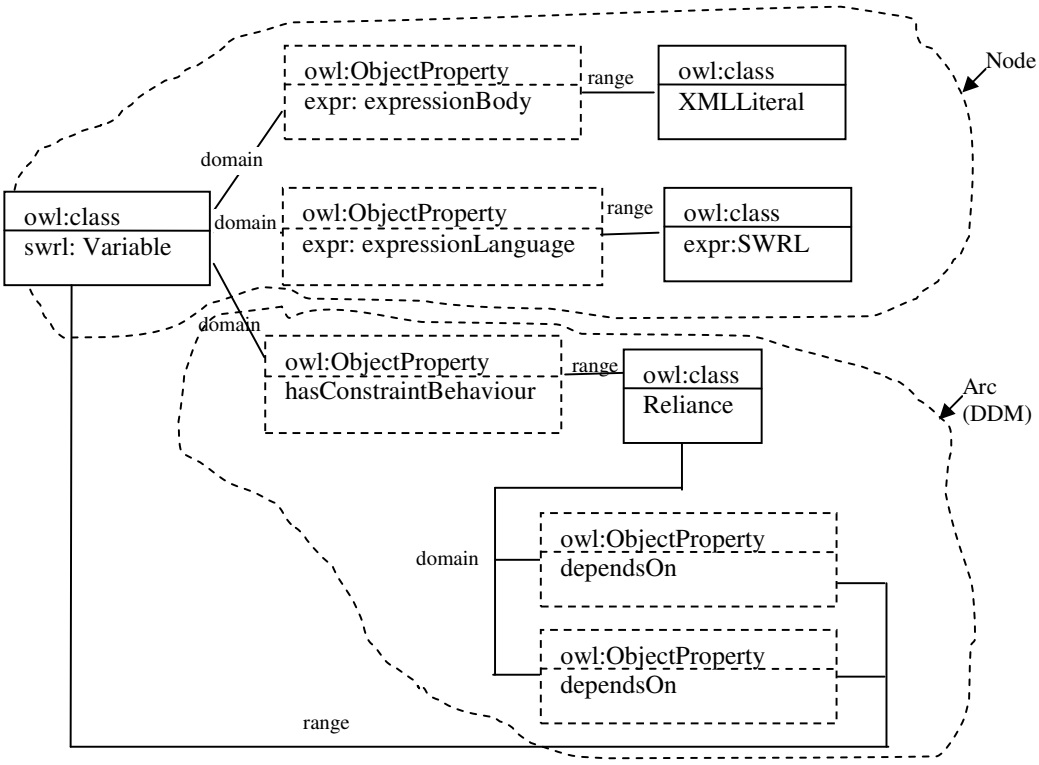


Figure 43 DDM Reliance behaviour

We already have defined variables as part of case representation and domain constraint on variable (node) as shown in the as node description is required description. DDM extends case representation to reflect the dependency between variables using directed arcs.

The semantic framework can deal with the DDM representation as follows:

Table 19 DDM Representation

Variable (Mandatory description)	Node (Required description)	Arc (DDM optional description)
Output Price	ConstraintOnPrice $parameterValue(ExpectedPrice, y) \wedge parameterValue(OutputPrice, x) \wedge lessthanorequal(x, ExpectedPrice, true) \Rightarrow parameterValue(ResultPrice, x)$	$price \rightarrow solution$ dependsOn (Price, Instance)
Expected QoS	ConstraintonQoS $ExecutionDuration(OutputQoS, y) \wedge ExecutionDuration(ExpectedQoS,1.5) \wedge lessthanorequal(y,ExpectedQoS,true) \Rightarrow ExecutionDuration(ResultQoS, y)$	$ExpectedQoS \rightarrow Solution$ dependsOn (ExpectedQoS, Solution)

DDM could be an optional module as DDM may not be required if the ADoM is acceptable. However when the search has to use knowledge substitution, absence of such DDM descriptions might compromise the consistency at the case representation, therefore we need to safe-guard using "semantic policing" policy.

Due to the aforementioned reasons, the framework considers DDM as an optional layer which may or may not be used or defined but useful to gear towards the adaptation stage. In absence of DDM the system could create invalidate results as shown this section. DDM is necessary to protect system against such inconsistency, however is defined on a different layer and granularity providing the possibility to switch DDM descriptions off. This is possible with our multi-stage semantic definition.

After formalizing domain dependency module to define and solve variable dependency using CSP, in the following section base *ApplyKBS* algorithm is extended to integrate DDM verification process that addresses the consistency problems associated with using adaptation for Web services composition.

ApplyKBS with DDM: DDM inspired Knowledge based substitution in XSCBR

Problem definition:

$P\ (R,\ C_{cand},\ S_{cand,}\ C)$, where C_{cand} is the candidate case with S_{cand} as solution that has the highest *ADoM* for request R but violates constraint cs that is based on variable v. Assuming S_{cand} is a solution composed of a finite set of service instances $S_{cand}\ (i_1, i_2 \ldots\ldots\ldots i_n)$ and C represents finite set of cases in case library $(c_1, c_2, \ldots c_e, \ldots . c_n)$, then solving problem P involves discovering set of service instances $(i1_{sub}, i2_{sub} \ldots\ldots\ldots in_{sub})$ that individually match the description of the instances in the candidate solution $S_{cand}\ (i_1, i_2 \ldots\ldots\ldots i_n)$. It is important to note that the substitution service instances $(i1_{sub}, i2_{sub} \ldots\ldots\ldots in_{sub})$ can originate from different solution sets.

To assist the reasoner, there exists an acyclic CSP graph G that depicts relationship between the domain variables V.

1. Retrieve initial state for the problem. This is achieved by creating tree from the graph G and finding root of the violated variable v. Store the variables in the path of variable v

including the root variable; let's assume that these variables are V_1. Retrieve variables which are dependent on the root variable using dependency relationships; let's assume these variables are V_2.

2. Start the process by varying value for the root variable. For web services composition this corresponds to finding service instances $(i1_{sub}, i2_{sub}\ldots\ldots\ldots in_{sub})$ using TR and RR relationships. For proceeding further retrieve cases which contain these service instances. These cases will be considered $ball\ park\ cases$ and will be scrutinized further.

3. Exclude any case from the $ball\ park\ cases$ which still violate constraint cs for the variable v.

4. Apply node and arc consistency on ball park cases for the variables set V_1, where node consistency will be measured against request R, and arc consistency will be measured against relationships in V_1. The qualified cases are termed $plausible\ cases$.

5. Apply node and arc consistency on plausible cases for the variables set V_2, where node consistency will be measured against execution values in S_{cand}, and arc consistency will be measured against relationships in V_2. The qualified cases are termed $resultant\ cases$.

6. Retrieve service instance $(i1_{sub}, i2_{sub}\ldots\ldots\ldots in_{sub})$ from the *resultant cases* and apply algorithm $modifyOWLS\ (S_{cand},\ (i1_{sub}, i2_{sub,}\ldots\ldots\ldots, in_{sub}),\ (i_1, i_{2,}\ldots\ldots\ldots, i_n))$ to modify $S_{cand.}$

Evaluating ApplyKBS with DDM Algorithm

Here the revised algorithm is evaluated on the scenario described in section 5.2.5. where the preliminary algorithm failed.

Scenario-2 = "Find a Trip for a traveller Mr Osman; Mr Osman wants to travel from Paris to Tokyo and also wants to reserve a hotel at Tokyo, he prefers to travel by air but wants to avoid travelling by WizzAir. He prefers to pay in EUR…. (Other requirements are snipped...)"

Scenario-3 = "Find a Trip for a traveller Mr Al-Dabass; Mr Al-Dabass wants to travel

from Paris to Tokyo and also wants to reserve a hotel at Tokyo, he prefers to travel by air but wants to avoid travelling by Easyjet. He prefers to pay in GBP…. (Other requirements are snipped...)"

The matchmaking algorithm discovers `CaseWizzAir` and `CaseEuroAir` respectively that represent candidate solution for the travel request in scenario-2 and scenario-3, albeit the exceptions of the travel medium instance – `WizzAir` (`WizzAir` is constrained in the scenario-2) and currency (as in the scenario-3 the preferred currency is GBP). Table 20 applies revised algorithm to evaluate the possibility of adapting `CaseWizzAir` and `CaseEuroAir` for respective travel requests.

Table 20 Evaluating ApplyKBS with DDM Algorithm

Algorithm steps	Applying to scenario – 2 (Table 21)	Applying to scenario – 3 (Table 22)
1. Retrieve initial state for the problem. This is achieved by creating tree from the graph *G* and finding root of the violated variable *v*. Store the variables in the path of variable *v* including the root variable; let's assume that these variables are V_1. Retrieve variables which are dependent on the root variable using dependency relationships; let's assume these variables are V_2.	Violating variable `v = solution instance` The root of variable v in the travel domain CSP graph (Figure 44) is variable `Domain` V_1 *(variables in the path of instance)* = `{domain}` V_2 (variables dependent on root variables excluding variables covered by V_1) = {$QoS_{duration}$, $QoS_{reputation}$, `city, currency }`	Violating variable `v = currency` In CSP graph for the travel domain in Figure 44, tracing solution dependency will result in root variable `Domain` V_1 *(variables in the path of currency to domain)* = `{instance, domain}` V_2 (variables dependent on root variables – variables covered by V_1) = {$QoS_{duration}$, $QoS_{reputation}$, `city, }`
2. Start the process by varying value for the root variable. For web services composition this corresponds to finding service instances *(i1$_{sub}$,i2$_{sub}$…….in$_{sub}$)* using TR and RR relationships. For proceeding further retrieve cases which contains these service instances. These cases will be considered *ball park cases* and will be scrutinized further.	Varying values for the root variable (domain) and retrieve cases with services instances related to this domain. Ballpark cases are `CaseJapanAirLine ,CaseWizzAir2, CaseBA` and `CaseEuroLine` *(for demonstration few instances are chosen)*	Varying values for the root variable (domain) and retrieve cases with services instances related to this domain. Ballpark cases are `CaseJapanAirLine ,CaseWizzAir, CaseBA` and `CaseEuroLine` *(for demonstration few instances are chosen)*
3. Exclude any case from the *ball park* cases which still violate constraint *cs* for the variable *v*.	Check if cases `CaseJapanAirLine ,CaseWizzAir2, CaseBA` and `CaseEuroLine` violates constraint `cs` (instance must not be `WizzAir`). `CaseWizzAir2` has value for the instance variable= `WizzAir`, which still violates constraint on instance, hence will be excluded from the next step. Rest of the cases qualifies, hence the Ballpark cases = { `CaseJapanAirLine, CaseBA, CaseEuroLine` }	Check if cases `CaseJapanAirLine ,CaseWizzAir, CaseBA` and `CaseEuroLine` still violates constraint `cs` (currency must be `GBP`). `CaseJapanAirLine` has value for the currency variable= Yen, which still violates constraint on currency, hence will be excluded from the next step. Rest of the cases qualifies. Ballpark cases = {`CaseWizzAir`, `CaseBA`, `CaseEuroLine` }
4. Apply node and arc consistency on ball park cases for the variables set V_1, where node consistency will be measured against request *R*, and arc consistency will be measured against	V_1 = `{domain}` Hence, verify ball park cases to make sure that the constraints on `domain` (must be `Airline`) are not violated.	V_1 = `{instance, domain}` Hence, verify ball park cases to make sure that the constraints on `instance` (instance must not be `EasyJet`) and `domain` (must be `Airline`) are

relationships in V_1. The qualified cases are termed *plausible cases*.	*CaseEuroLine* has value for the domain variable= Coach, which violates constraint on domain, hence will be excluded from the next step. Rest of the cases qualifies. Plausible cases = { *CaseJapanAirLine*, *CaseBA* }	not violated. *CaseEuroLine* has value for the domain variable= Coach, which violates constraint on domain, hence will be excluded from the next step. Rest of the cases qualifies. Plausible cases = {*CaseWizzAir*, *CaseBA* }
5. Apply node and arc consistency on plausible cases for the variables set V_2, where node consistency will be measured against execution values in S_{cand}, and arc consistency will be measured against relationships in V_2. The qualified cases are termed *resultant cases*.	V_2 = {currency, $QoS_{duration}$, $QoS_{reputation}$, city } *CaseJapanAirLine* has value for the currency variable= Yen, which violates constraint on currency, hence is invalidated. *CaseBA* satisfies $QoS_{reputation}$ $QoS_{duration}$, and *currency* Hence, Substitution service will be *BA* with the case {*CaseBA*}	V_2 (variables dependent on root variables – variables covered by V_1) = {$QoS_{duration}$, $QoS_{reputation}$, city } *CaseBA* satisfies $QoS_{reputation}$ but violates $QoS_{duration}$ While *CaseWizzAir* satisfies $QoS_{reputation}$ and $QoS_{duration}$ Hence, Substitution service will be *WizzAir* with the case { *CaseWizzAir* }

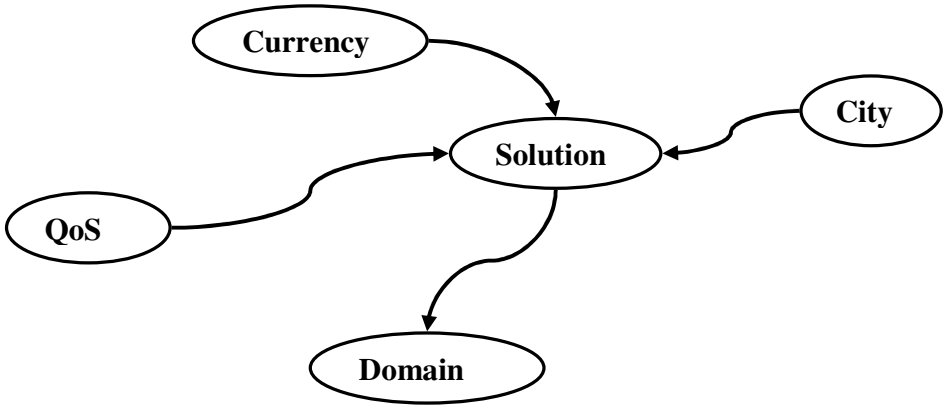


Figure 44 CSP graph for travel domain case study

Table 21 Scenario-2 (Revisited)

	Problem	**CaseWizzAir with highest ADoM** C_{cand}	**CaseJapanAirline**	**CaseWizzAir2**	**CaseBA**	**CaseEuroLine**
Constraint Instance *Steps: 3*	WizzAir	WizzAir	JapanAirline √	WizzAir ×	BA √	EuroLine √
Preference Domain *Steps: 4*	Airline	Airline	Airline √	Airline	Airline √	Coach ×
Preference Currency *Steps: 5*	€	€	¥ ×	€	€ √	€
QoS execution duration *Steps: 5*	<0.5	0.4	0.5	0.4	0.5 √	0.5
QoS reputation *Steps: 5*	At least GradeA	GradeA	GradeA	GradeA	GradeA √	GradeA
Result		Rejected by user/ADoM not acceptable	Disqualified at step 5	Disqualified at step 3	Passes all stages, selected as solution	Disqualified at step 4

Table 22 Scenario-3

	Problem	**CaseEuroAir highest ADoM** C_{cand}	**CaseJapanAirline**	**CaseWizzAir**	**CaseBA**	**CaseEuroLine**
Preference Currency *Steps: 3*	£	€	¥ ×	£ √	£ √	£ √
Preference Domain *Steps: 4*	Airline	Airline	Airline	Airline √	Airline √	Coach ×
Constraint Instance *Steps: 4*	EasyJet	EuroAir	JapanAirline	WizzAir √	BA √	EuroLine √
QoS execution duration *Steps: 5*	<0.5	0.4	0.5	0.4 √	0.5 ×	0.5
QoS reputation *Steps: 5*	At least GradeA	GradeA	GradeA	GradeA √	GradeA	GradeA
Result		Rejected by user/ADoM not acceptable	Disqualified at step 3	Passes all stages, selected as solution	Disqualified at step 5	Disqualified at step 4

5.2.6. Planning based transformation in XSCBR framework

The planning based transformations is applicable when the available solutions can not fulfil the problem requirements with normal matchmaking and discovery mechanism or by applying minor modifications using *ApplyKBS with DDM* algorithm does not produce any outcome. In this scenario, we can reuse existing planning technique and existing planners to form a plan for composition from a scratch. Following is the definition of a planning problem.

