UNIVERSITY OF Southampton

University of Southampton Research Repository
ePrints Soton

Copyright © and Moral Rights for this thesis are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given e.g.

AUTHOR (year of submission) "Full thesis title", University of Southampton, name of the University School or Department, PhD Thesis, pagination

http://eprints.soton.ac.uk

UNIVERSITY OF SOUTHAMPTON

Faculty of Physical Sciences and Engineering

Electronics and Computer Science
Web and Internet Science

EXPRESS: Resource-Oriented and RESTful Semantic Web Services
by

Areeb Alowisheq

Thesis for the degree of Doctor of Philosophy

17 October 2014

ABSTRACT

This thesis investigates an approach that simplifies the development of Semantic Web services (SWS) by removing the need for additional semantic descriptions.

The most actively researched approaches to Semantic Web services introduce explicit semantic descriptions of services that are in addition to the existing semantic descriptions of the service domains. This increases their complexity and design overhead. The need for semantically describing the services in such approaches stems from their foundations in service-oriented computing, i.e. the extension of already existing service descriptions. This thesis demonstrates that adopting a resource-oriented approach based on REST will, in contrast to service-oriented approaches, eliminate the need for explicit semantic service descriptions and service vocabularies. This reduces the development efforts while retaining the significant functional capabilities.

The approach proposed in this thesis, called EXPRESS (Expressing RESTful Semantic Services), utilises the similarities between REST and the Semantic Web, such as resource realisation, self-describing representations, and uniform interfaces. The semantics of a service is elicited from a resource's semantic description in the domain ontology and the semantics of the uniform interface, hence eliminating the need for additional semantic descriptions. Moreover, stub-generation is a by-product of the mapping between entities in the domain ontology and resources.

EXPRESS was developed to test the feasibility of eliminating explicit service descriptions and service vocabularies or ontologies, to explore the restrictions placed on domain ontologies as a result, to investigate the impact on the semantic quality of the description, and explore the benefits and costs to developers. To achieve this, an online demonstrator that allows users to generate stubs has been developed. In addition, a matchmaking experiment was conducted to show that the descriptions of the services are comparable to OWL-S in terms of their ability to be discovered, while improving the efficiency of discovery. Finally, an expert review was undertaken which provided evidence of EXPRESS's simplicity and practicality when developing SWS from scratch.

Table of Contents

List of tables

List of figures

DECLARATION OF AUTHORSHIP

I, AREEB ALOWISHEQ

declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

EXPRESS: RESTful and Resource-Oriented Semantic Web Services

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:
    1. Alowisheq, Areeb, Millard, David and Tiropanis, Thanassis (2011). Resource-Oriented Modelling: Describing Restful Web services Using Collaboration Diagrams. In, The 8th International Joint Conference on e-Business and Telecommunications, Seville, Spain, 18 - 21 Jul 2011.
    2. Alowisheq, Areeb and Millard, David (2009) EXPRESS: EXPressing REstful Semantic Services. In, 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, Doctoral Workshop, Milan, Italy, 15 - 18 Sep 2009. , 453-456.
    3. Alowisheq, Areeb, Millard, David and Tiropanis, Thanassis (2009) EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. In, 8th International Semantic Web Conference (ISWC 2009), Doctoral Consortium, Chantilly, VA, USA, 25 - 29 Oct 2009. Springer Berlin/Heidelberg, 941-948.

Signed: .....

Date: 17.10.2014

Acknowledgements

I would like to thank the following people for their support:

My family: Muneera Alwohaiby and Dr Abdullah Alowaisheq, my husband Hassan Alowairdhi, my sisters and brothers for their overwhelming love, support and guidance your help was absolutely critical for completing the PhD, and of course My children Faisal and Deema for your love and patience.

My supervisor: Dr David Millard for your guidance, assistance, patience, persistence, and continuous encouragement. I have learnt so much from your research skills, integrity and research ethic. It was a privilege working with you.

My advisor Dr Thanassis Tiropanis, for your valuable comments and encouragement, and my internal examiner Dr Nicholas Gibbins for your constructive feedback and support.

My friends who supported me during the PhD: Nada Albunni, Fatimah Akeel, Nora Al Rajebah, Nora Alothman, Alaa Mashat, Dr Adolfo Ruiz-Calleja, Dr Kathryn Bradbury, Dr Reena Pau, Dr Xin Wang, Dr Ilaria Licardi, Dalal Alazizy, Dr Heba Kurdi, Dr Sarah AlHumoud, and Dr Hend Al-Khalifa.

Members of the WAIS lab, especially Dr Yvonne Howard for her support and vital role in making the lab a positive and welcoming environment, and Dr Charlie Hargood and Rikki Prince for their valuable advice.

Dr Kevin Page, who took the time to look at my work and provide me with useful insights, and Dr Hugh Glaser for the enthusiastic discussions and encouragement.

The experts who participated in the expert review, for their time and insightful comments.

I would also like to acknowledge the Saudi Cultural Bureau in the UK and the Imam Muhammad bin Saud University for the PhD scholarship and their support during this period, and the Royal Academy of Engineering fro the travel grant to present my work at the WI-IAT 2009 conference in Italy.

Definitions and Abbreviations

endpoint a URI, which is an entry point to a service or resource, to expose them on the Web; it should be registered at the Web server for it to be available.

service ontology/vocabulary: a data model that defines concepts and properties for describing services.

semantic service description: a semantic description of a service instance that uses concepts and properties defined in service ontologies or vocabularies.

domain ontology: Is a data model that captures valid knowledge for a specific domain.

service-oriented/resource-oriented meta model: a model either a vocabulary, ontology or conceptualisation of an interface as services/resources.

RESTful Web services: also referred to as Web APIs, these are web services that expose endpoints to resources, which respond to HTTP requests and in practice may not adhere to all of REST's constraints.

client: The term client has been used in this thesis to refer to a service consumer.

server: The term server has been used in this thesis to refer to a service provider.

Resource-Oriented Modelling: A modelling approach which focuses on modelling resources in an interface and their static relationships and dynamic interactions.

EXPRESS	EXPressing REstful Semantic Services
REST	REpresentational State Transfer
SWS	Semantic Web Services
SPARQL	SPARQL Protocol and RDF Query Language
RDF	Resource Description Framework
HTTP	Hypertext Transfer Protocol
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services (OWL Services)

Chapter 1: Introduction

The advancement of software, hardware and networking has caused distributed systems to evolve since the times of the ARPANET email application in the 1960s. Distributed systems have gone from 1-tier architectures, to n-tier architectures, built with middleware to accommodate the heterogeneity of underlying systems and enable them to work together.

The emergence of the Web had a great impact on the way which distributed systems were built. The distributed systems community was influenced by its success, but instead of viewing the Web as a distributed system in itself, it was viewed as a convenient transport mechanism: Web servers were widely available, and easy to set up, and hence created a broad common layer through which middleware could be tunneled together with a global unique addressing system offered by URI. Another lesson the distributed community learnt from the Web was the communicative power of text-based markup languages, which could overcome the heterogeneity problems in exchanged messages.

As a result of this view of the Web, Web services emerged, wrapping the functionality offered by existing solutions in XML-based descriptions. These Web services are the XML-based parallels of their middleware predecessors, and are heavily influenced by Remote Procedure Call (RPC) (Birrell and Nelson, 1984). For example, The WSDL (Christensen *et al.*, 2001) service description contains a similar type of information offered by earlier Interface Definition Languages (IDLs) i.e. the types of inputs and outputs of the service and how to invoke it. Moreover the concept of a service directory has been mirrored by the Universal Description Discovery and Integration (UDDI) in Web services.

Another result of this view was implicitly passing down the design objectives of RPC to Web services, which aimed to ensure that a remote procedure should run as if it was a local one. This design objective aimed to relieve programmers from the burden of dealing with the complexities of the network and to maintain the

reliability of the distributed system (Birrell and Nelson, 1984). This idea of hiding remoteness, was one of the reasons the Web alone was overlooked as a successful mechanism for providing services, it was lossy, stateless, and was unable to accommodate the requirements of legacy systems built on the expectation of reliable middleware. As a result, the development of Web services continued to aim towards overcoming the unreliability of the Web and providing richer descriptions for the services to automate or semi-automate their discovery and invocation processes.

The request for richer descriptions was because the Web Service Discovery Language (WSDL) standard provided syntactic descriptions of services. Offering syntactic descriptions, however, is insufficient for the automation or semi-automation of service discovery and composition, for example, stating that a service accepts an integer and returns a string will not offer information on what the service does, especially on a Web scale.

The Semantic Web is a set of technologies enabling the semantic description of resources using standards such as Resource Description Framework (RDF) and Web Ontology Language (OWL), hence providing machines with the ability to infer more information about what a resource represents. Thus, the Semantic Web offers a solution to the lack of semantics in the Web services world. The Semantic Web services research community has introduced several approaches for Web service semantic descriptions. These range from lightweight solutions like SAWSDL (Farrell and Lausen, 2007) to complex ones like OWL-S (Martin *et al.*, 2004) and WSMO (Bruijn *et al.*, 2005a). The complexity of these latter approaches stems from their heavy reliance on logical reasoning for the automation of discovery, matchmaking and composition. This complexity also means it is very challenging for these features to be available at Web scale (Klusch, 2008b; Fensel and van Harmelen, 2007; Hench *et al.*, 2008). There is a trade-off between automation and scalability, and existing Semantic Web service approaches tend to focus on automation. However, recently there has been a rising interest in lightweight Semantic Web services, for reasons of scalability and minimising complexity and design overhead.

Another issue with these approaches, whether heavy or lightweight, is that they require semantic service descriptions, therefore necessitating service ontologies or vocabularies. This requirement of service descriptions stems from the RPC mindset these approaches are based on. This was the prevalent mindset in traditional Web services when SWS research began. However, there was an increased realisation that the WSDL-based services were not gaining the

popularity anticipated, and that, for the reasons discussed above, they could not scale the way the Web has scaled.

As a result, another approach, RESTful Web services, was put forward. This approach is based on an understanding of the properties that make the Web scale well, and attempts to offer the functionality of Web services through the manipulation of Web resources; consequently these Web services do not have service descriptions. REST (Fielding, 2000) is an architectural style for network-based systems. It provides a set of constraints learnt from the Web's HTTP development and when applied can make systems scalable, reliable, reusable, resilient and provide other desirable features of the Web as a network-based system. The constraints of REST are: identification of resources, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine of application state. Although REST was not introduced as an approach to designing Web services, it has been adopted by the majority of developers as an alternative to WSDL/SOAP. Although not always adhering to all of REST's constraints (Fielding, 2007; Richardson and Ruby, 2007; Vinoski, 2008a), RESTful Web services are gaining popularity and are adopted by major service providers like Google, Amazon and Yahoo. The popularity of RESTful Web services comes from their being light-weight (with no added layers of specification), accessible, resource-oriented, and declarative (Zhao and Doshi, 2009).