AI planning [43] problem can be described as a five-tuple problem (S, $s0$, G, A, Γ) where,

S is the set of all possible states of the world,

$s0$ denotes the initial state of the planner,

G denotes the set of goal states the planning system should attempt to reach,

A is the set of (ground) actions the planner can perform in attempting to reach a goal state, and the transition relation

Γ defines the semantics of each action by describing the state (or set of possible states if the operation is non-deterministic) that results when a particular action is executed in a given world state.

The creation of an AI planner and research on using planning for Web services composition is out of scope for this research and we rely on the available planning methodologies for Web services composition to form composition schemas. If XSCBR matchmaking algorithm and *ApplyKBS with DDM* fails to find any solution then, the planner is employed to do transformations where the planner generates composition schemas from existing service descriptions, and XSCBR execution engine allows executing these descriptions, takes feedback and stores as a case, which is analyzed for the future problems.

5.3. Conclusions

In this chapter an aspect of CBR, case adaptation was explored to extend Web services discovery and matchmaking framework for Web services composition.

The extension for case adaptation requires standardizing the case representation format to make case representation applicable to any application domains. The proposed generic case representation has provision for the facilitations to service participants and

especially service requestor as it was outlined, extension of OWL-S descriptions to support service requestor in terms of specifying constraint and preference on search. The rules were introduced to handle complex constraint and preferences relationships in case descriptions. The Quality of Service (QoS) parameters were also considered as an important selection criterion for web services discovery and to modify (QoS) descriptions from the previous framework inline with the modifications and improvements in the case representation, especially using rules.

In this chapter the solution was also represented semantically in order to adapt the solution for new problems when required. This is made possible using existing OWL-S process model for modelling composition schemas required for adaptation module. The richness of the workflow based patterns supported by OWL-S process model and provision of semantics in the specification itself were the reasons selecting OWL-S. The *ModifyOWL-S* algorithm was outlined to demonstrate how OWL-S process file could be modified with the help of API to insert or remove service reference to satisfy new problem requirements.

In the XSCBR framework, case adaptation was applied to solve a new problem by merging the local solutions from previously solved problems and creating a globally consistent solution for the new problem. In-depth investigation of the CBR literature concluded that there are two major challenges in CBR for achieving case adaptation in this manner: first the development of an efficient methodology for case adaptation and second, maintaining consistency of solutions and knowledge supplement to assist case adaptation. A methodology was designed based on the Constraint Satisfaction Problem (CSP) to address these challenges and handle the inconsistency problem with a CSP inspired Domain Dependency Module (DDM). This approach allows defining dependency between variables of the domain and ensuring the consistency by maintaining dependency constraints between these variables whenever a solution is adapted.

In addition to maintaining the consistency of solution, in the XSCBR framework, we also advocate a knowledge-intensive approach to automate the process of adaptation in CBR inspired Web services discovery and composition problem. The argument is that Web services composition is a developer-intrusive problem solving method, the automation of which requires the reasoning about domain-specific knowledge at their

disposal. This chapter also discussed the methodologies to define and utilize such knowledge.

A case study based evaluation of the algorithms is carried out to validate the framework.

6 Chapter Six: Implementation and Evaluation of XSCBR framework for Web Services Discovery and Composition

The previous chapter overviewed the utilization of a semantic case based reasoner XSCBR for automated Web services discovery and composition. The argument is in favour of highlighting the importance of considering the execution values for semantically-described non-functional Web services' parameters in decision making regarding Web service adequacy for particular task. A novel, semi-transparent framework was proposed that captures Web service execution experiences as cases, which can be subsequently orchestrated to present solutions to the new problems. The XSCBR framework extensively uses ontologies, as semantics are used both for describing the problem parameters and for implementing components of the CBR system: representation, indexing, storage, matching and retrieval.

In this chapter, the details of XSCBR framework implementation for solving the automated Web services discovery and composition problem is presented. This is followed by an evaluation of the precision and recall of the Web services discovery mechanism in the XSCBR framework. Investigation of the incurred impact of such mechanism on the performance of the framework is also discussed.

6.1. Choice of Tools and Specification for Implementation

The implementation of the framework relies extensively on semantics to implement the component of the CBR system namely indexing, matching, retrieval and ranking. These components are developed using Web Ontology Language (OWL) ontologies and java based ontology reasoner Pellet.

6.1.1. Web Ontology Language (OWL) and Pellet Reasoner

Description Logics (DL) [96] is the name for a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain (the "world") by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description). A distinguished feature of DL is the emphasis on reasoning as a central service that allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base. Based on these principles of DL, the semantic web community have developed Web Ontology Language (OWL) [22] to encode the knowledge required in the semantic web mission of making the WWW machine interpretable.

To address different levels of requirements for expressivity and reasoning in Semantic Web, OWL specification offers three different dialects: OWL- Full, OWL-DL and OWL-Lite. They are ordered based on the expressiveness these dialects provide: OWL-Full provides maximum expressivity while the OWL- Lite provides the least. The OWL-DL dialect is used for XSCBR implementation, as OWL-DL imposes a number of restrictions on RDF graphs, some of which are substantial (for example, the set of class names and individual names be disjoint) and some less so (that every item have a *rdf:type* triple) in order to achieve completeness and inference.

For implementing the components of XSCBR framework, Pellet reasoner is utilized that works on top of OWL-DL ontologies. Pellet [71] is a sound and complete OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined data types, and debugging of ontologies. The support for cardinality in describing cases with non-functional parameters is essential for frameworks such as XSCBR that allows granular service requests involving non-functional parameters. Pellet has also proven to be a reliable tool for working with OWL-DL ontologies and experimenting with OWL extensions [71]. Apart from these features, open source access to Pellet has been one of the main criteria for choosing Pellet for this research project. Bossam [97], a forward chaining rule engine was used to reason SWRL based constraint and preferences conditions.

6.1.2. OWL-S: Specification and API

In the CBR-based approach to automatic Web services discovery and composition, when the existing solutions are not sufficient to solve the service request, case adaptation process is utilized to modify an existing solution to adapt to the new request. In the XSCBR's adaptation algorithm, the decision to select the components of an existing solution which needs to be adapted is based on a variety of descriptions and situational parameters, i.e., functional and non-functional parameters. Once the components are identified, necessary adaptation changes are made to the solution of an existing case, which in XSCBR framework is represented by OWL-S process file which is an OWL based Web services description. The task of amending OWL based solution will be consistent with the policy of using semantics for automation as compared to using XML based BPEL composition schemas.

The OWL-S API [88] is used to support reading, modification and execution of the OWL-S process models. The API is instrumental in execution of services when the user selection is made, and also utilized for the implementation of *ModifyOWL-S* where OWL-S process file is modified with the help of API to insert or remove service reference to satisfy new problem requirements.

The following section provides detail on how various components of the XSCBR framework are implemented using the tools and specifications explained in this section.

6.2. XSCBR Framework Implementation

A context diagram of the XSCBR framework is given in the Figure 45, outlining the functionality of the framework. XSCBR allows service participants to perform their publishing, composition and discovery tasks in transparent manner, where the framework components work as a black box and dynamically match service requests with published service definitions.

The CBR controller module is the first point of entry for the framework users and provides matchmaking and ranking of existing services to service request and also performs lightweight knowledge-based substitutions of service descriptions if the resultant solution is unsatisfactory.

The indexer module is responsible for assisting controller in effective discovery of Web services using indices to index cases in the case library. The adaptation engine module

performs substantial, necessary substitutions of service solution components if the controller fails in finding satisfactory results and generates new executable composition schemes using case adaptation process.

The execution engine allows enacting existing or newly generated composition schemes. The case library and knowledge base store assist the framework in finding relevant results by supplying knowledge stored in terms of cases and heuristic rules. The error reporting and recovery unit stores any errors and exceptions occurring during the framework operations into case library to avoid repetition.

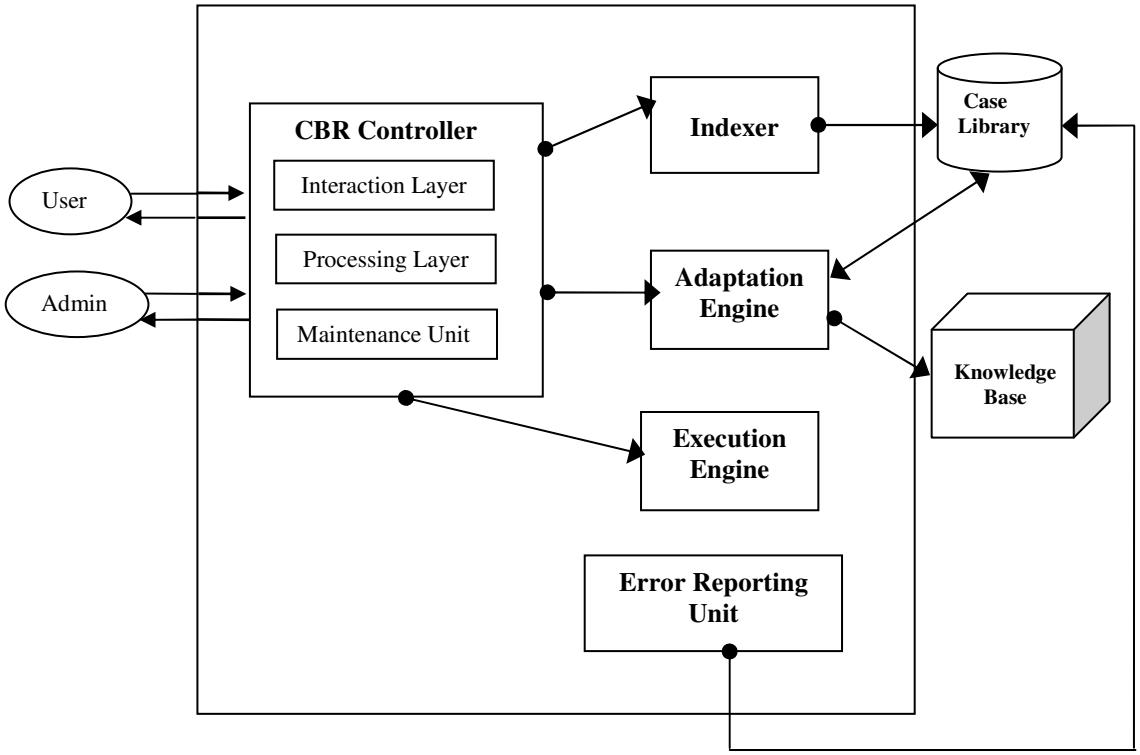


Figure 45 XSCBR framework modules

The following sub-sections provide more detail on the framework components.

6.2.1. CBR Controller

The CBR controller is the main component of the architecture, which processes user requests for solving a discovery problem and handles admin requests for framework maintenance. The component has been divided into three units: interaction layer, processing layer and the maintenance unit. The multi-component architecture is intended to improve the system performance.

Interaction Layer

The interaction layer is the first point of entry for the users of the framework. This module contains `createQueryInstance` method that converts user requests to an OWL request and pass the query instance to the processing layer module for further processing. The interaction module considers the domain of case representation selected by the user and instantiates the request as an instance of `CaseRepresentation` accordingly. For example, if the user is looking for finding a sensor service and selects `sensorCaseRepresentation1` for outlining their request among the other sensor case representations available, then the interaction layer takes the textual data from the user interface and creates an instance (query instance) of `sensorCaseRepresentation1`, which will be the semantic equivalent of the provided user textual request.

Processing layer

The processing layer contains the implementation of the ranking algorithm and light-weight knowledge-based substitution algorithm which operates at the description level. The functionality is implemented by two modules: the first is `SemanticComparision`, which contains methods to handle matchmaking and ranking of requests with the available service executions. This module is an implementation of matchmaker that semantically compares functional and non-functional parameters in the user request with the available cases and ranks them based on the degree of match. `SemanticComparision` interacts with the second module in the processing layer: `KnowledgeSubstitution` in situations where the service descriptions are not matching hence requires finding possible bridge-rules that allow the semantic discrepancies to be consiled. `KnowledgeSubstitution` takes possible discrepancies and applies ontology and SWRL reasoning on the knowledge base to solve the differences.

Maintenance module

The framework also includes `administration` module for various bookkeeping activities i.e., entering new cases, removal of existing cases, extending case representation and setting up domain specific acceptable degree of match.

6.2.2. Indexer

The indexer module implements the *"partitioning_case_library"* method, where the case library is partitioned based on certain vocabularies and the new problem is recognized based on the identical vocabularies to decide which partition the problem falls into. The process of searching entire case library is computationally expensive and indexing cases and searching cases based on indices allows frameworks to efficiently find a solution as the indexing process effectively reduces number of cases to be investigated. For example, case containing *EasyJet* as a solution can be indexed as the *AirLine* domain case, hence falls under *Airline* partition. The framework trying to solve any problem request that specify *Airline* as a domain preference will retrieve this case as a potential solution. The processing layer in this fashion initiates the *indexer* module using the information in the user request based on constraint or preference of domain, hence portioning case library based on the domain of cases.

6.2.3. Adaptation Engine

The adaptation engine contains logic for deciding the applicability of the adaptation process by comparing the aggregate degree of match for the best ranked result against the acceptable ADoM (Aggregate Degree of Match) set by the administrator and also in response to the intervention of the user, when the user deems the returned results unsatisfactory.

The adaptation engine adapts cases using knowledge based substitution. The engine applies substitution using available knowledge and modifies solution component of existing case to adapt to a new problem request. The module contains two classes to achieve this: *AdaptSolution* and *CreateNewProcess* which are implementation of the algorithms *ApplyKBS with DDM* and *ModifyOWL-S* respectively. The *AdaptSolution* class contains methods that consider a case with the highest degree of match but with the unsatisfied constraints to modify the solution of this case and generate a satisfactory solution that is applicable to the current problem request. The mechanism uses Domain Dependency Module (DDM) tree to scrutinize singleton service cases (cases that have just one service as *serviceIsPartOfSolution*) by first narrowing down them to ball park cases (cases that satisfy part of DDM constraints) to the resultant cases (cases that satisfy all of the DDM constraints). The resultant case (*caseR*) along with the case that has the highest ADoM (*caseE*) is input to the

CreateNewProcess class, which finds the matching components (*perform*, *process*, *result*, *input* and output bindings) from the existing case *caseE* and replaces them with relevant components in case *caseR*. The final output of the adaptation module is an executable composition scheme.

6.2.4. Execution Engine

The framework uses execution engine provided by the OWL-S API [88] that allows executing WSDL-grounded OWL-S descriptions. The *Input* and *Output* in *CaseDiscovery* section of *CaseRepresentation* class has similar semantics to OWL-S process model functional parameters (Input and Output), hence we are able to exploit the compatibility.

6.2.5. Knowledge sources: Knowledgebase and Case Library

The file system was utilized to store cases in case library, where OWL files of each case along with domain ontologies and rules presenting heuristics are stored on a web server. The implementation supports SWRL and OWL to represent such knowledge. The knowledge contributed to the framework comes from two sources: administrator and service providers. The administrator can input, edit or delete cases from the case library and also can input the rules that will contribute to the functioning of the system. The service providers can submit any knowledge in terms of ontology or rules necessary to make their services part of composition.

6.2.6. Error reporting unit

The error reporting unit contains logic for error reporting. Errors occurring during the functioning of the system are stored as new cases to avoid future failures. For example, using the error reporting module, the administrator can log a new case for a service that is no longer available at the specified access point and can delete such case when notified as a permanent error.

6.3. Graphical User Interface

The GUI of the XSCBR framework is developed using Java programming language. It includes a portal for each of the framework users: framework administrator (service composer), service requestor and service provider (Figure 46). The administrator can use the interface to set up acceptable ADoM for various domains, to add/remove cases

from the case library, update the knowledge base, or perform other maintenance duties such as acting on errors and exceptions reported in the framework. The options available to the service providers are to select the domain of the service they intend to publish their service under, in addition to aid in describing their service in OWL-S.

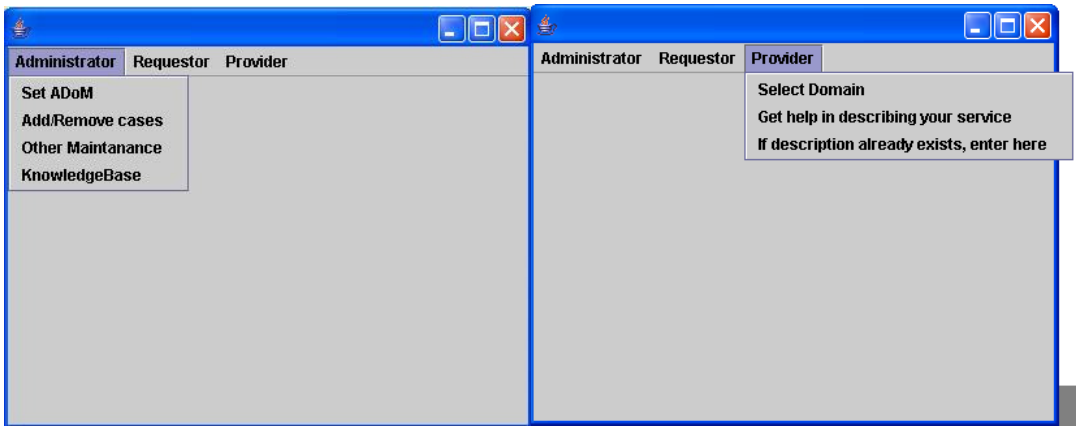


Figure 46 Graphical Interface for the Administrator and Provider

For case searching, the framework assists the case requestor in formulating service queries with a user interface similar to that available for case administrator, and transparently creates semantic description for the new problem parameters (Figure 47). The generated index for such semantically described problem governs the decision regarding which partition the problem falls into. The cases from that partition are retrieved for further matching. The result of the matching procedure displays the case instances, which have similar problem situation to the new problem.

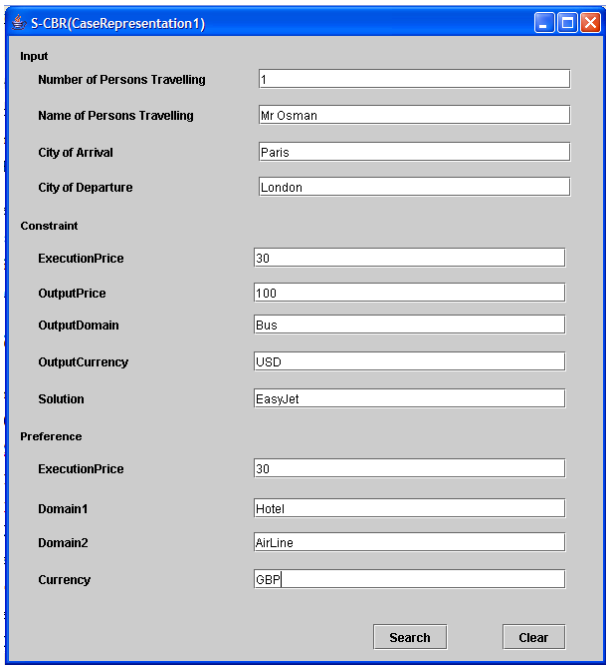


Figure 47 GUI for Web services requestor

The framework also displays the aggregate matching coefficient associated with such suggested case instances for the *case requestor* to view and make appropriate selection.

The requestor can analyze the cases using GUI (Figure 48) and can execute the service if they are satisfied with the results or they can point to a service execution which they will prefer to be adapted.

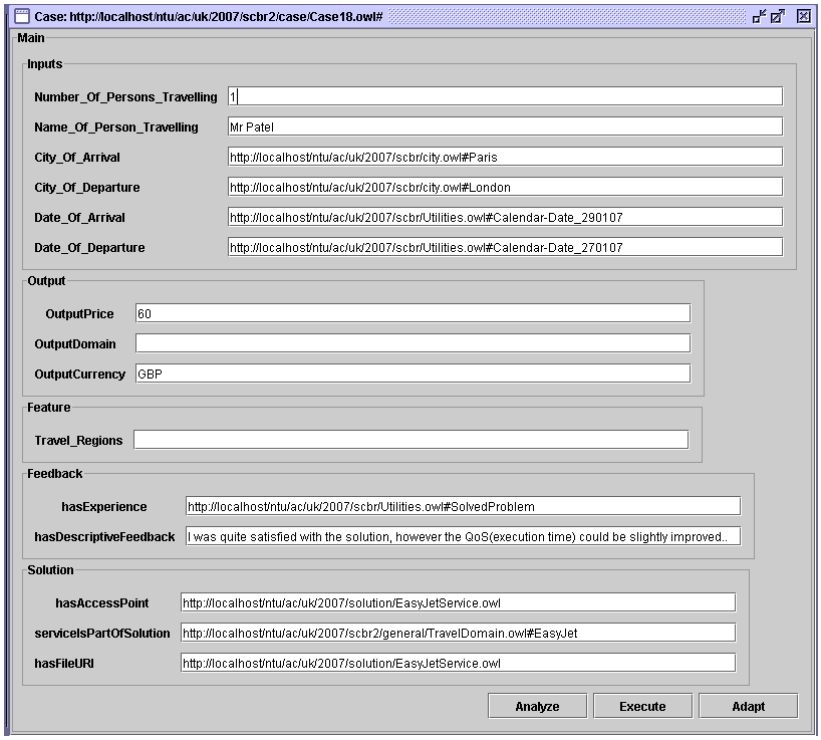


Figure 48 Case analyzed by requestor

The next section provides performance study results of the XSCBR framework when measured for the effectiveness and efficiency.

6.4. Evaluation

A group of experiments were carried out to evaluate the impact of XSCBR on the quality of precision and recall of automatically composed and the incurred overhead on the performance of the composed application.

6.4.1. Objectives

The evaluation of the framework is categorized into qualitative and quantitative. The qualitative evaluation answers the research questions as outlined in the motivation section of chapter 1 and contrasts them to what has been achieved in this research effort.

For the quantitative evaluations the majority of the traditional approaches on automated Web services discovery and composition focus only on the performance evaluation (efficiency) of the system in terms of execution time, i.e., the average response time the search engine takes to find a suitable service. In contrast, the argument is that the composition engines should be also measured for effectiveness in terms of how closely and accurately they match user requirements. Similar to evaluations of information retrieval engine [18] the proposal is that the qualitative evaluations of service-retrieval engine should be based on the precision and recall performance of web service discovery engine for a range of queries. Inline with this conclusion, evaluation of XSCBR framework on precision, recall along with execution time and performance is performed.

6.4.2. Qualitative evaluation of the XSCBR framework

The main research question of "building dynamic web services composition framework in semantic web" has been answered, in general, by the design and implementation of a semantic web and case based reasoning based system for automated Web Service retrieval and composition.

Research Questions

Following are the answers to the underlying and more specific research questions (RQi):

RQ1: Web services composition is mainly a task performed by human developer, how can this task be automated using software programs?

At the beginning of this research it was established that to imitate human reasoning in service composition task first and foremost it is necessary to arm software programs with intelligence to identify the capability of Web services.

Further research lead to investigating semantic web based Web service descriptions as such descriptions provide a mechanism to describe Web services capability. However, to interpret these semantic descriptions and in order to compose and execute Web services for achieving the desired functionality, it requires an intelligent layer that can replace human developer. In other words, the intelligent layer should comprehend the descriptions in order to accurately decide the possible services and build flow management for those services.

RQ2: Workflow-based techniques are a popular and widely adopted option for application integration/Web services composition. Can semantic technologies inject the required intelligence to aid the workflow techniques in achieving more dynamic, perhaps automated service composition?

The BPEL specification solves the immediate problems industry is facing regarding the use of Web services for enterprise application integration. However, in its present form, the specification overlooks the possibility of binding the service participants and performing flow management on the fly, hence only specifies how the service composer can perform both activities manually.

As demonstrated with the DPDWS tool in the chapter 3, enriching BPEL specification with semantics achieved automatic selection of the Web services with pre-agreed interfaces. The hybrid approach presents a practical solution to a current problem. However, the approach could only achieve limited automation to the composition process as the process model on top of WSDL, which is an XML grammar. Using XML one cannot define concepts or relations between concepts, which is essential to convey our understanding of real-world domains to the intelligent reasoners performing the automation. The issue related to the current discussion is the use of non-semantic grammar for the composition specification. For the composition engine to provide automatic discovery and flow management, the process model needs to have the consideration of the semantics in the specification. The addition of semantics within an XML centric standard like BPEL will not achieve the automation as the composition engine apart from being executable similarly to BPEL, also needs an intelligent reasoner which can interpret the semantic description.

RQ3: Investigation of problem solving methodologies that represents a viable approach for solving the problem of automatic Web services composition problem.

An exhaustive investigation was performed for finding a methodology that can utilize service descriptions to achieve greater level of accuracy in web services discovery and matchmaking compared to the existing approaches that rely on planning, agents, software synthesis and workflow. This investigation led to consider the importance of the execution values for semantic non-functional Web services parameters in decision making regarding Web service adequacy for the task. The Case Based Reasoning (CBR) was established as a methodology that supports such specification and an XSCBR

framework was developed to achieve dynamic Web services discovery and composition mechanism.

The main benefit of using CBR for this research is the consideration of experience-based knowledge for judging capability of Web services for the discovery phase. The adaptation mechanism in CBR allowed the framework to model the problem of service composition while considering role of knowledge in addressing discrepancies.

RQ4: Selecting the appropriate implementation technology from the abundance of standards available.

The Web Ontology Language (OWL) was utilized for constructing ontologies in the XSCBR framework. OWL is the most expressive Semantic Web knowledge representation so far and the layered approach adopted by semantic web, allows reasoning and inference based on ontologies, which is the most powerful and ubiquitous feature of Semantic Web

Similarly, the vital component of XSCBR framework, case representation is an extension of OWL-S descriptions and adds support for `Preference` and `Constraint` components to assist the service requestors in providing granular level of request descriptions. It was also decided to utilize existing OWL-S process model for modelling adaptation in the framework. The richness of workflow based patterns supported by OWL-S process model and provision of semantics in the specification itself were the motivating factors for selection.

RQ5: The main thesis of this research work is based on the theory that assistance with the facilitation of the composition process to the service participants (service requestor, provider and the composers) plays a major role in encouraging the adoption of the Web services technology. This research shall address question of the facilitation by providing assistance to the service participants in their respective tasks in the composition process.

- We have addressed the issue of facilitation to the participants by automating the steps the participants have to perform in their specific role in the composition process.
  - For the service requestor, the framework delivers a user-transparent search engine that exhibits high precision and recall of results without requiring intervention

from their part, which is the first facilitation offered by XSCBR. The existing approaches for Web services discovery and composition has no standard representation available for requestor to request their selection. We fulfil this requirement by extending OWL-S specification for requestor descriptions with *Constraint* and *Preference* components of *CaseRepresentation* to specify their search. The extension allows requestors to describe requests at a granular level and allows providing more details on what they are looking for.

An additional facilitation to the service requestor is that the requestor can work independently from the composer. For example, the framework allows service requestor to work independently from the composer by translating text-based request to ontology based case, allowing the description of the expected service functionality in an unambiguous form.

- Using XSCBR, the service developers have the opportunity to reuse their services to be part of a composition and can provide knowledge which bridges the gap between their intentions in describing services and the standard service interfaces they refer to. This facility serves real world scenarios as service offer, request descriptions and composer descriptions are ideally designed independently where service providers describes their offers, clients query services using a semantic request and a matchmaker finding offers that match the request. XSCBR addresses these concerns with generic case representation that caters for the circumstances where the services with different descriptions from the composers can exist.

  The service providers work independently as the framework assists provider in subscribing services or mapping their service descriptions to existing case representation formats but expects the provider to supply knowledge for any mismatches in the Web services description to the existing representation. The composer can assist this process by acting as a domain expert and either creating or reusing heuristics to address mismatches. From this point onwards, the framework takes care of matching service requests to service offerings and also provides rankings indicating relevance of the match to the service request.

- In the XSCBR framework, the role of composer - who analyzes service request and matches against service offers and if required combines number of offers to meet the request, is transformed to a domain expert or domain administrator. In this role, the composer is responsible for maintaining XSCBR framework since

the framework alleviates the burden of verifying the service capability and interpreting requests and the composer is only involved in the integration of services when the framework is initiated hence does not contain any cases and at the time when the framework returns no results in response to a service request.

- A facilitation which is applicable to all participants is that the framework is domain-independent and applicable to heterogeneous domains. This is achieved using generic case representation schema which is applicable to heterogeneous application domains services.

- With the evidence of effective and efficient implementation of framework, we provide support for the argument of using semantics in achieving automation for Web services discovery and composition [98].

RQ6: How does the functionality offered by the XSCBR framework compare to that offered by other Web services composition frameworks?

The criteria for comparing the framework functionality are:

1. *Expressiveness* measuring how expressive the framework is in terms of representation provided for service request and composition workflow patterns.

2. *Transparency* of the framework is determined in terms of how seamless it is to utilize the framework for Web services composition with priority given to the service consumers. It is possible to measure transparency of a framework based on the level of automation achieved by mechanizing the process of service discovery, composition and execution.

3. *Adaptability* analyzes frameworks from the perspective of finding out if the framework can adapt to change, i.e. how particular framework deals with situations such as when a service is no longer active in the composition or services with various service descriptions exist?

Using the aforementioned criteria, the XSCBR framework is compared against prominent frameworks that are based on DAML-S, UDDI and the frameworks that extend and utilize workflow based specification such as BPEL.