This research focuses on developing an approach to provide RESTful Semantic Web services, with the aim of reducing the complexity involved in developing Semantic Web services. It does so by exploiting similarities between REST and the Semantic Web, such as resource-realization, self-describing representations, and uniform interfaces.

1.1 Motivation and Approach

As discussed above, the influence of RPC resulted in Web services having service descriptions, and consequently this has influenced Semantic Web service approaches. More specifically, this is to have semantic descriptions for both the service itself (semantic descriptions and vocabularies/ontologies) and the resources the service interacts with (domain ontologies). This overhead is not without consequences. Bachlechner and Fink (2008) surveyed and analysed opinions from both practitioners and researchers about the potential of Semantic Web services as integration architectures. According to their results one of main challenges that SWS face is that they are perceived as highly complex, and it is not clear how the research vision can be grounded into reality.

The objective of this research is to simplify the development of SWS, by eliminating the need for semantic service descriptions and vocabularies, through an approach called EXPRESS (Alowisheq and Millard; Alowisheq *et al.*, 2009). EXPRESS uses ontologies that describe classes, instances and relationships among them to create and describe resources accessible via RESTful interfaces. Figure 1 shows how EXPRESS aims to simplify providing SWS, by contrasting components required in existing methods to the ones required in EXPRESS.

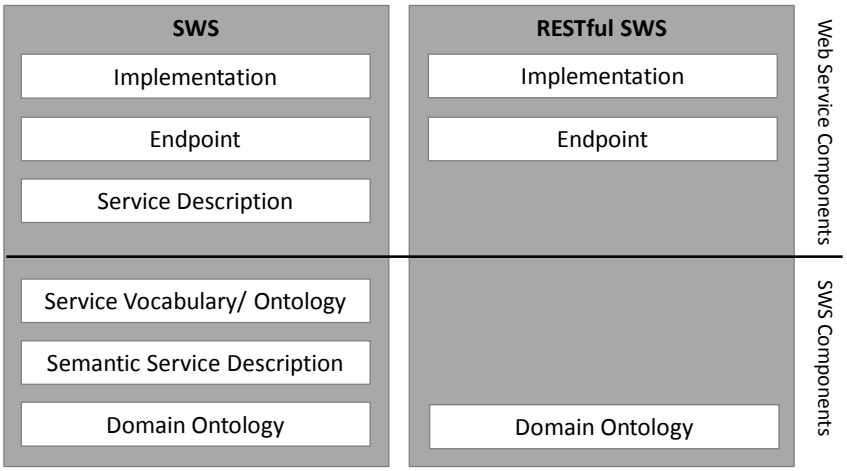


Figure 1 Components of Web services and SWS

A description of these components is provided below:

1. Implementation: this component encompasses the business logic, and its functionality is to respond to service requests and manipulate them, by dealing with the internal system components.

2. Endpoint: This is a URI, and its purpose is to expose the service on the Web, it should be registered at the Web server for it to be available.

3. Service Description: This is the XML-based service description (usually in WSDL but can be in other formats) this description exposes the types of inputs and outputs and the endpoint.

4. Service Ontology/Vocabulary: An ontology/vocabulary defining concepts and properties for describing services.

5. Semantic Service Description: Mechanisms to describe various aspects of the service instance semantically, using the semantic service ontology mentioned above, such as the services' inputs, outputs, preconditions, and effects.

6. Domain Ontology: This provides a semantic description of the resources referenced in the Service Description.

In EXPRESS stub-generation becomes a by-product of the mapping between entities in the domain ontology and resources; therefore, by providing a domain ontology describing the resources, endpoints can be automatically created as a result of the mapping.

1.2 Hypothesis and Research Questions

The research hypothesis is as follows:

Utilising the semantics in the domain ontology and REST can provide a RESTful SWS approach that (1) eliminates service ontologies/vocabularies and explicit descriptions of interfaces, and (2) generates semantic descriptions as a by-product of its provision, and this can simplify the development of SWS while preserving a similar level of semantic expressivity as existing SWS approaches.

"semantic expressivity" refers to the degree to which the exposed semantic descriptions offer automated discovery and composition.

"simplify" means it reduces development effort and increases development speed.

EXPRESS is the RESTful SWS approach devised and evaluated in this thesis. The above hypothesis is tested by answering the following research questions:

1. Is it possible to eliminate explicit service descriptions and service ontologies/vocabularies while their semantic descriptions become a by-product of their provision?
2. Does it simplify the process of providing SWS services?
3. Can it provide a similar level of semantic expressivity to existing approaches, and what are the trade-offs in terms of practicality?

Figure 2 illustrates how the hypothesis and research questions relate to research activities, which is discussed further in the next section: Research Methodology.

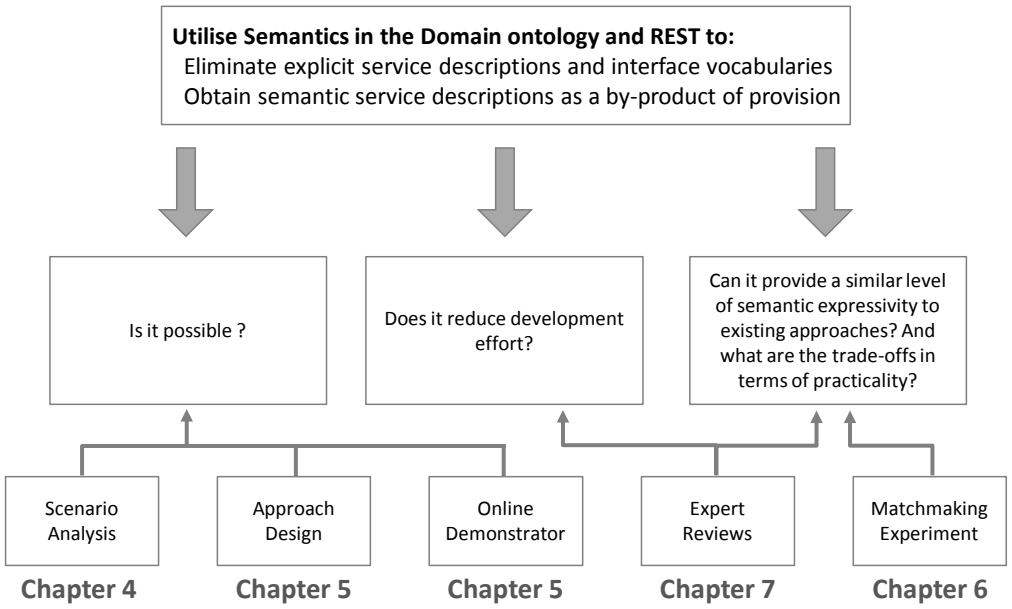


Figure 2 Hypothesis, research questions and research activities

1.3 Research Methodology

This section explains how the research questions were addressed by the research activities.

Question one asks whether it is possible to have a RESTful SWS approach that: (1) eliminates service ontologies/vocabularies and explicit interface descriptions and (2) generates semantic descriptions as a by-product of its provision.

Three research activities were undertaken to answer this question.

Both the scenario analysis and approach design answer the first part of the question, which is whether it is possible to eliminate service ontologies/vocabularies and explicit interface descriptions.

The scenario analysis involved analysing the requirements of 20 Web service scenarios from a resource-oriented perspective; this analysis results in identifying interaction requirements that need to be addressed when utilising the domain ontology and HTTP for semantically describing the services in those scenarios.

The approach design builds on the interaction requirements identified in the scenario analysis and shows how those requirements can be fulfilled in EXPRESS, the RESTful SWS approach proposed in this thesis.

With regard to the second part of question one, whether is it possible to have a RESTful SWS approach that generates semantic descriptions as by-product of its

provision, the online demonstrator for EXPRESS shows how, by semi-automatically generating interface stubs from the domain ontology, they become semantically described.

Question two, which asks if EXPRESS reduces the development effort, is addressed by the expert review, where experts in Semantic Web technologies assess EXPRESS and compare it to two other SWS approaches: OWL-S (Martin *et al.*, 2004) and RESTdesc (Verborgh *et al.*, 2011).

Question three addresses the level of semantic expressivity in EXPRESS, and the trade-offs in terms of practicality. The expert review mentioned above addresses both aspects. In addition, the matchmaker experiment compares the discoverability of EXPRESS to OWL-S services by running the same matchmaker algorithm on two service test collections, one in EXPRESS and the other in OWL-S and compares the performance of the matchmaker in terms of speed and accuracy.

1.4 Contributions

The work described in this thesis has a number of specific contributions that will be of value to the Semantic Web service research community:

1. The description of an approach called EXPRESS, for offering Semantic RESTful Web services from domain ontologies, which embodies this approach of eliminating service descriptions and interface vocabularies, and an online demonstrator of an EXPRESS deployment engine that shows how the semantic descriptions are a result of the service provision.
2. An analysis of 20 real scenarios in five Web service communities of interest, resulting in the identification of interaction requirements that guide the design of EXPRESS.
3. A Resource-Oriented Modelling approach based on UML collaboration diagrams.
4. A mapping between EXPRESSive descriptions and OWL-S descriptions.
5. The evaluation of EXPRESS in both a matchmaker experiment, which required the creation of an EXPRESSive service test collection (EXPRESS-TC) and the adaptation of a semantic matchmaker, and in an expert review, in which experts were asked to compare EXPRESS to two other SWS approaches in terms of development effort and practicality.

1.5 Thesis Structure

The thesis contains eight chapters which are summarised in this section.

This first chapter presented the motivation of this thesis, the hypothesis it examines, the research questions and the methodology to answer them, and the contributions.

Chapter 2 provides a background to the technologies and concepts that influence the design of RESTful Semantic Web services. These are: middleware, the Web, Web services, REST and the Semantic Web. It explains how Web services and Semantic Web services were heavily influenced by earlier middleware approaches, and how this influence led to adding extra layers of descriptions and treating the Web as merely a transport layer for Web services. It also highlights the distinguishing features in the Web, REST and the Semantic Web, which are: abstracting distributed components as resources, not services, assigning them URIs, and linking them together.