Under the category of DAML-S based frameworks we consider the works of the authors Wu, D., *et al.* in [42] and Richards, D., *et al.* in [56]. Wu, D., *et al.* in their framework

[42] applies SHOP2 planner that utilizes IOPE based DAML- S profile and process model to achieve Web services composition. In their framework advocated by Richards, D. *et al.* applies and extends Agent Factory- an automated facility for composing software agents, to use Web services as agent components. Their framework use the DAML-S profile models to provide descriptions of the components at the conceptual level for the discovery and the grounding model to provide the descriptions at the implementation level for the integration.

UDDI based framework in [59] by Limthanmaphon, B. *et al.* utilizes Case Based Reasoning (CBR) for Web services composition. In their BPEL based framework in [67] by Mandell, D. *et al.* proposes a bottom-up approach for Web services interoperation in BPEL4WS and use OWL-S based descriptions for runtime binding of service partners.

Expressiveness

The framework presented by Wu, D., *et al.* relies on DAML-S profile for service descriptions hence supports functional parameters such as IOPE; however in their framework authors do not specify provision for non-functional parameters such as Quality of Service(QoS). The framework also has no scope to consider `Preference` and `Constraint` parameters to help service requestor in describing their requests at a granular level.

The framework advocated by Richards, D. *et al.* utilizes DAML-S profile and process model coordination patterns to model agents as DAML-S components although it does not consider non-functional parameters for selection criteria and also ignores granular service request expressiveness in their implementation.

The framework by Limthanmaphon, B. *et al.* relies on keyword search on service names or `tModels` for Web services discovery. The framework supports `Constraints` and `Preferences` in free-text format and contains engine to process the constraints. For composition patterns, the framework has customized service relationship and flow management patterns.

The framework implementation described in Mandell, D. *et al.* relies exclusively on the functional parameters provided by BPEL4WS and maps the inputs and outputs in BPEL4WS to OWL-S profile. Their approach does not extend BPEL4WS to include non-functional or service request parameters.

The XSCBR framework supports OWL-S specification and extends it for including granular level service requests and for QoS based non-functional parameters.

Transparency and Automation

In the planning based framework by Wu, D., *et al.*, the service requestor starts with a simple user interface where an OWL-S service description for any desired task can be loaded. When the service description for the example domain is selected, a form to enter the required parameters for the task is presented to the user. This form is generated based on the ontologies used to describe the input parameters of the service. The UI will also automatically fill out some of the fields such as the home address from a user specified knowledge base. The user is shielded from the complexities of semantic web and the background composition process.

The service provider requires providing a DAML-S description with the WSDL file however the framework does not consider the discrepancies in the service descriptions. In order to do planning in a given planning domain, SHOP2 needs to be given the knowledge about that domain. A SHOP2 knowledge base consists of operators and methods (plus, various non-action related facts and axioms). The framework provides a DAML-S to SHOP2 translator, which is a java program that reads in a collection of DAML-S process definitions files and outputs a SHOP2 domain file. The final output of SHOP2 is a sequence of Web services calls that can be subsequently executed. However the composer receives no assistance in addressing knowledge discrepancies. This way the composer is semi-transparent from the process of composition.

In the framework advocated by Richards, D. *et al.*, the service requestor is masked from the underlying mechanism that is managed through agent technology. The service provider requires providing DAML-S description of their services however the mapping of such services to agent factory is performed transparent from the service provider. The role of service composer and transparency provided to the composer is not apparent from the publications as they contain internal working of Agent Factory to create composition schemes however how much composer is involved in composition process is not explicitly specified.

Mandell, D. *et al.* propose a bottom-up approach for Web services interoperation in BPEL4WS; they use OWL-S based descriptions for runtime binding of service partners. The Implementation collects the OWL-S profiles into a repository and exploits the

profile semantics to query partners for desired properties. The service requestor is transparent from the background process, however there is no mention of wrapper supporting service provider to map WSDL to OWL-S. The service composer has to build composition schemas for abstract services hence the framework provides limited transparency to the composer.

In Limthanmaphon, B. *et al.*, the proposed framework utilizes UDDI for service directory and since UDDI service registries are described in XML-based format, the framework use JAXR (Java API for XML Registries) to parse the XML-based service descriptions and to find out a service matching the query by comparing the XML attributes of each service and hence enable a wider range of service selection. The framework relies on UDDI and therefore inherits the limitation of UDDI where the user has to form smart key words in order to receive adequate responses making the framework less transparent. The service provider submits the services to the UDDI where unlike semantic web based approaches provider just has to provide a WSDL description and utilize UDDI – standards already familiar and in practice in the industry. However, in the situation where the provider has service specification and format conceptually similar but syntactically different from what the composition interface expects, the approach does not offer any support to the providers. The authors provide wrappers for converting service descriptions to CBR descriptions however the reliance of free-text restricts the opportunity for performing discovery and composition in dynamic manner.

Compared with the above discussed approaches, the XSCBR framework delivers a user-transparent search engine that exhibits high precision and recall of results without requiring intervention from their part keeping the utilization of framework completely transparent from service requestors.

Using XSCBR, the service developers have the opportunity to reuse their services in composed applications part of a composition and can provide or simply use existing knowledge which bridges the gap between their intentions in describing services and the standard service interfaces they refer to. The role of composer - someone who analyzes service request and matches against service offers and if required combines number of offers to meet the request, is transformed to a domain expert or administrator- someone maintaining XSCBR system since the framework alleviates the burden of verifying the service capability and interpreting requests. However the composer is involved in the

integration of services when the framework is initiated hence does not contain any cases and when the framework returns no results in response to a service request making the utilization of the framework semi-transparent to the composer.

Adaptability

The frameworks by Richards, D., *et al.*, Limthanmaphon, B., *et al.* and Wu, D., *et al.* do not prioritize adaptation as there is no mechanism present to deal with the situation where the participant services have different service descriptions to what is expected by the respective composition frameworks. Similarly, these frameworks provide no mechanism to find out whether service is active or disabled. The framework by Mandell, D. *et al.* allows to fire services with concrete details using abstract processes supporting limited level of adaptability.

The XSCBR framework allows addressing discrepancies by accepting OWL and SWRL rules that bridge such discrepancies. The framework also has provision of error reporting unit containing logic for error reporting. The errors occurring during the functioning of the system along with any unavailability of services for a suggested solution are stored as a new case to avoid future failures. However, when using error reporting unit, the administrator has to manually delete such case when notified as a permanent error. Hence, the framework has provision to consider faulty and inactive service however is performed manually. Table 23 outlines the comparison of XSCBR with these frameworks.

Table 23 Comparing frameworks

Frameworks	**Expressiveness**				**Level of transparency**				**Level of support for adaptation**	
	Support for Functional Parameters	**Support for Non-Functional Parameters**	**Support for Granular level service requests**	**Support for composition patterns**	**Requestor**	**Provider**	**Composer**	**Automation**	**Discrepancies in service interfaces**	**Faulty, Inactive services**
Wu *et al.* [42]	√	×	×	√	Complete	Limited	Limited	Limited	None	None
Richards *et al.* [56]	√	×	×	√	Complete	Limited	Inconclusive	Limited	None	None
Limthanmaphon *et al.* [59]	√	×	√	√	Limited	Limited	Limited	Limited	None	None
Mandell *et al.*[67]	√	×	×	√	Complete	Limited	Limited	Limited	None	Limited
XSCBR	√	√	√	√	Complete	Limited	Limited	Limited	Limited	Limited

6.4.3. Quantitative evaluation of XSCBR framework

This section presents the result of experiments, which were carried out to evaluate the incurred impact of XSCBR on the quality of precision and recall of XSCBR Web services discovery and composition mechanism.

Experiments design

The frameworks which are analyzed and compared with the XSCBR development are an implementation of OWL-S matchmaker and jUDDI [99] which is a private UDDI registry.

The main motivation behind selecting these tools for comparison is to evaluate the XSCBR framework against two diverse approaches which are widely adopted by industry and academia. The UDDI registry is an XML-based registry based on Universal Description Discovery and Integration protocol and utilizes XML for Web services discovery and matchmaking. Although the mainstream public registries were closed in year 2006 after successfully demonstrating the interoperability and robustness of the UDDI specifications through a public implementation, the majority of software vendors now include private UDDI registry support as a key feature in their software products where private UDDI registries are being broadly deployed to solve application and service integration challenges [100]. For experiments, the jUDDI private registry is used, which is an open source java implementation of the Universal Description, Discovery, and Integration (UDDI) specification for Web Services.

The other tool used for comparison is an OWL-S matchmaker which solely relies on matching Web services functional parameters (inputs and outputs) with service request. OWL-S ontology provides a mechanism to describe the capability of Web services in machine-readable form, which makes it possible to discover and integrate Web services automatically. The matchmaker built on top of OWL-S relies only on functional description of services, while in XSCBR we extend OWL-S with non-functional attributes and encode CBR reasoning in the Web services discovery process.

The evaluation of the frameworks is performed on two categories of queries: coarse-grained and fine-grained. The classification queries are an example of coarse-grained queries and contain taxonomical terms for search. For example, a service requestor looking for a service from taxonomical domain such as airline or sensor domain. The

classification queries are perceived to be less informative and generally serve as a precursor to fine-grained search where the user provides more information after initial retrieval such as constraints or preferences on some results.

Fine-grained queries are detailed queries with complex search set. An example from the travel domain can be requesting a bus service with payment in currency USD or an airline service with execution price of 30, with execution duration 0 seconds. Note that it is not possible to perform such queries on the OWL-S matchmaker as the matchmaker does not support such level of fine-grained queries.

Experimental setup

The experiments were performed on closely coupled workstations within the Nottingham Trent University departmental LAN, the connection speed of which is 100 Mbit/se. The web services are developed using Apache Axis 1.2 and are hosted on web services container provided by Apache Tomcat server 5.5. The hardware configuration for the end user is with AMD Athlon XP, 2.01GHz processor, 1 GB of RAM and 100 Mbit/sec connection speed running on Windows XP platform.

The XSCBR framework and OWL-S matchmaker are implemented using NetBeans Integrated Development Environment (IDE). Both implementations utilize reasoner APIs from Pellet, Jena and Bossam. The implementation of ontologies and rules utilized in frameworks is using Protégé editor.

The jUDDI version 0.9rc4 was installed and configured with tomcat 5.5 supported by MySQL server 5.0 for persistence storage. The publication and discovery of services was performed using Eclipse 3.2.1 IDE with Web Tools Platform (WTP) plug-in for Web services.

In the developed test bed, there are 150 Web services with a variety of sub-domains from travel industry and 250 cases involving these web services in the case library.

The following sections outline definitions of recall and precision for Web services discovery and matchmaking and how the XSCBR framework is evaluated based on these interpretations.

Recall and Precision for Web services discovery and matchmaking

We adapt definitions of `Recall` and `Precision` from the IR literature and define these measures for Web services discovery and matchmaking as follows:

$$recall = \frac{Number\ of\ \ relevant\ Web\ services\ retrieved}{Total\ number\ of\ \ relevant\ Web\ services\ in\ registry}$$

Equation 6 Recall of Web services search engine

$$precision = \frac{Number\ \ of\ \ relevant\ Web\ services\ \ retrieved}{Total\ number\ of\ \ Web\ services\ \ retrieved}$$

Equation 7 Precision of Web services search engine

`Recall` is a measure of the completeness of the results achieved by counting number of relevant Web services retrieved by search engine against total number of relevant Web services in the registry as perceived by human observer; `precision` measures the usefulness of results by counting number of retrieved relevant web services out of the total number of Web services retrieved by the search engine.

Evaluation for classification queries

In the XSCBR framework the provision for explicit consideration is achieved by recording atomic services as part of `serviceIsPartOfSolution` object property in the definition of `Solution` where value for this object property points to the URI of the candidate solution services. This explicit consideration allows requestor to query available services by providing `PreferenceonSolution` and allows the framework to achieve close to 100% for classification query based precision. To contrast these results with the other frameworks, the OWL-S matchmaker and the jUDDI framework were evaluated on average 10 requests where the queries were formed with a variation in the number and type of domains. Figure 49 depicts experimental results obtained for classification queries. The abbreviations used in the figure are: Pr= Precision and R=Recall.

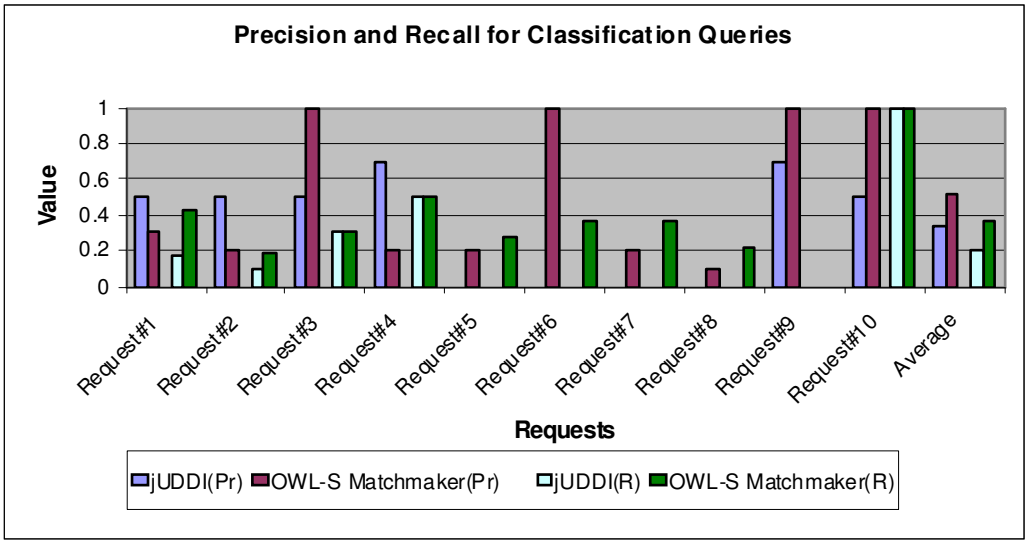


Figure 49 Precision and Recall study (Classification Queries)

For the OWL-S matchmaker, the classification is based on the number of inputs and outputs along with types of these inputs and outputs required for a particular class of domain and comparison is performed based on comparing the number and type of these inputs and outputs from service request and from available services in registry. For example searching for Airline domain services requires providing the following inputs/types pair: (Date of Departure, Date), (Date Of Arrival, Date), (City of departure, City), (City of arrival, City), (Name of passenger, String) and (No of passengers, Integer) while searching for hotel domain related services less number of inputs i.e., (Date Of Arrival, Date), (City of arrival, City), (Name of passenger, String) and (No of passengers, Integer). OWL-S matchmaker achieves average 35% recall and 52% precision for such classification queries.

To explain why XSCBR compares favourably against OWL-S, let's assume that the user is looking for a hotel domain service, in OWL-S that means inputting the above mentioned four inputs, however this will disqualify any composed service which is integration of Hotel and Airline services, because the composed service will have more number of inputs than aforementioned four inputs.

For experimenting with jUDDI, we form as many random queries as possible and retrieve results from the registry and for each request we take mean value of these results. jUDDI search achieves lowest results in terms of 20% recall and 33% precision.