Chapter 3 discusses a total of 27 SWS approaches, which were either service or resource-oriented, and the variations in their description means: whether they introduced interface ontologies or vocabularies or introduced service descriptions as extension mechanisms. Chapter 3 also discusses the research strategies conducted to evaluate the viability of these approaches. It concludes by establishing the research strategy for this thesis. Figure 2, above, illustrates how chapters 4, 5, 6 and 7 fit into answering the research hypothesis.

Chapter 4 addresses the following two questions: if the resources are semantically described in domain ontologies, what other aspects are required to be expressed in an interface, so that the client can interact with the interface to fulfil a specific scenario, and how can these be achieved using only REST and the domain ontology? It presents the compilation and analysis of a total of twenty representative Web service scenarios from five communities of interest. Interaction requirements which emerged from the analysis are used inform the design of the proposed RESTful SWS approach, EXPRESS.

Chapter 5 introduces EXPRESS, the RESTful SWS approach proposed by the thesis. It provides an overview and shows how the interaction requirements identified in Chapter 4 are achieved. It also presents a proof-of-concept demonstrator for EXPRESS that shows how RESTful Services can be provided semi-automatically.

Chapter 6 assesses the discoverability of EXPRESSive descriptions, using a standardised test-collection and evaluation environment. It discusses how service

matchmaking works in EXPRESS, the methodology for evaluation and the results of the matchmaking experiment.

Chapter 7 discusses the expert review experiment, its methodology and results. In the expert review, six experts were interviewed about EXPRESS as a Semantic Web service approach, and how it compares to two other approaches: OWL-S and RESTdesc.

Chapter 8 concludes the thesis. It discusses the overall results and conclusions in the light of the hypothesis, and suggests future research directions.

Chapter 2: Background: Web Services, Representational State Transfer and the Semantic Web

This chapter provides an overview of the technologies and concepts influencing the design of RESTful Semantic Web services. It starts by providing an overview of the Web then Web services and explains the effect of earlier middleware technologies on their design, it then explains REST, its relationship with the Web and how it has influenced the development of RESTful Web services. It also discusses relevant Semantic Web technologies, and how Semantic Web services emerged.

2.1 The World Wide Web (WWW)

The WWW was created at CERN by Tim Berners-Lee and Robert Cailliau in 1989. It originally aimed to enable physicists to record and share data, results and news. It was created as a distributed hypertext (text containing links to other text) system, and Berners-Lee's vision of the Web was heavily influenced by hypertext pioneers such as Bush (1945), Engelbart (1963) and Nelson (1980).

There already existed successful hypertext systems with more complex hypertext capabilities than the Web offered; however the Web's focus on being distributed over Wide Area Networks, rather than offering complex hypertext constructs (Berners-Lee *et al.*, 1992) turned out to be the key factor in its massive success.

Berners-Lee, with other collaborators, wrote proposals, protocols and developed the first Web server and browser. This started in 1989, and by 1992 it grew beyond CERN and expanded globally. This required formally written standards, governed by standards organisations such as the Internet Engineering Task Force (IETF), and later by W3C. Three main standards govern the Web, and have

contributed to its massive success: URI, HTML and HTTP, and these are explained next.

2.1.1 Uniform Resource Identifier (URI)

URI provides a universal naming mechanism for resources on the Web, and other application layer protocols. However, it is mainly associated with the Web. It is used for locating and linking documents and resources. Other than its universality, the importance of the URI was its compactness. One string—the URI—combines the protocol used to access the resource (usually HTTP, but it accommodates others), the host where the resource resides, the name of the resource itself, and query strings and fragments (Kozierok, 2005). Berners-Lee authored the first URI standard RFC 1630 in 1994, published by the IETF (Berners-Lee, 1994). The URI standard went through several refinements. RFC 3986 is the current standard, published in 2005, co-authored by Roy Fielding, who coordinated the community refinement efforts (Berners-Lee *et al.*, 2005).

2.1.2 Hypertext Markup Language (HTML)

The second important standard, HTML (Raggett *et al.*, 1999), governs the format of the content, and defines constructs for linking to resources. HTML is a subset (profile) of Standard Generalised Markup Language (SGML). SGML is an ISO standard, originally designed to share machine-readable documents in industry and government (ISO 8879:1986). Web browsers interpret the HTML document to display a formatted page, and also GUI elements that a user can interact with, such as links and forms. When a user submits a form or follows a link, the browser uses the appropriate HTTP method to contact the server.

2.1.3 Hypertext Transfer Protocol (HTTP)

HTTP is a TCP/IP application layer protocol. It has evolved since it was first defined by Tim Berners-Lee in 1991 (Berners-Lee, 1991): this original version was known as HTTP/0.9. It was designed to be very simple; it was only intended for document transfer and it had only one method, GET. In 1996, HTTP/1.0 (Fielding *et al.*, 1996), RFC 1945, was introduced, which discussed headers, intermediaries, media types, caching, status codes and two more methods HEAD and POST, but it had been in use for several years prior to that publication. This version was very successful; however, it suffered from some limitations: 1) did not support multiple URLs for the same IP, as the hostname was not required as part of the message, 2) each HTTP session handled one client request, which increased traffic

unnecessarily. 3) Limited support for caching and proxying affected performance. In 1997, RFC 2068 HTTP/1.1 (Fielding *et al.*, 1997) was introduced and later enhanced and republished in 1999 as RFC 2616 HTTP/1.1 (Fielding *et al.*, 1999). HTTP/1.1 resolved the issues with HTTP/1.0, so it enhanced caching and proxying mechanisms, supported multiple host names, enabled the retrieval of partial resources, supported persistent connections and added content negotiation. HTTP/1.1 also introduced new methods: PUT, DELETE, OPTIONS and TRACE.

2.2 Web Services

This section provides an overview of Web services and their origins in middleware technologies and explains the influences of middleware concepts on how these services were designed.

2.2.1 The Origins of Web Services

Ever since the ARPANET email application in the 1960s, distributed systems have evolved from one-tier systems (on a single machine), to two-tier systems (client and server), to three-tier (client–middleware–server) systems. The motivation behind this development has been to generalise the mechanism of remote interaction, not only for specific application types, such as email servers, or file servers, but also for any application through middleware in three-tier architectures.

The term "middleware" in computer science literature was popularised by Bernstein (1996) in a CACM article (Emmerich *et al.*, 2007). Middleware evolved as a response to the increasing demand for distributed systems, It provided programming paradigms to facilitate the development of software components capable of remote interaction.

Middleware plays two main roles in distributed systems (Alonso *et al.*, 2004):

1. *As programming abstractions*

   To simplify the development process, middleware masks the complexities of the underlying networks and protocols behind programming abstractions, for example, procedures, messages, objects, services and resources, hence enabling developers to concentrate on application-specific problems. The more useful the abstraction is, the more likely it is to be adopted.

2. *As infrastructure*

> Those abstractions hide complex implementations provided by the middleware infrastructure. The infrastructure provides both development support, for example stub-generation, compilation and deployment, and run-time support, such as interacting with network layers and marshalling and translating messages.

RPC (Birrell and Nelson, 1984) was the first key middleware abstraction (Emmerich *et al.*, 2007). The main purpose of RPC was "to make distributed computing easy". The principal idea was to enable developers to invoke procedures on remote hosts in a similar fashion to invoking local ones. RPC aimed to deal with both the distribution and the heterogeneity in different systems. Clients and servers in an RPC system interact through corresponding stubs; the stubs deal with synchronisation, serialisation, data mapping and network communication. By having the procedure's interface (signature) defined in the form of Interface Definition Language (IDL), IDL compilers can then generate the stubs automatically. IDLs were introduced to overcome differences in programming languages and machine architecture.

Another noteworthy aspect of RPC was "dynamic binding", where a directory and name server binds a client call with a service that matches the signature, hence providing further decoupling between clients and servers.

Most middleware platforms were enhancements or extensions of RPC: they were either built on top of RPC platforms (Alonso *et al.*, 2004, p.44), or highly influenced by the RPC paradigm (Emmerich *et al.*, 2007). Object brokers demonstrated this dependency by extending RPC to facilitate the development of distributed object-oriented applications. Object Brokers were a response to the shift towards object-orientation. Object methods replaced the role of procedures in RPC. Specifications such as Common Object Request Broker Architecture (CORBA) (Object Management Group, 1995) were established. CORBA allowed brokers to expose object interfaces and provide access to them and to common services that provide the functionality, such as concurrency, querying, naming, licensing etc., needed by most objects (Alonso *et al.*, 2004, p.54). A main issue with CORBA is the incompatibility between different implementations. This is mainly to do with overly complex and sometimes conflicting specifications (Henning, 2006).

Although only RPC and CORBA are explained here, there are other extensively deployed middleware paradigms, such as Transaction Process Monitors, Message Brokers and Workflow Management Systems, all of which have been used in Enterprise Application Integration (EAI). EAI aims to solve issues with integrating

heterogeneous systems within one organisation. Nevertheless, middleware platforms were expensive and unnecessarily complex, and did not provide an adequate solution for business-to-business (B2B) demands (Alonso *et al.*, 2004, p.128). Unlike EAI, B2B integrates multiple organisations, which means integrating over the Internet, rather than through LANs, hence adding more complexities and scalability issues. Because there are different organisations to integrate, this also means that they needed to support heterogeneous middleware platforms (Alonso *et al.*, 2004, p.128).

2.2.2 Web Service Standards

Originally the World Wide Web (WWW) emerged as a massively distributed system for sharing documents. But these documents do not have to be static, they can be dynamically generated according to the client's actions. Technologies such as the Common Gateway Interface (CGI) and server-side scripting emerged to support the creation of dynamic websites, which expose and enable communication with a server's application logic through a Hypertext Markup Language (HTML) presentation layer.

As a result of these advances, the WWW became a promising platform for B2B, because it meant, unlike in RPC, RMI or other middleware protocols, integration could happen by exchanging dynamically generated documents, which can pass through firewalls. This led to considerable efforts in two directions:

1. The creation of application servers that encapsulate several middleware technologies, making them accessible to Web applications.
2. Standardising the format of exchanged documents.

Extensible Markup Language (XML) (Bray *et al.*, 2008) played a huge role in format standardisation: it was both human-legible and machine-processable and provided a standard way of structuring data and documents. Like HTML, XML is also a profile of Standard Generalised Markup Language (SGML). The standardisation of XML in 1998 (Bray *et al.*, 1998), and its simple syntax, made it well-supported, as it led to the development of a plethora of parsers and validators.

WWW Consortium (W3C) discussions for XML protocols for distributed applications began in 1999. In 2000 SOAP (discussed in the next section), a protocol for exchanging structured information, became an acknowledged W3C submission. In 2001, WSDL (discussed in section 2.2.1.2), a protocol for describing services also became an acknowledged submission. These two protocols form the basic protocols for Web services. A third, less popular, specification, is Universal

Description Discovery and Integration (UDDI), designed to facilitate the discovery of Web services (Bellwood *et al.*, 2002).

According to the W3C Web Services Architecture Working Group, a Web service is: "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 serialisation in conjunction with other Web-related standards." (Booth *et al.*, 2004)

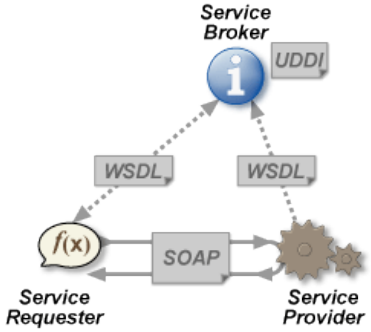


Figure 3 Web Services Architecture[1]

There are three entities involved in the Web service usage scenario: the service provider, the service requester, and the service registry. The service provider publishes a description of the service to a registry (publishing stage), a developer (on the client side) then looks for a desired service in that registry (finding stage). The developer gets the service description, constructs the messages accordingly, and then binds to the service (binding stage).

The aim of Web services is to provide well-defined descriptions for underlying components, and offer them a Web interface. These services can then be discovered, invoked and composed to perform a workflow of tasks.

2.2.2.1 Simple Object Access Protocol (SOAP)

One of the main Web service Technologies is SOAP (Box *et al.*, 2000) which provides a mechanism for representing a service call and its response in XML. The word "Object" in the acronym indicates the influence of the Object-Oriented paradigm at that time.

Box (2001) (the co-author of the SOAP specification) explains that what motivated SOAP was the need to design a protocol for exchanging messages over the Internet, and to design an XML serialisation format for those messages. They

[1] Web Services Architecture, Wikipedia, https://en.wikipedia.org/wiki/File:Webservices.png

reviewed several RPC protocols and serialisation formats and aimed to satisfy the majority of cases targeted by those specifications. Box also emphasises that much of the effort at the beginning was to overcome the lack of a typing mechanism in XML; however the focus shifted to integrating the XML schema, once it became standardised.

SOAP defines messages as envelopes containing a header and a body. Originally, SOAP was designed to work over the Hypertext Transfer Protocol (HTTP). However, in version 1.1, it was improved so that it could be used in other transport protocols. Version 1.2 clarifies and extends version 1.1 for protocol binding and XML encoding.

The SOAP specification was written for the following purposes:

1. Standardising a message structure in XML: an envelope, containing a header and a body, each of which could have multiple blocks.
2. Standardising how to structure an RPC request containing the variables and method name, and its response in XML, containing the results. In addition to sending messages as RPC, SOAP offers the option to exchange documents.
3. Defining the rules for processing the messages: how different entities have different roles, and the elements the entities must understand, and actions to take if they do not.
4. Describing SOAP bindings to HTTP and SMTP, and a generic binding framework to other protocols.
5. Defining how to encode data in XML, this led to the design of the SOAP data model.

The SOAP data model aims to represent data as object graphs. The SOAP encoding defines the serialisation of the SOAP model into XML. The definition of this model took up a substantial proportion of the effort invested in designing SOAP and increased its complexities. This was because the XML Schema at that time was far from standardised. Box, the co-author of SOAP explains: “SOAP's original intent was fairly modest: to codify how to send transient XML documents to trigger operations or responses on remote hosts. Because of our timing, we were forced to tackle issues that the Schemas WG [Working Group] has since solved, which caused the ‘S’ in SOAP to be somewhat lost.” (Box, 2001).

2.2.2.2 Web Service Description Language (WSDL)

WSDL (Christensen *et al.*, 2001) is an XML language used to describe Web service interfaces. It plays for SOAP services the same role as IDL for RPC and other

middleware platforms. A WSDL document describes the XML types of inputs and outputs of the service. A WSDL 1.1 document is structured as follows:

1. Types: this part of the WSDL file defines the exchanged data types in the XML schema.
2. Messages: this defines the structure of exchanged messages, and designates a message part for each parameter.
3. Operation: this defines the inputs and outputs of a service, and its message exchange pattern, which can be any of: one-way, request-response, solicit-response and notification.
4. Port type: this defines the port type or interface groups in the operations offered by the Web service.
5. Binding: this specifies the SOAP binding the RPC or document and the transport protocol.
6. Service: this contains the actual ports, with their corresponding URIs; however, these are usually available at the same address.

Even though WSDL 2.0 became a W3C recommendation in 2007 (Moreau *et al.*, 2007), WSDL 1.1 is still more popular and has more tool support. One of the objectives of the WSDL 2.0 model was to better support RESTful Web service descriptions, these are explained further in section 2.3.

2.2.3 Service-Oriented Architecture

The emergence of Web services popularised the vision of Service-Oriented Architecture (SOA). SOA can be defined as: "A software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls." (Natis, 2003). SOA takes a unified view of both Enterprise Application Integration (EAI) and Business-to-Business (B2B), where systems in organisations can integrate internally in a similar fashion to integrating with other organisations externally.

According to Erl (2008), SOA principles are: standardised service contracts, loose coupling, abstraction, reusability, statelessness, autonomy, discoverability, composability, and service-orientation and interoperability.

Because of Web services' standardisation and their seamless use of the WWW as a transport medium, they became a basic component of SOA. The vision of SOA was to have loosely-coupled reusable services, and dynamically build applications from them, thus enabling integration across enterprises. This vision drove the

research behind Semantic Web services, which will be discussed in more detail in Chapter 3.

Although the WSDL/SOAP approach to Web services has become a widely accepted standard and significantly reduced coupling compared to CORBA, RPC and other middleware technologies, the RESTful approach based on the Web architecture where resources are key actors, as discussed next, reduces coupling more, scales further, and takes full advantage of the Web architecture.

2.3 REST Representational State Transfer (REST)

2.3.1 Origins

Fielding, in his PhD dissertation, introduced the REST architecture style (Fielding, 2000). It aimed to realise and sustain the architectural aspects that made the Web—the HTTP protocol—succeed as a scalable network-based hypermedia system. Fielding was an author of the Web standards such as HTTP and URI, and in his dissertation he discussed the REST constraints on a system. These are: it is client-server, stateless and enables caching; it has a uniform interface, is layered and enables code on demand. The uniform interface constraint is further explained by the following constraints: identification of resources, unified semantics for resource access methods, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine of the application state. The client-server constraint makes the system scalable, portable and decoupled. The statelessness constraint means that a request from the client must contain all the information needed to process this request; this enables simpler replication of the server, and hence more scalability. It also increases the reliability of the system. The cache constraint increases the efficiency and scalability. The layering constraint increases modularity, reusability, scalability and resilience. Code on demand is an optional constraint which simplifies client implementation.

Moreover, Vinoski (2008b) explains how the uniform interface constraints maximise reuse. Because resources have a uniform interface, client applications are simplified: there is no need to code them for customised interfaces. Error handling becomes uniform. The server guides the client throughout the interaction, hence maximising the decoupling. It also simplifies the adding of intermediaries, increasing the modularity and scalability. The system design becomes simpler and extensible, which decreases the number of defects. The uniform interface constraint of having the application state controlled by

hypertext transitions provides a standard method for interaction and enables further decoupling between the server and the client. Hence REST was never intended as a Web service architecture; instead, it was a set of constraints on network-based systems "specifically targeted at distributed information systems" (Fielding, 2000, p.100).

2.3.2 Resource-Oriented Architecture

REST's potential as an architectural style for Web services was identified by Mark Baker and Paul Prescod (Fielding, 2007), who advocated it as alternative to the SOAP approach (Prescod, 2002). Developers welcomed the RESTful Web service approach. They saw it as a natural fit for the Web: it provided a simple, uniform interface and did not impose additional layers, as did WSDL/SOAP. Many service providers, such as Google, Yahoo and Amazon, started offering RESTful Web services. The increasing popularity of RESTful Web services[2] is based on many factors: they are lightweight, provide easy accessibility, and are resource-oriented, making them declarative (Zhao and Doshi, 2009).

This rapid uptake came at a cost: RESTful Web services were not always RESTful. This was because of the misconception that as long as HTTP methods were used, then the Web service was inherently RESTful.

The so-called RESTful Web services violate two REST constraints mentioned above: the uniform interface and statelessness. The other constraints are maintained because they are embedded in the HTTP servers' architecture and do not require implementation. Conversely, the uniform interface and statelessness required implementation for each Web service. An example of violating the uniform interface is the use of the HTTP method GET for updates; this in fact should have a read-only effect. Violating the statelessness constraint is by having the server store client-specific information, which should be stored on the client and sent when needed to the server.

These violations happened because there was no authoritative reference for designing RESTful Web services. Fielding's dissertation was an abstract explanation of the REST constraints and rarely provided examples for existing scenarios or technologies. The need for a guide on how to design RESTful Web services was met by Richardson and Ruby's book 'RESTful Web Services' (Richardson and Ruby, 2007). This book, which provides practical examples and highlights common mistakes, is considered an authoritative resource among the

[2] According to the Programmable Web, on 16.7.2013, 69% of Web Service APIs are RESTful http://www.programmableweb.com/apis.

REST community. The authors, however, focus on Resource-Oriented Architecture (ROA), which is an architecture that adheres to REST constraints and provides a concrete set of rules for designing resources and using HTTP methods.

The main idea in ROA is for the server to identify the resources in the Web service, provide a uniform interface to those resources—a set of actions—through which a client can create, read, update and delete the resources. These actions are mapped respectively to the HTTP methods POST, GET, PUT and DELETE, taking into account the HTTP constraints on these methods: GET is read-only and GET, PUT and DELETE are idempotent. ROA also emphasises the use of the standard HTTP error messages. They introduced a design method for developing Web services. Its steps are as follows (Richardson and Ruby, 2007):

1. Identify the data set; and
2. Map the data into resources.

Then, for each type of resource:

3. Specify the URIs;
4. Expose a subset of the interface (establishing which HTTP methods can be performed on the resource—these methods are GET, PUT, POST and DELETE);
5. Design the representations sent and accepted to and from the client, and decide on the media types;
6. Integrate the resources into existing resources using hyperlinks and forms;
7. Consider the typical course of events; and
8. Consider error handling.