R-Precision for ranking based Web services discovery and matchmaking

For evaluating frameworks for fine-grained queries the outlined measures of precision and recall are insufficient as these measurements are set-based measures in that they are computed using unordered sets of documents [18]. Therefore it requires using alternative measures to evaluate ranked retrieved results that are more accurate measurement of web services search engines such as XSCBR. To summarize, recall and precision are measures for the entire recalled results and they do not account for the quality of ranking the results have in the recalled results while the requestor would ideally want the retrieved services to be ranked according to their relevance to the query instead of just being returned as a set. R-precision provides a solution for achieving such type of analysis.

Following is the definition of R-Precision when applied to the Web services discovery and matchmaking problem [18]:

$$R\text{-}precision = \frac{r = Relevant\ Web\ services\ from\ Top\ R\ number\ of\ results}{R = Relevant\ Web\ services\ to\ request}$$

Equation 8 R-Precision of Web services search engine

If `R` is the number of relevant services to a request and `r` is the number of relevant services from the top `R` results of a framework, then R-Precision for this framework will be `r/Rel`. For example, a query "Bus service with payment in currency USD" has 5 relevant service, and a particular framework's response to the query with top 5 results has 3 relevant services then the `R-Precision` for such framework will be 3/5 = 0.6.

In the experimentation, 10 fine-grained queries were performed on jUDDI and XSCBR framework to find out average R-Precision for both the frameworks. As jUDDI (UDDI in general) only supports matchmaking and has no provision for composition, to evaluate on fair ground the XSCBR framework was turned off. Figure 50 charts result of the experiments for comparing R-Precision for jUDDI and XSCBR frameworks.

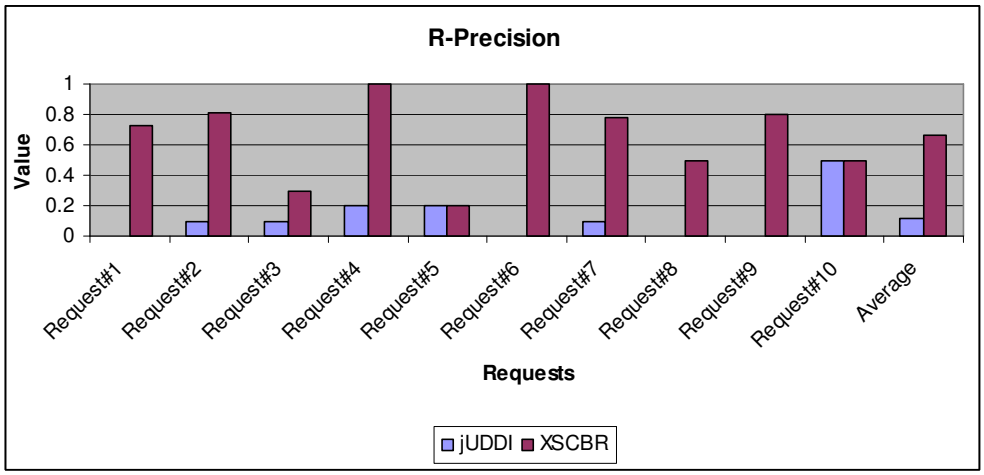


Figure 50 Comparison using R-Precision

The average R-Precision for XSCBR is 67% compared to 12% in jUDDI. The support for high-granularity in case representation to describe service requests make it possible for XSCBR to allow specifying fine-grained queries and interpret them semantically. However, the precision performance depends on availability of knowledge in terms of cases, hence as high as 100% precision for requests such as request 4 and request 6 while as low as 0.2 for request 5.

Performance study

The performance of the framework largely depends on the efficiency of the underlying semantic reasoning tools. In this effort, Pellet for DL semantic reasoning and Bossam for rule reasoning were utilized. The performance also depends on the complexity of the user requests. For example, if the framework doesn't require rules to describe constraints, the computational time can be decreased significantly. Therefore, the framework is capable of switching off peripheral modules such as those dealing with advanced constraints to achieve better performance and for serving non-critical systems.

Figure 51 shows the result of performance study on frameworks. Please note that the time is inclusive of external factors such as background threads served by CPU. As shown in the graph, average request is answered by jUDDI in 98.84 sec, by OWL-S matchmaker in 212.82 seconds and by XSCBR framework in 370.94 seconds. The results are indicative rather than conclusive as there are various optimization techniques employed by a mature implementation such as jUDDI, where the other implementations use basic optimization techniques. However, these results highlight the fact that the reasoning in OWL-S and XSCBR framework is slower than a database search

methodology adopted by frameworks such as jUDDI registries. The difference in the execution time of OWL-S matchmaker and XSCBR highlights the fact that the addition of knowledge consideration in XSCBR on top of OWL-S functional parameter matchmaking has considerable overhead.

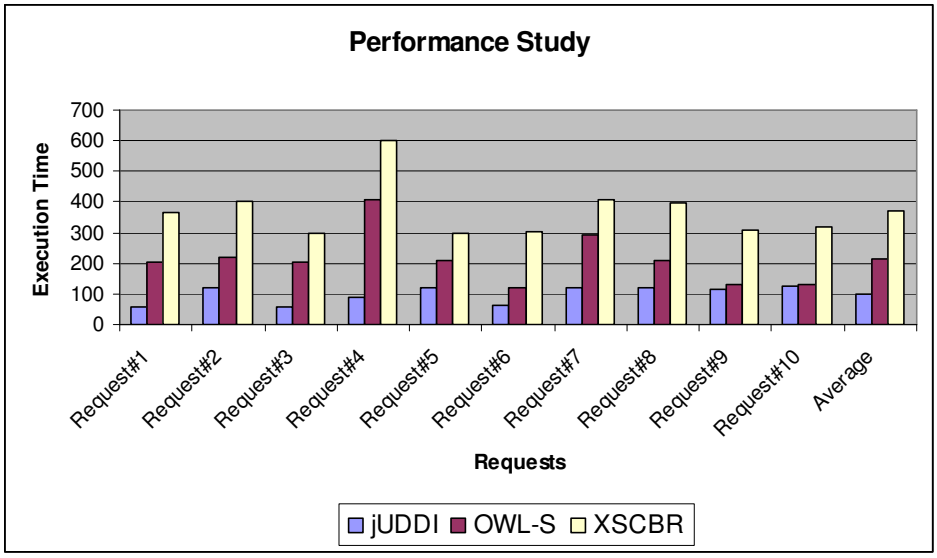


Figure 51 Performance study

In addition to aforementioned reasons, there are some operational limitations of semantic reasoning from the perspective of the execution speed, which is possible to overcome by a careful design. The root of this problem can be traced to the inherent performance penalties with XML processing. XML processing is considered resource and time intensive task compared to text processing, as XML trades some size and efficiency for the advantages of a portable, transparent information format. As semantic web languages build on the layers above XML, the processing is expected to take some toll [101] [102]. However, similar to XML, if the right tools and techniques are used, it is possible to build production system that could work with the magnitude of the real-world search engine that has acceptable level of performance. The rest of this section highlights some of the approaches we have considered for building optimized XSCBR system.

One of the major performance leaks identified during the research is the extent to which the loading ontology models into memory affects the efficiency of the overall system. The process of loading ontology models is slow as the reasoner needs to store and retrieve the entire tree structure from the memory. To address this, "good practice principles" were outlined and followed where efficient use of memory model for the import and call back results in improved system response time. Following is the summary of such principles:

- To use different models for storing different ontologies could result in slow system. Therefore, when possible, create optimum number of models and share the same model between numbers of ontologies while trading balance with the modularity.

- In case of ontology being used widely in the java class, creating model storing this ontology within *private scope* of the program is less effective then having one within *public scope*.

- Another main performance leak of the program was identified as the use of imported ontologies. We employ an off-line caching system to enable framework to access the public ontologies locally.

This optimization improved responsiveness of XSCBR framework by a factor of approximately 1.5 as the average request is executed in 370.94 seconds compared to 668.04 seconds earlier.

These results confirmed objective of using Case Based Reasoning with semantic web for automated Web services discovery and composition to improve the precision and recall of Web services search with acceptable level of performance.

Observation on the computational and space costs of CBR systems

The complexity of a CBR system hence the computational and space costs are dependent on balancing two factors in the development: a) the coverage of a case base, i.e. the range of problems the system can solve in particular domain and b) the level of competence of the system, i.e. system precision.

As a general principal, new cases are added in the case base initially as they offer different perspective for the problems hence more likely to improve overall case-base coverage. However as the case base grows new cases are more likely to overlap with existing cases and so offer little in the way of new coverage [103]. The CBR system developer needs to find a methodology which guides the case population process to decide which case is worthy of storage and at the same time ensuring storage, computational penalties.

Further discussion on this subject is beyond the scope of this research and interested readers are referred to the works of Smyth & Keane [104] and McKenna [105] which

describe techniques for measuring the local coverage of individual cases with respect to a system's retrieval and adaptation characteristics.

6.5. Conclusions

The aim of this work is to create a framework that alleviates the burden of dynamic Web services composition. The argument is that despite the evident popularity of Web services as a secure distributed computing paradigm and the value-added dimension that composition adds to it, the practical adoption of the technology is still hindered by the knowledge and effort required for the compilation of the composition process and the manual adaptation of new and existing web services to it. A semantic case based reasoner was implemented, which captures Web service execution experiences as cases and uses these cases for finding a solution for newly posed problems.

A qualitative and quantitative analysis was carried out to evaluate the XSCBR framework. In the quantitative analysis experiments were carried out to investigate and evaluate the effectiveness using the recall, precision, R-precision measurements and execution time for measuring efficiency of the framework. The results of the experiments have shown that XSCBR has higher precision compared to UDDI and OWL-S albeit there are performance penalties in developing a higher level semantic web based tool such as XSCBR. It is feasible to believe that this performance shortcoming will steadily improve, as processing costs decrease and the need to more intelligently and automatically integrate services increases; hence the evaluation outcome was favourable and that the overheads are acceptable since the developer has the opportunity to balance the performance against the application requirements giving the indicator of applying XSCBR approach to automated Web services discovery and composition.

In the qualitative analysis, the XSCBR framework was evaluated against the research objectives set out at the beginning of this research. The XSCBR framework has satisfied all of these objectives in terms of providing a transparent and dynamic search engine for Web services discovery and composition.

7 Chapter Seven: Conclusions

This chapter highlights the contribution of this research work in utilizing semantic web based Case Based Reasoning (CBR) methodology for automated Web services discovery and composition.

In principle, every solution brings new limitations. Thus, the limitations of the proposed framework and unresolved issues are also discussed in this chapter. The lesson learnt during investigation stages of research are presented as guidance for further research.

This thesis is an effort to enrich the scientific knowledge of the automated Web services discovery and composition technology. Although there are no limits to scientific knowledge in general, the contributions this thesis embodies may motivate the research debates of evolution in Semantic Web research with respect to Web services composition. Accordingly, an outline of how the presented work can be improved by further research effort is given.

7.1. Overview

XML based Web services technologies have emerged as the de-facto middleware that can openly facilitate wide range of applications within enterprises and over the Internet. The seamless composition of such Web services using composition methodologies can be considered as the value-added dimension to the applicability of Web services.

The aim of this work is to create a framework that alleviates the burden of dynamic Web services composition. The argument is that despite the evident popularity of Web services as a secure distributed computing paradigm and the value-added dimension that composition adds to it, the practical adoption of the technology is still hindered by the knowledge required for the compilation of the composition process and the manual

adaptation of new and existing web services to it. The main thesis of this research work is based on the theory that assistance with the facilitation of the composition process to the service providers and the composers play a major role in encouraging the adoption of the Web services technology. For the service composer, which can be a human developer or intelligent agent, the facilitation constitutes automating as many steps as possible in order to build and program the composition logic. The facilitation to be provided to the service providers can be considered in terms of minimizing the effort they have to endeavour to subscribe their services to composition schemes.

The investigation identified that by and large enterprise based solutions utilize static composition methods which require performing composition stages of service selection and flow management done a priori and manually. Considering the growth of Web services and the scale and velocity with which new services are made available for Internet-based services, the manual effort involved in static composition is cost-prohibitive.

After critical analysis of current approaches to Web services composition, the conclusion was that there is scope for developing a practical and current solution that merges the benefit of practicality of use and adoption popularity of static workflow-based (BPEL-based) composition, with the advantage of using semantic description to aid the composition participants in automatic discovery and interoperability of the composed services.

To address this issue, a hybrid Web services composition framework was created that exploits BPEL for practicality of use and adoption popularity of workflow-based composition while utilizing semantics to aid both service providers and composers in building the composition scheme and adapting new Web services to it. In this framework, the domain functionality described in WSDL-XML grammar is accompanied by a semantic description of service parameters expressed in OWL ontology, allowing the description of the expected domain functionality in an unambiguous form and catering for any mismatches in the Web services description. A domain membership verification module was developed that allows the service providers to adapt their application services to the domain interface and making them with minimal effort.

Once a domain Web service is declared composition-ready, the dynamic composition framework transparently integrates the Web service into the BPEL process file, i.e. it is

automatically added to a pool of dynamic Web services for this domain. Chapter 3 described an algorithm for dynamic population of the domain pool with Web services, thus allowing the service composer to effortlessly select any possible combination of services from the composition domains and fire the composed service.

This hybrid approach presents a practical solution to a current and an urgent problem. However, the approach could achieve only limited automation to the composition process because for the composition engine to provide automatic discovery and flow management, the process model needs to take into the consideration of the semantics in the specification. The addition of semantics within an XML centric standard like BPEL will not achieve the sought-after automation as that would require an intelligent reasoner that can interpret the semantic description. Hence, the second phase of research introduced an intelligent semantic-based reasoner that builds on the AI theory of Case Based Reasoning.

Semantic description of Web services' profiles paves the way for automating the discovery and matchmaking of services since it allows intelligent agents to reason about the service parameters and capabilities. However, the accuracy of such automatic search mechanism largely relies on how soundly formal methods working on such semantic descriptions consume them.

In the second phase of the research work, the importance of considering the execution values for semantically described non-functional Web services parameters was stressed in decision making regarding Web service adequacy for the task. This is because the service behaviour is impossible to presume prior execution and can be only generalized if such execution values are stored and reasoned for deciding service capability. The AI planning and Intelligent Agent based reasoning methods provide rule-based reasoning methodology rather than experience-based. A Semantic Case based Reasoner (SCBR) was implemented, which captures Web service execution experiences as cases and uses these cases for finding a solution for new problems. The implemented framework extensively uses ontologies, as semantics are used for describing the problem parameters and for implementing components of CBR system: representation, indexing, storage, matching and retrieval. These components are modelled based on ontologies, making the application logic captured within semantic descriptions. The semantic approach for modelling CBR reasoner achieves the required automation in the Web

services discovery and matchmaking processes and also makes the CBR reasoner extensible and reusable.

In the following stage of research SCBR framework was extended to facilitate dynamic Web services composition using CBR methodology of case adaptation. The resultant framework is termed as e**X**tensible **S**emantic **C**ase **B**ased **R**easoner (XSCBR).

The final framework (XSCBR) also provides a standardized case representation format that is applicable to any application domain. The proposed generic case representation has provision for the facilitations to service participants and especially service requestor as we outlined extension of OWL-S descriptions to support service requestor in terms of specifying constraint and preference on search. The rules were introduced to handle complex constraint and preferences relationships in the case descriptions. The Quality of Service (QoS) was also considered as an important selection criterion for web services discovery and modify (QoS) descriptions from the previous framework inline with the modifications and improvements in the case representation, especially using rules.