These steps assume that resources have types, just before step 3, as it stated "for each type of resource". Therefore in most cases when designing an interface, the developed endpoint URIs represent resource types, not individual resources, the individual resource URIs are created dynamically. The conventions of having resource types and methods that are applied to them, have their roots in object-oriented (OO) design, this is an expected consequence considering the object-oriented influences on the design of HTTP[3], as the abstract of HTTP/1.0 states that HTTP is:

[3] This view is also held by other influential members of the W3C such as Dan Connolly
"Distributed objects are the very heart of the Web, and have been since its invention. HTTP was design as a distributed realization of the Objective C (originally Smalltalk) message passing infrastructure: the first few bytes of every HTTP message are a method name: GET or POST. Uniform Resource Locator is just the result of squeezing the term object reference through the IETF standardization process." Connolly, D. (1997). *A draft of the editorial of the Mar/Apr 1997 issue of Web Apps Magazine* [Online]. Available: http://www.w3.org/People/Connolly/9703-web-apps-essay.html [Accessed 12/12/2013].

"a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands)."

This OO influence has been passed on to the design of RESTful Web APIs, not only because of its aforementioned influence on the design of HTTP, but also because of the underlying OO programming languages and frameworks used to develop those APIs such as Java, PHP, .NET, etc. and also the legacy applications they are providing an interface for. As a result several parallels between ROA and OO exist, as both approaches model the world as entities manipulated by methods, they both have the notion of factories, and typed entities.

2.3.3 REST vs. ROA

The differences between REST and ROA can be summarised as follows:
1. REST is a set of architectural constraints, and a study of how they can be applied to HTTP development; ROA is an architecture based on REST that uses HTTP for developing Web services.

2. REST describes the requirements of the uniform interface, but it does not restrict it to a set of methods. However, in ROA, the main effort lies in designing the uniform interface by identifying resources, giving them URIs and deciding which HTTP methods can be performed on them.

3. Although ROA required the use of hyperlinks to guide a client's state, Fielding (Fielding, 2007; 2008a) criticises ROA for not focusing on the hypermedia constraint. This constraint means the use of media types to specify not only the representations of the resources, but to specify also hypermedia controls that denote what actions can be performed. As an example in HTML, from the anchor element `<a>`, the client knows it can perform a `GET`, also from `<form>` the client performs a `GET` or `POST`. Another example from the Atom Publishing Protocol (APP) (Gregorio and de hOra, 2007) is the way it uses `rel="edit"` to specify entries that are editable; hence atom clients know, from the media type, these entries accept PUT and DELETE.

Fielding (2008b) explained the reason media type design was ignored in ROA:
"To some extent, people get REST wrong because I failed to include enough detail on media type design within my dissertation. That's because I ran out of time, not because I thought it was any less important than the other aspects of REST."

However even though ROA Web APIs are not entirely RESTful, they are extremely popular. There are benefits from adhering to the Web Architecture, or parts of it,

as Richardson, the co-author of the RESTful Web services book, argues in his RESTful Maturity Model (Richardson, 2008), in which he elaborates on the use of media types.

The maturity model focuses on the use of three elements: resources, HTTP verbs and hypermedia, and defines four levels (0-3), to grade the API according to the REST constraints.

Level 0: HTTP Tunnelling
An example would be SOAP, which is usually sent over HTTP using a POST method. It does not utilise any properties of the transfer protocol. Interaction usually happens through a single endpoint (URI); even though there may be several services, the individual services are accessed using a different addressing mechanism, SOAP ports, for example.

Level 1: Resources
When resources are given different URIs, a URI is an endpoint to interacting with the resource. However, in this level, only one HTTP method is used, regardless of the semantics of the interaction.

Level 2: HTTP Verbs (Methods)
At this level HTTP methods with correct semantics should be used, GET for read only operations (safe), DELETE and PUT should be idempotent, and POST is for non-safe and non-idempotent operations. In addition, the use of the correct HTTP response codes is required.

Level 3: Hypermedia Controls
The Web services at this level adhere to the 'hypermedia as the engine of the application state' constraint. This means the responses are designed to contain hypermedia controls that tell the client what actions can be taken next. These hypermedia controls can be either from ATOM (Nottingham and Sayre, 2005) or defined in a new application-specific media type.

Moreover, although typically only APP (Gregorio and de hOra, 2007) and its media type ATOM are acknowledged to have reached this maturity level, recent publications such as Allamaraju (2010) and Webber *et al.* (2010) have enabled developers to understand the hypermedia constraint. Nevertheless, debates exist in the REST community on what media types to use. Opinion is divided between the use of generic media types, such as APP or customised media types for specific applications.

2.3.4 Comparison to SOA

As discussed previously, the main influence for WSDL/SOAP services was RPC. For RESTful Web services, the main influence was the Web architecture, and in particular the HTTP protocol. These influences were clearly manifest in the ways interfaces were conceptualised and abstracted: in the contrast between services and resources, and in the introduction of a machine-readable description layer in WSDL/SOAP services.

A description of the interface usually includes the address of the service, how to invoke it, and the structure and format of exchanged messages. WSDL descriptions of SOAP services state the address using the elements of port, binding and operation; the latter specifies the name of the actual operation to invoke (one endpoint can have more than one operation). The type and message elements specify how messages are structured, and in SOAP 1.1 a service was always invoked by sending an HTTP POST request with a SOAP message to the endpoint (SOAP 1.2 supported HTTP GET).

On the other hand, for RESTful Web services developed in practice, the interface descriptions are written as text in HTML pages to be read by developers. The descriptions state the endpoints' URIs, the HTTP method, and the structure and media type of the accepted messages. The HTTP methods invoked on those URIs could be any of the four HTTP methods. Although there are specifications such as WADL (Hadley, 2009) and WSDL 2.0 that provide machine-readable descriptions for RESTful Web services (as WSDL does for SOAP), because RESTful Web services have simpler interfaces, these specifications not nearly as essential as WSDL is for SOAP (Richardson and Ruby, 2007).

On this basic level of comparison, RESTful Web services are simpler than WSDL/SOAP ones for providing a programmable interface. They have no description layer, and interacting with them is very simple; for GET requests, only a web browser is needed, and for other requests, an HTTP client library is sufficient.

One of the benefits of WSDL descriptions is for tools that automatically create client stubs to interact with the SOAP services. As noted above, interacting with RESTful Web services is very simple in comparison, which eliminates the need for this automation. On the contrary, in many cases the automatically created code introduces unnecessary complexity (compared to RESTful Web services), which is

supposed to be hidden by those tools; however, when there is a need to debug the code, this complexity is amplified.

SOAP and WSDL were designed to provide versatility. For example, although SOAP typically uses HTTP as its transport protocol, it can also use other protocols such as the Simple Mail Transfer Protocol (SMTP) (Klensin, 2001). SOAP was designed so that intermediaries could process the messages and forward them. This is what Pautasso *et al.* (2008) referred to 'as freedom of choice' in WSDL/SOAP compared to 'freedom from choice' in RESTful Web services. The 'freedom of choice' mindset in the WSDL/SOAP approach is evident in the body of Web service specifications built on top of them, which are typically referred to as 'WS-*'. These where developed to address the vision of SOA (Section 2.2.3), where integration in EAI and B2B can be achieved using the same technologies and approaches, so more standards and specifications needed to be developed to address the requirements of these complex domains, such as support for security, reliability, transactions and other Quality of Service (QoS) requirements. For example, specifications such as WS-Addressing (Gudgin *et al.*, 2006) and WS-Security (Nadalin *et al.*, 2006) were developed to offer advanced features: for example, WS-Addressing is designed so that addresses can be embedded in SOAP messages. It also enables the specification of 'from' and 'reply-to' addresses. WS-Security and its related specifications provide end-to-end security, unlike in HTTP, where security is limited to the transport level; moreover, it enables the sender to encrypt part or all of the message body. There are many other WS specifications that, while they add features, nevertheless introduce further complexity.

Critics of REST argue that it does not offer the tool support and Quality of Service (QoS) options needed for enterprise application scenarios and that it is better suited to *ad hoc* integration over the Web (Pautasso *et al.*, 2008). This is because Web services standards were driven by vendors like IBM and Microsoft, building for the SOA vision, whereas REST supporters tend to be independent developers, arguing for simpler and less vendor-specific standards. However, REST has become the focus of increased interest and initiatives that offer QoS, for example Webber *et al.* (2010) discussed RESTful alternatives for providing security, reliability and transactions. Moreover there have been REST composition initiatives, such as specifying Business Process Execution Language (BPEL) for REST (Pautasso, 2009).

2.4 The Semantic Web

Tim Berners-Lee's vision for the Semantic Web was to provide a machine-comprehensible Web, a Web of Data where the data is expressed in a form that enables intelligent reasoning (Berners-Lee, 1998).

Representing machine-comprehensible data, where systems can infer meaning, was studied and implemented as knowledge-representation systems by artificial intelligence researchers years before the Web was developed. These systems were centralised, requiring users to share the same concepts, but it meant that the inferences the system made were accurate. Moreover, these systems limited the questions that could be asked to questions they could answer. The Semantic Web sacrifices the accuracy and reliability of knowledge-representation systems for the sake of interoperability, openness and decentralisation, in the same way that the Web sacrificed the accuracy and reliability of hypertext systems for the same reasons (Berners-Lee *et al.*, 2001).

The Semantic Web is based on four fundamental principles (Allemang and Hendler, 2011):

1. Anyone can say Anything about Any topic (the "AAA" slogan).
2. Open World Assumption (OWA), meaning that the absence of information does not mean it does not exist; there is always more information that could be known, this is in contrast to the Closed World Assumption (CWA), typically applied in databases and hence more intuitive, where absence of information means that information does not exist.
3. Nonunique naming: the same entity could be known by more than one name.
4. The network effect, where the more people join the Semantic Web, the more valuable it becomes.

To achieve the Semantic Web vision, languages such as Resource Description Framework (RDF) and Web Ontology Language (OWL) describing resources and relationships between resources were developed. The Semantic Web layer stack, illustrated below, illustrates how these technologies fit with Web technologies such as XML and URI.

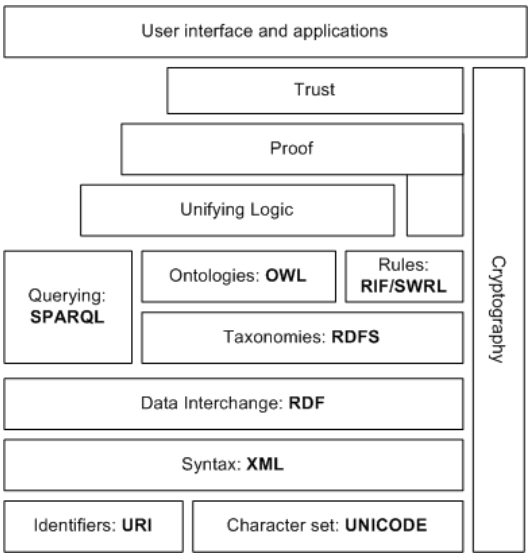


Figure 4 Semantic Web Layer Cake [4]

2.4.1 Resource Description Framework (RDF)

RDF (Beckett and McBride, 2004) models data as assertions about resources. Each assertion is a triple, in the following form: subject-predicate-object. A collection of RDF triples represents a labelled directed multi-graph, where subjects and objects are nodes and a predicate is a link from subject to object. RDF is designed as triples to enable logical reasoning,

RDF identifies subjects, predicates (properties) and objects using URIs. A new concept or relation can be defined easily by giving it a URI on the Web, hence the "AAA" slogan. Originally, RDF was specified as "a foundation for processing metadata", as stated in the first RDF W3C working draft (Lassila and Swick, 1997). However, the RDF data model described above proved successful in representing data as well. RDF is serialised in XML. RDF/XML is the standard syntax, but it has other popular serialisations, such Notation 3 (N3) (Berners-Lee *et al.*, 2008), which is more compact and readable than RDF/XML.

The RDF Schema language (RDFS) (Brickley and Guha, 2004) complements RDF, it is an approach to describe RDF vocabularies using RDF. It defines a vocabulary for describing vocabularies. In RDF there are no mechanisms to define a class (type) of resources, nor information about properties, such as which types of resources are described by a property, and what is the type of values of these properties. Therefore RDF Schema extends RDF so that these types of descriptions are possible, hence enabling a logical reasoner to infer additional information from

[4] Semantic Web Stack, Wikipedia http://en.wikipedia.org/wiki/Semantic_Web_Stack

the data. Ontologies are another Semantic Web mechanism for describing vocabularies, the Web Ontology Language (OWL) is discussed next.

2.4.2 Web Ontology Language (OWL)

OWL is a language for representing ontologies on the Web. It provides more expressive formalisms than RDF Schema, and hence more inferences. Ontologies emerged from research on modelling the domain of interests in the design of knowledge-based systems. They are used to conceptualise domains and share this conceptualisation. Ontologies occupy much of the research on the Semantic Web: for example, research areas include ontology design, engineering, evolution, management, reasoning, and alignment.

OWL is the standard language for ontologies on the Semantic Web. It is based on Description Logic (DL). DLs are formal knowledge representation languages, and their levels of expressivity vary. Baader (2003) provides a good overview of DL.

OWL ontologies have the following components: individuals (instances), properties and classes. A property has a domain and range. Properties can be either object properties (link to other individuals) or data properties (have literal values). Properties in OWL can be functional, inverse, transitive and symmetric. OWL enables complex class expressions. Classes can be defined using set operators, constraints on properties (cardinality, range, value), and universal and existential restrictions. Reasoning over ontologies can answer questions such as: Which class does an instance belong to? Is it possible to satisfy the constraints in the ontology (is it consistent)? And what are the subclasses and super-classes of a given class?

There are two main specifications of OWL, both are W3C recommendations: OWL 1.0 in 2004 (McGuinness and Harmelen, 2004) and OWL 2 in 2009 (Hitzler *et al.*, 2012). OWL 1.0 has three sublanguages:

1. OWL Lite: The least expressive language of the three, does not support the use of some modelling constructs or restricts their use; it aimed to simplify the implementation of supporting tools.
2. OWL DL: More expressive than OWL Lite, and computationally complete and decidable.
3. OWL Full: The most expressive of the three: it uses the same modelling constructs as OWL DL. However OWL Full does not restrict the way they are used; as a consequence, there are no computational guarantees.

OWL 2 is fully backward compatible with OWL 1.0, but is more expressive. For example, it enables the definition of keys, chained properties, and meta-modelling. OWL 2 has three sublanguages (profiles), which target efficiency for different application scenarios:

1. OWL EL: For applications that have ontologies with a large number of classes and properties, reasoning can be performed in a polynomial time with respect to the size of the ontology.
2. OWL QL: For efficient query answering in applications that have large volumes of instance data.
3. OWL RL: Restricts modelling constructs, so the language resembles an OWL-based rule language, aimed at applications that require scaled reasoning.

2.4.3 SPARQL Protocol and Query Language (SPARQL)

The SPARQL specification (Prud'Hommeaux and Seaborne, 2008) is a widely adopted W3C recommendation that defines a query language for RDF datasets, and a protocol for accessing SPARQL endpoints. Queries in SPARQL contain a graph pattern (a set of triples containing variables) and when processed, matching RDF graphs are returned from the dataset. New RDF graphs can be created using the keyword CONSTRUCT; this can be used to transform the structure of retrieved data. Update queries have been added to the specification, enabling the modification of the underlying datasets using INSERT and DELETE queries. This extension was proposed in 2009 (Schenk and Gearon, 2009), and became a W3C recommendation 2013 (Gearon *et al.*, 2013).

2.5 Linked Data

The Semantic Web community realised that the Web of Data, also referred to as Linked Data, had to be specifically created to expedite the emergence and spreading of the Semantic Web vision. The term 'Linked Data' was coined in Tim Berners-Lee's Design note in 2006 (Berners-Lee, 2006). It states four rules for publishing Linked Data:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things.

These rules show the movement from the earlier Semantic Web perspective on URIs as only identifiers, to using them as means for resource representation retrievals. The note also specifies a 5-star rating system for Linked *Open* Data. The rating system promotes publishing that is open-licensed, using open W3C standards (RDF and SPARQL), and linking the data to other published datasets. Recent statistics show that the overall number of triples published as Linked Data is 61,976,332,795[5] and is rapidly increasing. This trend is proving stronger with the publishing of government datasets in both the UK and the USA.

2.5.1 Publishing Linked Data

This section overviews some issues with publishing Linked Data, which are explained further in (Bizer *et al.*, 2009) and (Heath and Bizer, 2011).

1. Minting URIs

This involves selecting the structure of the URI to represent classes properties and individuals, and should follow guidelines for making them stable and simple (Sauermann *et al.*, 2008; Heath and Bizer, 2011).

2. Choosing RDF Vocabularies.

In describing the dataset well-known vocabularies should be used, and where new vocabularies are defined, then these should be mapped to other vocabularies.

3. Linking

This involves linking resources in the published dataset to other Linked Data datasets.

4. Metadata

Mechanisms have been introduced to describe datasets and how they are linked to other datasets, such as vocabulary of interlinked Datasets (voiD) (Alexander *et al.*, 2009) and the Co-reference Resolution Service (CRS) (Glaser *et al.*, 2009).

5. Publishing Tools

These can be classified as tools that serve the contents of RDF stores as Linked Data, and tools that provide a Linked Data view to legacy data (Bizer *et al.*, 2009).

2.5.2 Linked Data Applications

Bizer *et al.* (2009) classified Linked Data applications into: browsers, such as Tabulator (Berners-Lee *et al.*, 2006), search engines, such as Falcons (Cheng and Qu, 2009) and Sindice (Tummarello *et al.*, 2007), and domain specific applications:

[5] LODStats, 10 April 2014, http://stats.lod2.eu/

these harvest the data and the links to address complex informational requirements.

Recently there has been interest in the relationship between Linked Data and Web services, and in particular Semantic Web services: Pedrinaci *et al.* (2010a) present two views of their relationship—one is that the increase of semantic data on the Web presents a very promising environment for annotating Semantic Web services and publishing those annotations. The second is that complex services can be built to produce and consume Linked Data; the capabilities of these services go beyond data integration to cause real world effects.

A RESTful perspective to the relationship between Linked Data and Web services is realised in the recent W3C "Linked Enterprise Data Patterns Workshop", and the resulting member submission "Linked Data Basic Profile 1.0" (Nally *et al.*, 2012a), where conventions have been proposed to update Linked Data RESTfully. This submission reflects the increasing interest from both research and enterprise communities in the rapid growth in the size of published Linked Data. These conventions in the submission set out a set of standard patterns, design choices, and best practices to help developers when designing a Linked Data architecture (Nally *et al.*, 2012b).

2.6 Semantic Web Services

As discussed earlier in this chapter, WSDL provides syntactic-level descriptions for the services. Syntactic descriptions are insufficient for the automation or semi-automation of service discovery and composition. For example, stating that a service accepts an integer and returns a string will not offer information on what the service does, especially on a Web-scale.

The Semantic Web services vision (McIlraith *et al.*, 2001; Ankolekar *et al.*, 2001) utilises Semantic Web technologies to achieve automatic discovery, invocation, composition and execution of Web services. The approach is to augment or mark up (McIlraith *et al.*, 2001) Web services with semantic descriptions that can be interpreted and reasoned about by semantic-aware clients.

According to Cabral *et al.* (2004), Semantic Web service requirements can be categorised into three dimensions: activities, architecture and service ontology.

1. Activities define the functional requirements expected from SWS infrastructures. These are: publishing, discovery, selection, composition, invocation, deployment and ontology management.

2. Architecture is the set of components to achieve the activities mentioned above. These components include: a register, a reasoner, a matchmaker, a decomposer and an invoker.
3. Service ontology can differ amongst approaches and involves Inputs; Outputs; Pre-conditions: the necessary state of the world for executing the service; Post-conditions: the state of the world after executing the service successfully; Cost; Category; Atomic service; and composite service: whether the service can be described as a composition of atomic services.

The first dimension, activities, and the third, Service ontology, are the most significant, because activities define the requirements the SWS are expected to achieve, regardless of any architectural components used to achieve them, and the Service ontology addresses the elements typically contained in the semantic service description.

The activities are explained briefly below:

1. Publishing

Publishing is concerned with advertising the services' capability. It assumes there is a registry where these service descriptions are published. The concept of service registries can be traced back to RPC (directories), then UDDI.

2. Discovery

This means the discovery of services matching a given query. In the case of Semantic Web services the matching depends on service's semantic descriptions, which involve name, input, output, preconditions and postconditions. The selection activity is concerned with choosing between two or more matching services, based on other criteria, such as cost or category.

3. Composition or choreography

This is concerned with the automatic or semi-automatic composition of larger services from other services, and the control of how that composition is executed.