In the XSCBR framework, a methodology was outlined to represent solution semantically in order to adapt the solution for new problems whenever required. This is made possible using existing OWL-S process model for modelling adaptation in the XSCBR framework. The richness of the workflow based patterns supported by OWL-S process model and provision of semantics in the specification itself were the reasons selecting OWL-S. The *ModifyOWL-S* algorithm was outlined that formalizes the steps to modify OWL-S process file with the help of API to insert or remove service reference to satisfy new problem requirements.

In the XSCBR framework, case adaptation is applied to solve a new problem by merging the local solutions from previously solved problems and creating a globally consistent solution for the new problem. In-depth investigation of the CBR literature concluded that there are two major challenges in CBR: the first is the development of an efficient methodology for case adaptation, and the second is maintaining consistency of solutions and knowledge supplement to assist case adaptation.

A methodology based on Constraint Satisfaction Problem (CSP) was designed to address these challenges and handle the inconsistency problem with a CSP inspired Domain Dependency Module (DDM). The approach allows defining dependency

between variables of the domain and ensuring the consistency by maintaining dependency constraints between these variables whenever a solution is adapted.

In addition to maintaining the consistency of solution in the XSCBR framework, a knowledge-intensive approach is advocated to automate the process of adaptation in CBR inspired Web services discovery and composition problem. The argument is that the Web services composition is a developer-intrusive problem solving method, the automation of which requires the reasoning about domain-specific knowledge at their disposal. In contrast, existing composition approaches focus only on semantic descriptions of web services but are oblivious to the fact that in the process of Web services discovery discrepancies could occur and knowledge is required to address them.

Finally, qualitative and quantitative analysis was carried out for the framework. In the quantitative analysis experiments were performed to investigate and evaluate the effectiveness using the recall, precision, R-precision measurements and execution time for measuring efficiency of the framework. The results of the experiments have shown that XSCBR has higher precision compared to UDDI and OWL-S albeit there are performance penalties in developing a higher level semantic web based tool such as XSCBR. It is feasible to believe that this performance shortcoming will steadily improve over time, as processing costs decrease and the need to more intelligently and automatically integrate services increases. The conclusion was that the evaluation outcome was favourable and that the overheads were acceptable since the developer has the opportunity to balance the performance against the application requirements giving the indicator of applying XSCBR approach to automated Web services discovery and composition.

In the qualitative analysis, XSCBR framework was evaluated against the research objectives set out at the beginning of this research. The XSCBR framework has satisfied all of objectives in terms of providing a transparent and dynamic search engine for Web services discovery and composition.

The main contribution of the thesis is summarized in the following section.

7.2. Thesis contributions

This work has been undertaken at The Nottingham Trent University, School of Science and Technology as one of the Semantic Web services research network activities, within

the research track of Web services composition. Some of the contributions of this work have been published in [106] [107] [108].

A major contribution of this thesis is the development of an intelligent engine by modelling existing formal approach (Case Based Reasoning) to support the Semantic Web service composition problem. The contributions (Ci) of this thesis made to scientific knowledge are outlined in the following points.

C1: In this research, the concept of run-time behaviour of services and it's consideration in the Web services selection process was introduced. The static behaviour of a service can be measured in terms of whether the service has similar description to problem in terms of functional and non-functional parameters. The run-time behaviour of a service is the result of service execution and how the service will behave under different circumstances, which is difficult to presume prior to service execution. Moreover, with implementation it was demonstrated that the accuracy of automatic matchmaking of Web services can be further improved by taking into account the adequacy of past matchmaking experiences for the requested task. The research utilized experience-based reasoning methodology, Case Based Reasoning (CBR) to capture run-time behaviour of services.

C2: We have persuaded a pragmatic approach to the problem of Web services composition that strongly advocates considering the facilitation provided to the participants in the composition process. We believe that the facilitation to participants is required as Web services composition is very complex task and also believe that facilitation to participants will encourage yet further adoption of the Web services technology.

C3: In this research, the existing OWL-S specification was extended for facilitating service requestor in providing components to support a finer level of requests. We believe that we have contributed to the ongoing efforts [109][110] to support natural language queries for search, as our extension component to OWL-S covers very diverse range of queries that rely on constraints and preferences on the expected results.

C4: We identified number of areas of further research while investigating the use of semantics to inject intelligence into Web services composition. One such problem has not yet addressed sufficiently is the interoperation between independently developed reasoning engines for semantic matchmaking and composition. Without this

interoperation, the reasoning engines remain imprisoned within their own framework, which is a drawback, especially that most engines usually specialize in serving a particular domain, hence interoperation can facilitate inter-domain orchestration. The XSCBR framework extensively uses ontologies, as semantics are used both for describing the problem parameters and for implementing components of the CBR system: representation, indexing, storage, matching and retrieval. We believe that in this work we took a small step towards standardization at the reasoner level by describing the CBR reasoning model semantically.

C5: The central theme of our approach to the problem of automated Web services discovery and composition is to explore existing solutions before devising a full-fledged solution from scratch. To achieve this we apply case adaptation process by adapting local solutions from previously solved problems to create a globally consistent solution for the new problem.

While investigating case adaptation we discovered that the process of modifying existing solutions is more complex than reported in the literature. When individual service instances are composed, they might compromise the consistency of the solution primarily because they originate from different solution sets. We have designed a methodology based on Constraint Satisfaction Problem (CSP) to address this challenge and address the inconsistency problem with a CSP inspired Domain Dependency Module (DDM). Our approach allows defining dependency between variables of the domain and ensuring the consistency by maintaining dependency constraints between these variables whenever a solution is adapted.

To conclude, the work described in this thesis presents a significant advance towards the aims of the research and all the stated objectives were achieved.

7.3. Limitations

This section outlines limitations of the XSCBR framework.

Service composer transparency

One of the goals of the XSCBR framework was complete composer transparency to the composition mechanism. However, the problem of acquiring knowledge to bridge discrepancies led us to adopt a user-intrusive approach where the framework is reliant on the composer to supply the knowledge necessary for the framework operation. The

priority was to model human behaviour in solving composition problem and arm the system with knowledge, but that was achieved at the price of transparency to the application composer as the system expects composer to provide heuristics and casual models to bridge discrepancies in service descriptions.

The knowledge acquisition in CBR concentrates on acquiring knowledge such as adaptation rules that can bridge discrepancies in case descriptions and casual models that can provide rules to guide the reasoner in case of a particular decision making situation. Hence, in Case Based Reasoning (CBR), knowledge acquisition plays an important role allowing to progressively improving the system's functionality.

However, the process of knowledge acquisition is a complex and time-consuming task in general which requiring tools to assist with the acquisition in the process of case-based reasoning. There are some approaches in the literature that allow generating knowledge automatically. The approach presented in [111] consists of techniques to generate knowledge by recording property value differences of each pair of cases in case library. In their approach they create rules by finding difference in values of this pair of cases and making this difference as antecedent part of an adaptation rule, with the consequent part of the adaptation rule being the differences between the solutions in the compared cases. For example, if the case library contains following two case_A and case_B (See Figure 52) then comparison of property value differences between case_A and case_B gives rule R1 as follows:

Property	**Value**
Case_id	case_A
Nr_Bedrooms	2
nr-rec-rms	1-rec-rm
kitchen	good-kitchen
prices	20500

Property	**Value**
Case_id	case_B
Nr_Bedrooms	2
nr-rec-rms	2-rec-rm
kitchen	excellent-kitchen
price	25000

Figure 52 Generating adaptation knowledge

R1: if the value of the kitchen changes from excellent to good and the value of nr-rec-rooms changes from 2-rec-rooms to 1-rec-rooms then the house price is decreased by 4500.

In the same light, [112] propose an approach of knowledge learning based on a particular search technique called *frequent pattern extraction*.

These approaches are promising however are more effective in acquiring initial-stage knowledge only and need further work in order to expand the knowledge using learning process throughout the functioning of CBR cycle.

Reliance on the reasoner tools and performance penalties

The semantic web reasoners are inferior in functionality and performance compared to XML parsers and other XML-centric tools. The performance of reasoners is still far from achieving relational database par efficiency.

An example of the immaturity of standards is the disagreement surrounding the necessity of a more expressive language (SWRL) that exploits the capabilities of OWL descriptions to include complex relationship in terms that they include mathematical and logical operations. There are two main schools of thought: the first, to which this report author subscribes to, is that OWL-DL is not sufficient to provide the reasoning capability required by applications such as Web services composition engines and rule language such as SWRL is required to define complex concept relationships. The other school argues that SWRL descriptions lead to undecidability which is the main asset of OWL-DL descriptions hence not worth sacrificing.

This sort of uncertainty about OWL-DL extensions (such as SWRL) has dampened the enthusiasm for the development and optimization of tools that can support SWRL reasoning, and at present available tools either have preliminary support (such as Pellet) or are computationally expensive (such as Bossam). As we rely on Bossam for implementing our framework, the performance of our framework is affected. In the future, once the arguments are settled, we envisage that there will be efficient tools supporting SWRL reasoning and our framework will be able to take advantage of them.

Efficient Case Library Management

In the current version of the framework, the insertion of cases is automatic while the maintenance by editing and deletion is manual. The general principal applied in SCBR for inserting new cases in the case library is as follows: if a case extends or affects existing knowledge than the case shall be stored in the case library. This translates into

comparing the execution value of a new case against the stored cases and if they are better (hence contributes knowledge), then the case is worthy of storage. However, garbage collection of cases as a result of error-reporting and long term maintenance (periodical cleanups) is the responsibility of the administrator of the framework. In production systems where the framework utilization will be continuous and intensive, manual maintenance is a clear limitation.

7.4. Observations and Lessons Learned

Following are the number of observations made while investigating the use of semantics to inject intelligence into Web services composition.

Interoperation of Composition Engines

The interoperation between independently developed reasoning engines for semantic matchmaking and composition. Without this interoperation, the reasoning engines remain imprisoned within their own framework, which is a drawback, especially that most engines usually specialize in serving a particular domain, hence interoperation can facilitate inter-domain orchestration. We believe that in this work we took a small step towards standardization at the reasoner level by describing the CBR reasoning model semantically.

Sharing ontologies

Another issue is related to the use of ontologies. Traditionally ontologies are constructed for each new semantic web project limiting the reuse of existing knowledge structures. The reasons for such flawed approach include diversity of domain knowledge, different perspective for the same ontologies and most importantly the close coupling of domain knowledge with reasoning processes. Semantic web based Web services composition approaches need to address this problem in order to benefit from existing implementations. Industrial experience in taxonomy specifications [113] [114] for different domains can be a guideline to overcome this limitation. Similar to the taxonomy standardization, ontology elements or concepts can be standardized facilitating re-usability.

Tools for Transparency

Another problem demanding further investigation is related to the readability of the implementation code while using XML based standards i.e. OWL, due to the strong-type syntactical structure restrictions in XML. Semantic web being based on XML layers complicates this problem further as the readability of the ontologies is very poor. In addition applications built on semantics require great deal of knowledge from the developers related to artificial intelligence, logical reasoning and knowledge management demanding major efforts to build tools that abstract underlying complexity.

7.5. Future Work

Based on the aforementioned limitations, we propose some outstanding research issues that can take our effort further.

Close-coupling with planner

In the framework, when the Knowledge Based Substitution (KBS) algorithm fails to serve user's request with the process of case adaptation, we leverage the request for the AI planner to perform transformations where planner generates composition schemas from existing service descriptions. Although the experience of integrating semantic web with case based reasoner for solving the problem is successful, we would like to extend the current framework by passing the knowledge KBS algorithm might have gathered in the failed attempt to find a solution, to an AI planner. The planner can then benefit from this matchmaking attempt rather than relying on service descriptions to solve the composition problem from the scratch. This knowledge could be in terms of narrowing down the number of services planner has to inspect in order to build composition schema or such knowledge can state preferred services for the planner to utilize in the composition.

Extension to include other semantic web services specifications

In last couple of years, the semantic web community have seen the emergence of alternative semantic web services specifications to OWL-S such as WSMO (Web Service Modelling Ontology) [23] and Web services semantics (WSDL-S) [24].

WSMO provides ontological specifications for describing the core elements of Semantic Web services and consists of four main elements: (1) ontologies that provide the terminology, (2) goals that state the intentions that should be solved by Web services, (3) Web services descriptions that define their various aspects, and (4) mediators which resolve interoperability problems. WSMO specification proposal was submitted to the W3C in June, 2005. WSDL-S defines a mechanism to semantically annotate Web services described using WSDL. Annotations can be provided with different ontology languages (e.g. OWL, UML). WSDL-S specification proposal was submitted to the W3C in November, 2005.

The XSCBR framework can take advantage of these emerging service specifications by providing description support in all three major specifications of OWL-S, WSMO and WSDL-S while maintaining the intelligent layer of CBR that works on top of service descriptions. This way the framework will achieve wider adoption in the semantic web services community. The major challenge here would be sustaining the transparency offered by SCBR to their participants considering the introduced heterogeneity in service specifications.

Integration with Grid computing

We have recently witnessed increases in demand-driven access to computational power by integrating heterogeneous, distributed systems in a so-called computational grid. In recent times, many enterprise software vendors have borrowed from this concept, offering on-demand access for software applications prone to peak congestion patterns or what is being marketed as a *pay-as-you-use* mechanism, a process which is strikingly similar to other non-IT grids, such as the electrical grid [115]. As it turns out, this process of virtually pooling computing resources and making them readily available via a network presents many of the underlying issues resolved by Web services technologies such as security, reliability and scalability common to distributed computing. In light of these similarities, a special initiative whose intent is to jointly advance grid and Web services technologies was created, its name: Open Grid Services Architecture (OGSA) [116].

We clearly see the benefit of applying our XSCBR service discovery and composition mechanism by mapping web services to OGSA grid services and by managing other grid administration tasks. The dynamic discovery of grid resources this way to address

computational requirements on the fly will rapidly increase the motivation towards deploying more applications.

Semantic Web based Query Expansion for interpreting user requests

Lately query expansion (QE) techniques [117] have gained a lot of attention in attempting to improve the recall of document and media queries. QE methods fit naturally into our web services retrieval technology as we rely on computing the aggregate degree of match (ADoM) for the semantic relations describing a particular service to determine its match to the original query. Hence, we can easily determine the quality of the returned results in terms of accuracy and volume and decide whether to apply QE techniques for replacing the query concepts to improve the quality of the recall. This is particularly feasible for semantic-based knowledge bases as they provide language expressiveness for specifying the similarity of the concepts (Implicit and Explicit) at different granularity. For example, it is possible to define that two individual are equal (for example, *Taxi* and *Cab* are the same individuals of the concept *TravelMedium*) or to specify that due to subsumption relationship if the child concept has no matching individuals then the individual of the parent concept are potential replacement.

References

[1] Nickull, D., Kumar, R. & Mccabe,F., 2006. *Reference Model for Service Oriented Architecture 1.0.* OASIS Standard, 12 October 2006 [Internet] Available at: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html [accessed 22 March 2008]

[2] Lomow, G. & Newcomer, E., 2005. *Understanding SOA with Web Services.* 1st Ed. Boston: Addison-Wesley.