4. Invocation

This activity happens after the service is discovered and selected. It is concerned with the actual invocation of the service, like preparing inputs and dealing with exceptions.

5. Deployment

Cabral *et al.* (2004) assume that the deployment of a Web service is independent of the publishing of its semantic description. However, there can be mechanisms for instance deployment.

6. Ontology management

Traditional Semantic Web services rely heavily on ontologies for both the domain and the service description. This requires management of those ontologies in terms of upgrading, maintenance, and accessibility.

Semantic Web service approaches will be surveyed in Chapter 3.

2.7 Summary

This chapter discussed approaches to achieve interoperability in distributed systems, and the influences of the Web and later the Semantic Web in developing Web services and Semantic Web services as solutions for distributed interoperable systems.

It explained how Web services and Semantic Web services, were also heavily influenced by earlier middleware approaches. For example, RPC provided IDL descriptions for procedures, Web services provided WSDL descriptions for services, and Semantic Web services augmented and annotated those service descriptions further. Another result of this influence is that the Web is merely a transport layer for Web services.

This chapter explained that abstracting distributed components as resources, not services, assigning them URIs, and linking them together, has been the distinguishing feature in the Web, REST and the Semantic Web.

Chapter 3: Approaches to Semantic Web Services

Chapter 2 discussed how traditional Web services were heavily influenced by earlier middleware approaches, and how that resulted in a divergence from how the Web works. It also discussed REST and how resource-oriented and RESTful Web services emerged. The different semantic technologies used in the service descriptions were also explained in Chapter 2, together with the functionalities and goals of SWS.

In this chapter several approaches for implementing Semantic Web services are reviewed. Sections 3.2 and 3.3 discuss a total of twenty-seven SWS approaches. These sections show how the differences between how traditional Web services and RESTful ones are conceptualised have led to interesting variations in how they are semantically described. Section 3.2 includes approaches that are service-oriented, whereas section 3.3 covers those that are resource-oriented. Section 3.4 further classifies these approaches according to whether they introduce service or resource ontologies/vocabularies or extension mechanisms. Section 3.5 compares the approaches according to the capabilities they offer.

SWS approaches are considered emergent, and have not been adopted outside their research communities (Wilkinson *et al.*, 2009). Nevertheless they differ in their maturity; some have supporting frameworks and architectures, whereas others only present descriptive approaches. Since these approaches have not been used in practice, with the exception of demonstrating use cases, there is no actual user base to evaluate their viability. Therefore, the research activities undertaken by the proposed approaches to provide evidence for their viability are of relevance to developing an evaluation methodology for EXPRESS. Section 3.6 discusses these approaches in more detail.

3.1 Meta-Models in SWS Descriptions

Fensel (2004) states that there are two paths to SWS (Figure 5). The first starts from traditional Web services and complements them with semantics. The other starts from the Semantic Web and develops it further by adding more ontologies and semantic annotations, with services that make use of this data then emerging gradually.

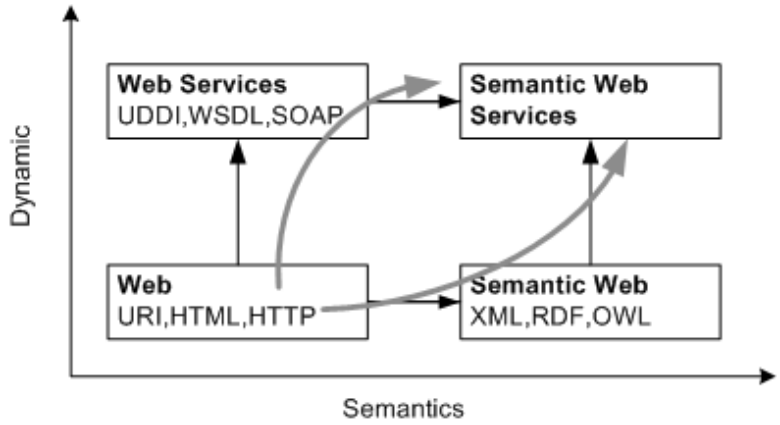


Figure 5 Paths to SWS (Fensel, 2004)

Figure 5 implies that Web services add dynamicity to the Web; however, dynamic Web pages predate Web services, although from a program point of view, utilising the functionality offered by a remote server, and hence dynamically interacting with it, was facilitated by Web services. This was a result of standardised, machine-readable service descriptions and standardised formats for exchanged data. One possible reason that RESTful Web services became a much more popular approach was that they do not need machine-readable service descriptions and blur the distinction between Web pages and Web services. The difference is thus found in the standards used for exchanging data, i.e. HTML versus XML, JSON or other data representation standards.

The existence of service descriptions is an interesting aspect to take into account when discussing approaches to SWS. This is because these influence whether the problems are conceptualised as an interaction with services, and hence a service meta-model is introduced, or whether they are conceptualised as an interaction with resources on the Web, and hence a resource-oriented meta-model is introduced.

Imposing a semantic meta-model for Web service descriptions has been the conventional route taken by the overwhelming majority of SWS approaches. The meta-models imposed in these approaches differ in regard to whether the approach describes a service or a resource. The description orientation of meta-

models imposes constraints, transforms conceptualisations, and adds artefacts when describing the service. The more this orientation fits the actual functionality described, as well as the Web's architecture, the smaller the descriptions, and thus the easier it becomes to describe it. Moreover, the more complex and demanding the semantic meta-model, the more it affects the adoption of the approach, and thereby lessens its value.

Below is an explanation of what is meant by service-oriented and resource-oriented meta-models.

1. Service-Oriented Meta-Models: These approaches separate service descriptions from the domain descriptions, and introduce meta-models to describe services, such as the names of operations, inputs and outputs, preconditions and effects. They can describe either WSDL or RESTful Web services. These include existing Web services which are semantically described to form SWS and weave them into the Semantic Web. They are based on the RPC mindset discussed in Chapter 2 and can be further classified into:
    a. WSDL-based SWS Approaches: These assume that the described Web services are traditional WSDL/SOAP services.
    b. RESTful Web Service Approaches: These describe RESTful Web services or Web APIs. These are considered as service-oriented approaches because they are treated and described as services not resources.
2. Resource-Oriented Meta-Models: these follow the lower path in Figure 5. In these meta-models conceptualise interactions as resources rather than services. Therefore the meta-models describe elements such as resource types, collections, representations and methods.

3.2 Service-Oriented Meta-Model Approaches

These approaches separate the service description from the domain description and can be either WSDL-based or RESTful Web services.

3.2.1 SWS Approaches for WSDL Web Services

Semantic Annotations for WSDL (SAWSDL) (Farrell and Lausen, 2007), which was developed from WSDL-S (Akkiraju *et al.*, 2005), is a lightweight solution and the only W3C SWS recommendation. It annotates WSDL components such as inputs and outputs with references to ontologies. It adds the attribute

`sawsdl:modelReference` to elements of the inputs and outputs, the value of the attribute would be a URI that points to a concept in an ontology. SAWSDL discards the precondition and effect attributes that were in WSDL-S, and it aims to be compatible with existing specifications and improve the automation of discovery and composition.

More ambitious W3C submissions for SWS, such as OWL-S, WSMO and SWSF, are more complex. OWL-S (Martin *et al.*, 2004) is based on OWL. OWL-S defines an ontology for describing Web services. It describes three aspects of the service: profile, process and grounding. The profile is for advertising and discovery and contains non-functional and functional properties: inputs, outputs, pre-conditions and effects (IOPE). The description of IOPE for a service originates from the AI notion of actions in the automated planning domain. The service process describes the logic of the service in regard to how inputs relate to outputs and pre-conditions to effects. The grounding describes mapping from the ontological description to a concrete specification of a service, for example to WSDL. OWL-S describes how to provide descriptions for composite services. These enables explicit yet manually built compositions of services. Moreover several approaches for automated composition for OWL-S have been surveyed by Klusch (2008a). Meaning that OWL-S lends support for both manual and automated orchestration. OWL-S use of OWL as a language based on description logics, hence operating under the open world assumption, moreover description logic restricts its ability to represent complex rules, OWL-S overcomes this by incorporating Semantic Web Rule Language SWRL (Horrocks *et al.*, 2004) for defining rules for preconditions and effects.

Another approach is WSMO (Bruijn *et al.*, 2005a), which is based on four major elements for modelling Web services: ontologies, Web services, goals and mediators. Ontologies provide the terminology to describe the domain and services. Web services describe service capabilities (pre-conditions, assumptions, post-conditions and effects) and interfaces (choreography – defining exchanged messages – and orchestration). Goals model the service requester's requirements, which are used for matchmaking with service capabilities. The definitions of choreographies and orchestrations in WSMO are based on Abstract State Machines (ASM), and are described by states and guarded transitions. WSMO uses WSML (Bruijn *et al.*, 2005b) as the language for modelling ontologies and rules, which is based on Frame logic (FL), unlike DL it follows the closed world assumption, meaning that unless something is stated it is assumed false. One of the criticisms of WSMO is that it drifted from the W3C standards (Bournez, 2005), although efforts have been made to build bridges between them. Klusch (2008a)

classified automated discovery and composition methods for SAWSDL, OWL-S and WSML. What is interesting is that many discovery methods can be applied to the three approaches, as they depend on the extraction of IO or IOPE, while for automated composition/planning more methods targeted OWL-S than either WSML or SAWSDL. The reason being that these planning methods are variations of well-established AI planners, and OWL-S as mentioned above are conceptualised as actions in AI planning. Research efforts in WSMO have stopped but are continued in lighter-weight approaches such as MicroWSMO (Kopecky *et al.*, 2008) and WSMO-Lite (Vitvar *et al.*, 2007) (discussed below).

Semantic Web Services Framework (SWSF) (Battle *et al.*, 2005) is another SWS approach, which builds upon the experiences of OWL-S and WSMO. It focuses on supporting workflows and like WSMO it has its own language for defining the Semantic Web Services Ontology (SWSO) called Semantic Web Services Language (SWSL) which supports both first-order logic and logic programming, hence offers greater expressivity than OWL-S. It provides a process model for web services that introduces concepts for control, ordering, states and exceptions. It has received less interest from the research community that the approaches above.