[3] Fremantle, P., Weerawarana, S. & Khalaf, R. 2002. Enterprise Services: Examining the Emerging Trends of Web services and How It is Integrated into Existing Enterprise Infrastructures. *Communications of the ACM*, 45(20), p.77-82.

[4] Kim, M.S. & Rosu, M.C., 2004. A Survey of Public Web Services, *5th International Conference, EC-Web 2004*, Zaragoza, Spain, August 31-September 3, 2004. Lecture Notes in Computer Science, Springer Verlag, p. 96-105.

[5] Amazon Web services, Amazon.com, [Internet] Available at: http://www.amazon.com/gp/browse.html?node=3435361 [accessed 10 April 2008]

[6] BEA, 2007. *Avis Budget Group Implements SOA on BEA WebLogic server.* BEA Systems, Inc. Press Release.

[7] Vaughan-Nichols, S., 2002. Web Services: Beyond the Hype, *IEEE Computer*, 35(2), pp. 18-21.

[8] W3C, 2002. *Charter on Web Services Architecture.* W3C Working Group, 22 July 2002 [Internet] Available at: http://www.w3.org/2002/01/ws-arch-charter [accessed 22 March 2008]

[9] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. & Nielsen, H.F., 2000. *Simple Object Access Protocol Version 1.2.* W3C Note, 8 May 2000 [Internet] Available at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ [accessed 18 September 2007]

[10] Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S., 2001. *Web Services Description Language Version 1.1.* W3C Note, 15 March 2001 [Internet] Available at: http://www.w3.org/TR/wsdl [accessed 18 September 2007]

[11] Bryan, D., Draluk, V., Ehnebuske, D., Glover, T., Hately, A., Husband, Y. L & Karp, A., 2003. *Universal Description Discovery and Integration (UDDI), Version 2.0,* Oasis Standard, 9 July 2002 [Internet] Available at: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2 [accessed 18 September 2007]

[12] Glatard, T., Emsellem, D. & Montagnat, J., 2006. Generic Web Service Wrapper for Efficient Embedding Of Legacy Codes in Service-Based Workflows. *The*

Grid-Enabling Legacy Applications and Supporting End Users Workshop (GELA'06). Paris, France, 20 June 2006. Springer Verlag, p. 44-53.

[13] Peltz, C., 2003. Web Services Orchestration and Choreography. *IEEE Computer,* 36 (10), p. 7-46.

[14] Castro-Leon, E., 2004. The web within the web, IEEE Spectrum, 41(2), p. 42-46.

[15] Berners-Lee, T., Hendler, J. & Lassila,O., 2001. *The Semantic Web,* Scientific American, May, 2001 [Internet] Available at: http://www.sciam.com/article.cfm?id=the-semantic-web, [accessed 10 July 2007]

[16] Klyne, G. & Carroll, J., 2004. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* W3C Recommendation, 10 February 2004 [Internet] Available at: http://www.w3.org/TR/rdf-concepts/ [accessed 15 May 2008]

[17] McGuinness, D. & Harmelen, F., 2004. *Web Ontology Language Overview.* W3C recommendation, 10 February, 2004 [Internet] Available at: http://www.w3.org/TR/owl-features/ [accessed 15 May 2008]

[18] Singhal, A., 2001. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24 (4), p. 35-43.

[19] Berendt, B., Hotho, A. & Stumme, G., 2002. Towards Semantic Web Mining, *Proceedings of the First International Semantic Web Conference on the Semantic Web*, Brisbane, Australia, June 09 - 12, 2002. Springer-Verlag, p. 264 - 278

[20] Sussna, M., 1993. Word Sense Disambiguation for Free-Text Indexing Using a Massive Semantic Network. *The Second International Conference on Information and Knowledge Management,* Washington, D.C., United States, ACM Press, p. 8-13.

[21] Martin, D., Paolucci, M., Mcilraith, S., Burnstein, M., Mcdermott, D., Mcguinness, D., Parsia, B., Payne, T. R., Sabou, M., Solanki, M., Srinivasan, N. & Sycara,K., 2004. *OWL-S: Semantic Mark-up for Web Services.* W3C Member Submission, 22 November 2004 [Internet] Available at: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ [accessed 15 May 2008]

[22] Mcilraith, S., Son, T. C., & Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web,* 16(2), p.46-53.

[23] Roman,D., Keller,U., Lausen,H., Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler., C. & Fensel, D., 2005. Web Service Modelling Ontology. *Applied Ontology,* 1 (1), p. 30-77.

[24] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A. & Verma,K., 2005. *Web Service Semantics - WSDL-S Version 1.0. , W3C Member Submission, 7 November 2005* [Internet] Available at: http://www.w3.org/Submission/WSDL-S/ [accessed 15 May 2008]

[25] White, P. & Grundy, J., 2003. Experiences Developing a Collaborative Travel Planning Application With .Net Web Services. *The 2003 International Conference on Web Services (ICWS).* Las Vegas, USA, June 23-26, 2003.CSREA Press, p.306-312.

[26] Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B. & Barros, A.P., 2003. Workflow Patterns. *Distributed and Parallel Databases,* 14 (3), p.45-5.

[27] Zisman, M.D., 1977. *Representation, Specification and Automation of Office Procedures.* Ph.D Dissertation, Working Paper 77-09-04., The Warthon School, University of Pennsylvania, Scrancton, 1977

[28] Leymann, F., Roller, D. & Schmidt, M-T., 2002. Web Services and Business Process Management. *IBM Systems Journal,* 41 (2), p.13-198.

[29] High, R., Kinder, S. & Graham., S., 2004. *IBM's SOA Foundation: an Architectural Introduction and Overview, V 1.0*, White Paper, IBM Corporation, December 8, 2005 [Internet] *available at: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/* [accessed November 10, 2007]

[30] Pasley, J. 2005. How BPEL and SOA Are Changing Web Services Development. *IEEE Internet Computing*, 9(3), p. 60-67.

[31] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I. & Weerawarana, S., 2003. *Business Process Execution Language for Web Services Version 1.1.* OASIS Specification, 5 May 2003 [Internet] available at: http://xml.coverpages.org/ni2003-04-16-a.html [accessed 20 September 2006]

[32] Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y. & Barreto,C., 2005. *Web Services Choreography Description Language Version 1.0.* W3C Candidate Recommendation, 9 November 2005 [Internet] Available at: http://www.w3.org/TR/ws-cdl-10/ [accessed 20 September 2006]

[33] Van Der Aalst, W.M.P., Dumas, M., Ter Hofstede, A.H.M. & Wohed, P., 2002. *Pattern-Based Analysis of BPML (and WSCI).* Brisbane: Queensland University of Technology.

[34] S. Thatte. 2001. XLANG: Web Services for Business Process Design. Microsoft.

[35] F. Leymann. 2001. Web Services Flow Language (WSFL 1.0). IBM.

[36] WSIF, 2005. *WSIF Apache Software Foundation Web Services Project.*

[37] Austin, D., Barbir, A., Peters, E. & Ross-Talbot, S., 2004. W*eb Services Choreography Requirements.* W3C Working Draft, 11 March 2004 [Internet] http://www.w3.org/TR/2004/WD-ws-chor-reqs-20040311/ [accessed 20 September 2006]

[38] Oracle, 2005. Oracle BPEL Process Manager, *Available at: http://www.oracle.com/technology/bpel/index.html*

[39] BizTalk, 2008. Microsoft BizTalk Server, *Available at: http://www.microsoft.com/biztalk/en/us/default.aspx*

[40] BPWS4J, 2005. IBM Run Time Environment for BPEL4WS1.1, *Available at: http://www.alphaworks.ibm.com/tech/bpws4j*

[41] Russell, S. & Norvig, P., 2003. *Artificial Intelligence: A Modern Approach.* 2nd Ed. New York: Prentice Hall.

[42] Wu, D., Parsia, B., Sirin, E., Hendler, J. & Nau,D., 2003. Automating DAML-S Web Services Composition Using SHOP2. *The 2nd International Semantic Web Conference (ISWC2003).* Florida, USA, October 21-23, 2003. IEEE computer society, p. 195-210.

[43] Kuter, U., Sirin, E., Nau, D., Parsia, B. & Hendler,J., 2005. Information Gathering During Planning For Web Service Composition. *Journal of Web Semantics,* 3 (2-3), p. 183-205.

[44] Sirin, E. & Parsia, B., 2004. Planning For Semantic Web Services. *In Semantic Web Services Workshop at the 3rd International Semantic Web Conference.* Florida, USA, November 8-10, 2004.

[45] Peer, J., 2004. A PDDL Based Tool for Automatic Web Service Composition. *Principles and Practice of Semantic Web Reasoning.* St. Malo, France, September 6-10, 2004. Springer, p. 149-163.

[46] Ghallab, M., Howe, A., Knoblock, C., Mcdermott, D., Ram, A., Veloso, M., Weld, D. & Wilkins, D., 1998. PDDL the Planning Domain Definition Language. *AIPS-98 Planning Committee*.

[47] Mcilraith, S. & Son, S., 2002. Adapting Golog for Composition of Semantic Web Services. *The 8th International Conference on Principles of Knowledge Representation and Reasoning.* Toulouse, France, April 20-22 2002. Morgan Kaufmann. p. 482-496.

[48] Levesque, H., Reiter, R., Lespérance, Y., Lin, F. & Scherl, R., 1997. Golog: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming,* 31(1), p.59-84.

[49] Srivastava, B. & Koehler, 2003. Web Service Composition – Current Solutions and Open Problems. *ICAPS'03 Workshop on Planning for Web Services.* 10 June, 2003. Trento, Italy.

[50] Matskin, M. & Rao, J., 2002. Value Added Web Services Composition Using Automatic Program Synthesis. *International Workshop on Web Services, E-Business, and the Semantic Web.* London, UK, June 16-17, 2002. Springer-Verlag, p. 213-224.

[51] Mihhail, M., & Tyugu, E., 2001. Structural Synthesis of Programs and Its Extensions. *Computing and Informatics Journal,* 20(1), p.1–25.

[52] Rao, J., Kungas, P., & Matskin,M., 2003. Application of Linear Logic to Web Service Composition. *The 1st International Conference on Web Services.* Las Vegas, USA, June 23-26, 2003. CSREA Press, p. 3-9.

[53] Nwana, H., 1996. Software Agents: An Overview. *Knowledge Engineering Review*, 11(3), p.1-40.

[54] Buhler, P.A. & Vidal, J.M., 2003. Semantic Web Services as Agent Behaviours. *Agentcities: Challenges in Open Agent Environments,* Springer- Verlag, p. 25-31.

[55] Knoblock, C., 2001. Mixed-Initiative Multi-Source Information Assistants. *The 10th International World Wide Web Conference.* Hong Kong, China, May 1-5 2001, ACM Press, p. 697-707.

[56] Richards, D., Van Splunter, S., Brazier, F. & Sabou, M., 2003. Composing Web Services Using an Agent Factory. *AAMAS Workshop on Web Services and Agent-Based Engineering.* Melbourne, Australia. p. 1-6.

[57] Hewitt, C., 1996. Viewing control structures as patterns of passing messages, *AI Memo* 410, MIT AI Laboratory, Cambridge, MA.

[58] Hammond, K., 1986. Learning to Anticipate and Avoid Planning Problems through the Explanation of Failures. *Fifth Conference on Artificial Intelligence, AAAI86.* Philadelphia, USA, August 11-15, 1986. Morgan Kaufmann, p. 556-560.

[59] Limthanmaphon, B. & Zhang, Y., 2003. Web Service Composition with Case-Based Reasoning. *The Fourteenth Australasian Database Conference on Database Technologies.* Adelaide, Australia, February, 2003. ACM International Conference Proceeding Series, p. 201 – 208.

[60] Diaz, O. G. F., Salgado, R. S., Moreno, I. S. & Ortiz, G.R., 2006. Searching and Selecting Web Services Using Case Based Reasoning. *Workshop on Ubiquitous Web Systems and Intelligence (UWSI 2006) In Conjunction with Computational Science and Its Applications (ICCSA 2006).* Glasgow, UK, May 8-11, 2006. Springer Berlin/ Heidelberg, p. 50-57.

[61] Porter, B. W. & Bareiss, R.E., 1986. Protos: An Experiment in Knowledge Acquisition for Heuristic Classification Tasks. *First International Meeting on Advances in Learning (IMAL86).* Les Arcs, France, p. 159-174.

[62] Madhusudan, T., Leon Zhao J. & Marshall, B., 2004. A Case-Based Reasoning Framework for Workflow Model Management. *Data & Knowledge Engineering,* 50 (1), p. 87-115.

[63] Cardoso, J. & Sheth, A. 2005. Adaptation and Workflow Management Systems. *International Conference WWW/Internet,* Lisbon, Portugal, 19-22 October, 2005. p. 356-364.

[64] Freuder, E., 1978. Synthesizing Constraint Expressions. *Communications ACM,* 21 (11), p. 958-966.

[65] Maruyama, D., Paik, I. & Shinozawa, M., 2006. A Flexible and Dynamic CSP Solver for Web Service Composition in the Semantic Web Environment. *Sixth IEEE International Conference on Computer and Information Technology.* Seoul, Korea, September 20-22, 2006. IEEE Computer Society, p. 43-43.

[66] Aggarwal, R., Verma, K., Miller, J. & Milnor, W., 2004. Constraint Driven Web Service Composition in Meteor-S. *IEEE International Conference on Services Computing*. Shanghai, China, September 15 - 18, 2004. IEEE Computer Society, p. 23-30.

[67] Mandell, D. & Mcilraith, S., 2003. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. *The 2nd International Semantic Web Conference (ISWC2003).* Sanibel Island, Florida, 20-23 October 2003. Lecture notes in computer science, Springer, p. 227-241.

[68] Traverso, P. & Pistore, I. 2004. Automated Composition Of Semantic Web Services Into Executable Processes. *Third International Semantic Web Conference (ISWC2004),* Hiroshima, Japan, November 9-11, 2004. Springer Berlin, p. 380-394.

[69] Milanovic, N. & Malek, I., 2004. Current Solutions for Web Service Composition. *IEEE Internet Computing,* 8 (6), p.51-59.

[70] McBride, B., 2002. Jena: A Semantic Web Toolkit. *IEEE Internet Computing,* 6 (6), p.55-59.

[71] Sirin, E., Parsia, B., Grau, B., Kalyanpur, A. & Katz,Y., 2007. Pellet: A Practical Owl-Dl Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web,* 5 (2), p. 51-53.

[72] Kolodner, J., 1993. *Case-Based Reasoning*. 1st Ed. San Mateo: Morgan Kaufmann.

[73] Schank, R. & Abelson, R., 1979. Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures. *The American Journal of Psychology,* 92 (1), p. 176-178.

[74] Aamodt, A. & Plaza, E., 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications,* 7 (1), p.39-59.

[75] Minsky, M., 1974. *A Framework for Representing Knowledge.* 2nd Ed. Cambridge, MA, USA: Massachusetts Institute of Technology.

[76] Rich, E. & Knight, K., 1992. *Artificial Intelligence.* 2nd Ed. New York: McGraw-Hill.

[77] Currency, 2003. Currency Ontology, Available at: http://www.daml.ecs.soton.ac.uk/ont/currency.owl

[78] Hennessey, D. & Hinkle, D., 1992. Applying Case-Based Reasoning to Autoclave Loading. *IEEE Expert,* 7(5), p.21-26.

[79] Remind 1992. *Developer's Reference Manual, Cognitive Systems.* Boston.

[80] Zhang, R., Budak, I. & Aleman-Meza, B., 2003. Automatic Composition of Semantic Web Services. *The International Conference on Web Services, ICWS '03*. June 23 - 26, 2003, Las Vegas, Nevada, USA. CSREA Press 2003, p. 38-41.

[81] Diaz-Agudo, B., Gonzalez-Calero, P, A. & Gomez-Martin, P, P., 2005. On Developing a Distributed CBR Framework through Semantic Web Services. *OWL Workshop.* Galway, Ireland, November 11-12, 2005.

[82] Agre, G., Atanasova, T. & Nern, H-J., 2005. Case-Based Semantic Web Service Designer and Composer. *Euromedia 2005.* Toulouse, France, April 11-13, 2005. Springer Verlag, p. 226-229.

[83] Zaremba, M., Vitvar, T. & Moran, M., 2007. Towards Optimized Data Fetching For Service Discovery. *The European Conference on Web service.* Halle, Germany, November 26-28, 2007. IEEE Computer Society, p. 191-200.

[84] Gurevich, Y., 2000. *Abstract State Machines: Theory and Applications*. 1st Ed. Berlin: Springer.

[85] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B & Dean, M., 2004. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, W3C Member Submission 21 May 2004 [Internet] available at: http://www.w3.org/Submission/SWRL/ [accessed 10 June 2007].

[86] Martin, D., Paolucci, M., McIlraith, S., Burnstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T. R., Sabou, M., Solanki, M., Srinivasan, N. & Sycara, K. 2004 Bringing Semantics to Web Services: The OWL-S Approach. *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, July 6-9, 2004. Kluwer Academic Publishers, p. 243-277

[87] Staab, S., 2003. Where Are The Rules? *IEEE Intelligent Systems,* 1 (1), p.76-83.

[88] OWL-SAPI, MINDSWAP Group, University Of Maryland, Institute for Advanced Computer Studies. [Internet] Available at: http://www.mindswap.org/2004/owl-s/api/ [accessed 14 May 2008]

[89] Watson, I., & Marir, F., 1994. Case-Based Reasoning: A Review. *The Knowledge Engineering Review,* 9 (4), p.355-381.

[90] Sqalli, M., Purvis, L., & Freuder, E., 1999. Survey of Applications Integrating Constraint Satisfaction and Case-Based Reasoning. *PACLP99: The First International Conference and Exhibition on the Practical Application of Constraint Technologies and Logic Programming.* London, UK, April 19-21, 1999. p.69-82.

[91] Purvis, L. & Pu, P., 1995. Adaptation using Constraint Satisfaction Techniques. *The First International Conference on Case-Based Reasoning Research and Development.* Springer, p. 88-97.

[92] Leake, D., Kinley, A. & Wilson, D., 2006. Learning to Improve Case Adaptation by Introspective Reasoning and CBR, *First International Conference (ICCBR-95).* Sesimbra, Portugal, October 23-26, 1995, Springer, p. 229-240.

[93] Koton, P., 1988. Reasoning About Evidence in Causal Explanations. *The 7th National Conference on Artificial Intelligence.* St. Paul, MN, USA, August 21-26, 1988, AAAI Press / the MIT Press, p. 256-263

[94] Codd, E.F., 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM,* 13 (6), p. 377-387.

[95] Kumar, V., 1992. Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine,* 13 (1), p.3-44.

[96] Baader, F. & Nutt, W., 2003. *Basic Description Logics.*. 1st ed. Cambridge: Cambridge University Press.

[97] Jang, M. & Joo-Chan, S., 2004. Bossam: An Extended Rule Engine for OWL Inferencing. *Third International Workshop of RuleML (RuleML 2004.* Hiroshima*,* 8-11 November 2004. Springer Berlin, p. 128-138.

[98] Cooper, W.S., 1997. *On Selecting a Measure of Retrieval Effectiveness.* 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[99] JUDDI, 2008. *JUDDI Private Registries*. Available At: http://ws.apache.org/juddi/

[100] Srinivasan, N., Paolucci, M. & Sycara, K., 2005. An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004).* 6-9 January, 2004, San Diego, California, USA. p. 96-110.

[101] Pan, Z., 2005. Benchmarking DL Reasoners Using Realistic Ontologies. *The workshop on OWL: Experiences and Directions.* Galway, Ireland, November 11-12, 2005.

[102] Gardiner, T., Horrocks, I. & Tsarkov, D., Automated Benchmarking of Description Logic Reasoners. *The 2006 International Workshop on Description Logics (DL2006)*. Windermere, Lake District, UK, May 30 - June 1.

[103] Mántaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Boi, F., Maher, M.L., Cox, M., Forbus, K.,Keane, M. &Watson, I., 2006. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review,* 20(3), p.215-240.

[104] Smyth, B. & Keane, M. T., 1995. Remembering to Forget: A Competence Preserving Case Deletion Policy for CBR Systems. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Canada. Morgan Kaufmann, p. 377-382.

[105] Mckenna, E., 1998. Modelling the competence of case-bases. Advances in Case-Based Reasoning: *Proceedings of the Fourth European Workshop on Case-Based Reasoning*. Dublin, Ireland, September 23-25, 1998. Springer-Verlag, p. 208 – 220.

[106] Thakker, D., Osman, T. & Al-Dabass, D., 2007. Semantic-Driven Matchmaking and Composition of Web services using Case-Based Reasoning. *The fifth IEEE European Web Services Conference (ECOWS 2007)*, Halle, Germany, November 26-28, 2007. IEEE Computer Society, p. 67-76.

[107] Osman, T., Thakker, D., Al-Dabass, D., Lazer, D. & Deleplanque, G., 2006. Semantic-Driven Matchmaking of Web services using Case-Based Reasoning.

The fourth IEEE International Conference on Web Services (ICWS 2006), September 18-22, 2006, Chicago, USA. IEEE Computer Society, p. 29-36

[108] Thakker, D., Osman, T. & Al-Dabass, D., 2006. S-CBR: Semantic Case Based Reasoner for Web services discovery and matchmaking. *The 20th European Conference on Modelling and Simulation (ECMS 2006),* Bonn, Germany, 28-31 May 2006. ECMS Press, p. 723-729.

[109] Flank, S., 1998. A layered approach to NLP-based information retrieval. *International Conference on Computational Linguistics.* San Francisco, California, 1998. Morgan Kaufmann Publishers, p.397-403.

[110] Liddy, E. D., 2000. Searching & search engines: When is current research going to lead to major progress? *The Year 2000 International Chemical Information Conference & Exhibition.* Annecy, France, 22-25 October 2000. Springer Verlag, p.109-114.

[111] Hanney, K. & Keane, M., 1996. Learning Adaptation Rules from a Case-Base, *Third European Workshop on Advances in Case-Based Reasoning.* Lausanne, Switzerland, November 14-16, 1996, Springer Verlag, p.179-192.

[112] d'Aquin, M., Badra, F., Lafronge, F. & Szathmary, L. 2007. Case base mining for adaptation knowledge acquisition. *The 20th International Joint Conference on Artificial Intelligence (IJCAI'07).* Hyderabad, India, January 6-12, 2007. Springer Verlag, p. 750-755.

[113] DUNS. *D-U-N-S ® number identifier system* [Internet] http://www.dnb.com/us/duns_update/index.html [accessed 10 May 2007]

[114] NAICS. *North American Industry Classification System* [Internet] http://www.census.gov/epcd/www/naics.html [accessed 10 May 2007]

[115] Foster, I., Kesselman, C., Nick, J. & Tuecke, S., 2002. Grid services for distributed system integration. *Computer, 35*(6), p.37-46.

[116] Foster, I., Kesselman, C., Nick, J. & Tuecke, S., 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum,* Edinburgh, Scotland, June 22, 2002.

[117] Yonggang, Q. & Frei, H., 1993. Concept based query expansion. *Annual ACM Conference on Research and Development in Information Retrieval,* Pittsburgh, Pennsylvania, United States, June 27 - July 1, 1993. ACM press, p.160 - 169.

Appendix A

Table 24 An example of a skeleton BPEL file

```
<process name="travelagency" targetNamespace="http://ntu.ac.uk/bpel/travelagency/"

xmlns:wizzair=http://travelagent.ntu.ac.uk/WizzAirFlightService
…
>

<partnerLinks>

<partnerLink name="WizzAirPL" partnerLinkType="wizzAir: WizzAirWSLink"
partnerRole="WizzAirWSProvider"/>
…

</partnerLinks>
<variables>
 <variable name="input" messageType="tns:travelagencyRequestMessage">
 <variable name="inputWizzAir" messageType="wizzair:getWizzAirFlightsRequest">
…
</variables>

<assign name="assign-deptdate">
  <copy>
   <from variable="input" part="payload"  query="/tns:FlightQuery/tns:departure-date">
    </from>
         <to variable="inputWizzair"  part="pQuery" query="/wizzair:FlightQuery/departure-date"/>
  </copy>
</assign>

<sequence name="RetrievePriceQuoteSequence">

        <invoke name partnerLink="WizzAirPL"                portType="wizzair:WizzAirPortType"
   operation="checkReservation" inputVariable="inputWizzAir"  outputVariable="outputWizzAir"/>

</sequence>
…
```

Table 25 A composition scheme with EasyJet Service

```
<process name="travelagency" targetNamespace="http://ntu.ac.uk/bpel/travelagency/"
xmlns:ejet=http://travelagent.ntu.ac.uk/
EasyJetFlightService
…
>

<partnerLinks>
<partnerLink name="EasyJetPL" partnerLinkType="ejet:EasyJetWSLink"
partnerRole="EasyJetWSProvider"/>
…
</partnerLinks>

<variables>
  <variable name="input" messageType="tns:travelagencyRequestMessage">
  <variable name="inputEasyJet" messageType="ejet:getEasyJetFlightsRequest"/>
…
</variables>
```

```
<assign name="assign-deptdate">
  <copy>
          <from variable="input" part="payload" query="/tns:FlightQuery/tns:departure-date">
   </from>
       <to variable="inputEasyJet" part="pQuery" query="/ejet:FlightQuery/departureFlightDate "/>
   </copy>
</assign>

<sequence name="RetrievePriceQuoteSequence">

            <invoke name partnerLink="EasyJetPL"            portType="ejet:EasyJetPortType"
                                        operation="checkReservation"
                        inputVariable="inputEasyJet" outputVariable="outputEasyJet"/>

</sequence>
```

List of Publications

Osman, T., Thakker, D. & Al-Dabass, D., Utilization of Case Based Reasoning for Semantic Web Services Composition, Submitted for the International Journal of Web and Grid Services (IJWGS).

Thakker, D., Osman, T., & Peytchev, E., Automated Run-time Composition of Web Services with Constraint Satisfaction, submitted for *the seventh IEEE International Semantic Web Conference (ISWC 2008)*,IEEE Computer Society, Karlsruhe, Germany, October 26-30, 2008.

Osman, T., Thakker, D. & Schaefer G., 2008. Semantic-based Expansion of Image Retrieval Queries. *Accepted for the 2nd International Language Resources for Content-Based Image Retrieval Workshop, The sixth international conference on Language Resources and Evaluation, LREC 2008*, Marrakech, Morocco, May 26, 2008, Springer Verlag.

Thakker, D., Osman, T. & Al-Dabass, D., 2008. Knowledge-Intensive Semantic Web services Composition. *The tenth IEEE conference on Computer Modelling and Simulation (UKSIM 2008)*, Cambridge, United Kingdom, April 1-3, 2008. IEEE Computer Society, p. 673-678.

Osman, T., Thakker, D. & Al-Dabass, D., Chapter XIII: Web Services Hybrid Dynamic Composition Models For Enterprise. *In the Book: Enterprise Architecture and Integration: Methods, Implementation and Technologies* Editors: Wing Lam and Venky Shankararaman, Publishers: Idea Group Inc, USA

Thakker, D., Osman, T. & Al-Dabass, D., 2007. Semantic-Driven Matchmaking and Composition of Web services using Case-Based Reasoning. *The fifth IEEE European Web Services Conference (ECOWS 2007)*, Halle, Germany, November 26-28, 2007. IEEE Computer Society pp. 67-76.

Osman, T., Thakker, D., Schaefer G. & Lakin, P., 2007. An Integrative Semantic Framework for Image Annotation and Retrieval. *The 2007 IEEE/ACM conference on web intelligence*, Silicon Valley, USA, November 11-13, 2007, IEEE Computer Society, p. 366-373.

Osman, T., Thakker, D., Schaefer, G., Leroy, M. & Fournier, A., 2007. Semantic Annotation and Retrieval of Image Collections. *The 21st European Conference on Modelling and Simulation*, Prague, Czech Republic, June 4-6, 2007.

Osman,T., Thakker, D., Yang, Y. & Claramunt, C., 2006. Semantic Spatial Web Services with Case-Based Reasoning. T*he 6th International Symposium on Web and Wireless Geographical Information Systems (W2GIS 2006)*, Hong Kong, China, 4-5 December 2006. Springer Verlag Lecture Notes in Computer Science, p.247-258.

Osman, T., Thakker, D., Al-Dabass, D., Lazer, D. & Deleplanque, G., 2006. Semantic-Driven Matchmaking of Web services using Case-Based Reasoning. *The fourth IEEE International Conference on Web Services (ICWS 2006)*, September 18-22, 2006, Chicago, USA. IEEE Computer Society, p. 29-36.

Thakker, D., Osman, T., & Al-Dabass, D., 2006. S-CBR: Semantic Case Based Reasoner for Web services discovery and matchmaking. *The 20th European Conference on Modelling and Simulation (ECMS 2006),* Bonn, Germany, 28-31 May 2006. ECMS Press, p. 723-729.

Thakker, D., Osman, T., & Al-Dabass, D., Semantics based automatic assignment in Web services composition. *The 9th International Conference on Computer Modelling and Simulation Conference*, 4-6 April, 2006, Oxford, UK. p. 67-72.

Thakker, D., Osman, T. & Al-Dabass, D., Web Services Hybrid Dynamic Composition Models for Ubiquitous Computing Networks. *IEEE 8th Int. Conf. on Advanced Communication Technology*, Phoenix Park, Korea, 22-24 Feb 2006. IEEE Computer Society, p. 274-280.

Osman, T., Thakker, D. & Al-Dabass, D., 2005. Bridging the Gap between Workflow and Semantic-based Web services Composition. *The Workshop on WWW Service Composition with Semantic Web Services 2005 (wscomps05), the 2005 IEEE/WIC/ACM International Joint Conference on Web Intelligence (WI 2005) and Intelligent Agent Technology (IAT 2005)*, Compiègne, France, September 19, 2005. IEEE Computer Society, p. 13-23.

Thakker, D., Osman, T. & Al-Dabass, D., 2005. Web services Composition: A Pragmatic View of the present and the future. The 19th European Conference on Modelling and Simulation, Riga, Latvia, June 1-4, 2005. p. 826-832.

Thakker, D., Osman, T. & Al-Dabass, D., 2005. Private UDDI registry models for B2B Web Services. *The Eighth United Kingdom Simulation Society (UKSIM) Conference,* Oxford, UK, 6-8 April, 2005. p. 13-16.