DIANE Elements (DE), which is an object-oriented language for service ontologies, is used by DIANE Service Description (DSD) (Klein *et al.*, 2005). DE provides reasoning support for sets and fuzzy sets that describe services inputs, outputs and effects. The rationale for introducing fuzzy sets is to enable variable degrees for matching of services, where the selection of a service is based on the fuzzy membership value of the service's effects in the requested effects. DSD takes an integrated approach towards service discovery and composition (Küster *et al.*, 2007).

iServe (Pedrinaci *et al.*, 2010b) is a publishing platform for semantic descriptions of WSDL services and Web APIs, to facilitate the discovery of services. It provides two annotation editors: one for Web APIs, called SWEET (Semantic Web sErvice Editing Tool), and the other for WSDL services, called SOWER (SWEET is nOt a Wsdl EditoR). The vocabulary used for the annotation combines several parts of other vocabularies, but is mainly based on the Minimal Service Model (MSM), which was designed to be the largest common denominator of the OWL-S, WSMO, and WSMO-Lite vocabularies. In addition, it uses some terms from other vocabularies, such as hRESTS, SAWSDL and WSMO-Lite. iServe works as follows: first, it facilitates the annotation of Web services; second, it publishes those annotations as Linked Data; third, it provides a Web API to create and retrieve the descriptions and a SPARQL endpoint to query the services' descriptions dataset.

3.2.2 SWS Approaches for RESTful Web Services

With RESTful Web services gaining more popularity on the Web, interest in RESTful SWS is rising. In REST-based approaches, existing RESTful Web services are semantically described. SA-REST (Lathem *et al.*, 2007) is similar to SAWSDL, as it introduces a vocabulary to semantically annotate RESTful Web services, but because there are no WSDL files for RESTful Web services, the annotations are embedded into HTML Web pages that describe the services for programmers. The annotations are embedded using RDFa (Adida *et al.*, 2008) or GRDDL (Halpin and Davis, 2007). By adding semantics, SA-REST aims to provide an easier way to create and coordinate mashups.

hRESTS (Kopecky *et al.*, 2008) is an HTML microformat for RESTful Web services. Microformats facilitate the extraction of accurate data from HTML pages. They provide designated values for markup tags' attributes to encode extra information about the content. Examples of popular microformats are hCalendar for events, and hCard for contact information. The attributes used are class, rel, and rev, usually in tags such as div, span, ul and il. In hRESTS, the attribute values are: service, operation, method, input and output. hRESTS highlights the important parts of a RESTful Web service description, however to add semantic annotations MicroWSMO (Kopecky *et al.*, 2008) was introduced. It extends hRESTS to add references to service models and lowering and lifting schemas. WSMO-Lite (Vitvar *et al.*, 2007) is a lighter-weight version of the WSMO service ontology, that can be used to describe services on top of MicroWSMO and also SAWSDL. Its aim is to reduce the overhead in describing services and to be able to annotate RESTful Web services.

These approaches aim to insert semantic annotation mechanisms into HTML documents, achieved by mechanisms such as hRESTS and RDFa. In comparison to hRESTS, RDFa is more flexible, as it does not restrict the type of triples added to the HTML documents, but hRESTS is less intrusive, because, as a microformat, it repurposes the use of certain attributes, whereas RDFa introduces new attributes that can cause compatibility problems.

RESTfulGrounding (Filho and Ferreira, 2009) is another method to semantically describe RESTful Web services. The authors introduce a new grounding ontology in OWL-S to accommodate RESTful Web services.

Another approach to RESTful SWS was introduced by Battle and Benson (2008). In their Semantic Bridge for Web Services (SBWS), they annotated WADL (Hadley,

2009) documents, similar to SAWSDL, which linked WADL components to ontologies. Their approach provided descriptions for WSDL too.

Several SWS approaches have emerged as a result of the increased interest in Linked Data. Linked Data Services (LIDS) (Speiser and Harth, 2011) and Linked Open Services (LOS) (Krummenacher *et al.*, 2010) are inspired by Sbodio and Moulin (2007), and Sbodio *et al.* (2010) in using SPARQL queries to describe services (SPARQL descriptions). LIDS aims to augment linked datasets dynamically with data extracted from Web APIs, so they focus on describing data services using RDF and SPARQL. LOS provides semantic wrappers for WSDL and Web APIs to function as RDF producers and consumers. It describes the functionality of services, using RDF and graph patterns, and then describes their composition in order to perform processes using SPARQL queries. However, LOS requires a shared triple space, where all service descriptions should exist.

SADI (Semantic Automated Discovery and Integration) (Wilkinson *et al.*, 2009) is a set of practices for the automated integration of bioinformatics data and services. It is based on the premise that compared to generic Web services, Web services in bioinformatics exhibit less functionality. They are atomic, stateless, and transformative. SADI utilises this by catering for these traits: as one of the distinctive aspects of SADI is based on the services being transformative, this is conceptualised in SADI by assuming that all services are annotating services. Hence, outputs are actually the inputs but with annotations linking them to other resources or transformations. SADI services also exchange RDF messages, which means that providing annotating services becomes straightforward, as the base URI of the input is the base URI of the output, but with more annotating triples. To describe the services, SADI uses the myGrid/Moby service model[6], with the inputs and outputs being OWL classes defined in a referenced ontology. The OWL classes used as inputs and outputs are named classes defined as equivalents of restrictions on properties (predicates). These predicates are important for SADI because they are used to facilitate the discovery and composition of services.

The discovery of SADI services is illustrated by providing a plug-in for Taverna (Oinn *et al.*, 2004). Taverna is a workflow management system for scientific workflows. It provides a canvas for dragging and dropping services and resources to create workflows. The SADI plug-in suggests applicable transformations according to the type of workflow output, which is done by displaying the properties that would be available as a result of executing the transformation service. Therefore, these properties link the inputs to the outputs of a service. SADI also demonstrates its composability through the Semantic Health And

[6] The myGrid Moby Service Ontology, http://www.mygrid.org.uk/mygrid-moby-service/ .

Research Environment (SHARE) system (Vandervalk *et al.*, 2009). SHARE enables users to query and analyse distributed data. It accepts SPARQL queries, and then extracts the query triples, and for each triple it finds Web services that provide triples matching the pattern. These Web services are executed, and then the intermediate results are returned and used to execute other matching services. SADI utilises HTTP to invoke services, so the service descriptions are retrieved by a GET method and data is sent by a POST. Moreover, it supports both synchronous and asynchronous services by utilising the HTTP response code 202 (Accepted but incomplete) for asynchronous services.

The main difference between SADI and other SWS is that it does not provide a new way of describing the service itself, as it adopts an existing model, but rather enforces constraints on how the inputs and outputs of that service are defined in the domain ontology.

3.3 Resource-Oriented Meta-Model Approaches

The majority of research efforts have so far been in semantically enhancing Web services, but recently, approaches that are based on semantic resources have appeared and these are discussed next.

Another part of Battle and Benson's work involved providing a RESTful interface for semantic data in a term they called Semantic REST (Battle and Benson, 2008). They mapped the HTTP methods (GET, PUT, POST and DELETE) into SPARQL commands, including extensions to SPARQL (proposed at that time by HP's Jena team) which were SELECT, INSERT, MODIFY and DELETE, these extensions were origins of the current W3C Recommendation SPARQL 1.1 Update (Gearon *et al.*, 2013). In this way, RDF datasets offering SPARQL endpoints can also offer new RESTful functionality, meaning they can be integrated with Web 2.0 clients.

Presto (DeLeon and Dumontier, 2008) provides a RESTful interface for resolving OWL ontologies and endpoints for DL and SPARQL queries. This is particularly effective when ontologies are large, e.g. in life sciences. Presto publishes the entities in OWL files and enables retrieval of axioms about these entities through a RESTful interface. This means Presto can be viewed as a RESTful Web service for resolving entities in OWL ontologies. Although Presto does not aim to offer a general framework for Web services, it shows the straightforward mapping from OWL entities to resources. Zhao and Doshi (2009) categorised RESTful Web services into three types: resources representing sets of resources, resources representing instances, and resources representing transitional services. They described these types using a lightweight ontology and rules for describing the

transitional services. Their aim was to facilitate the automatic composition of RESTful Web services, so they provided a framework for composing those services using a state transition system (STS) based on situation calculus. According to the classification introduced at the beginning of this chapter, their approach also includes a service-oriented meta-model. This is because they used ontologies to explicitly describe the third type of resources in their description, i.e. transitional services.

Another approach that is based on semantic resources is Triple Space Computing (TSC) (Riemer *et al.*, 2006), which is based on Tuple Space Computing. The communication is shifted from being message oriented, as in Web services, to reading and writing RDF triples in a shared triple space. TSC has been used in both Web service coordination (Fensel *et al.*, 2007) and communication (Francisco *et al.*, 2008). Hernandez and Garcia (2010) took TSC further by modelling resources in triple spaces, and mapping HTTP methods into triple space operations. Furthermore, they also provided a process calculus method for describing the composition of these resources. However, being confined to a shared triple space limits the scalability and accessibility of such approaches.

SSWAP (Gessler *et al.*, 2009) is a protocol and architecture for SWS. It enables the creation and discovery of RESTful Web service descriptions. It was developed to be used mainly in the field of bioinformatics, but it is proposed to be used for generic Web services too. SSWAP provides an ontology for describing a service, and this ontology has five main concepts: Provider, Resource, Graph, Subject and Object. A Provider (organisation) provides a Resource, which corresponds to a service. The Resource operates on a Graph. The Graph describes the mapping between the input of the service described by the Subject, and the output described by the Object. SSWAP assumes a relationship between the input and output, and conceptualises this relationship as a mapping. The descriptions of the services are called Resource Description Graphs (RDGs). SSWAP interacts by exchanging RDGs, so the client provides values for the inputs and POSTs the RDG to the service. This is called the Resource Invocation Graph (RIG). Then, the service provides values for the outputs and returns it to the client, and the returned description is called the Resource Response Graph (RRG). SSWAP provides an SDK for developing services, and a method for publishing them to the service directory that SSWAP hosts. This directory facilitates the discovery of Web services. The directory is used to build SSWAP.info, which is a Web-based interactive pipeline editor, where a user can drag and drop services and available services are filtered according the outputs of the selected ones.

ReLL (Alarcon and Wilde, 2010) (Resource Linking Language) is an approach that describes existing RESTful Web services on the Web (i.e. Web APIs) and also Web pages to enable a crawler called RESTler to crawl them and produce a typed graph representing the links, relationships between them and the representations. This graph can then be translated into RDF. The aim of ReLL (Alarcon and Wilde, 2010) is to establish a unified view of these resources. Although ReLL currently describes read-only situations, the aim is to extend it to support creation, modification and deletion.

RESTdesc (Verborgh *et al.*, 2011) explicitly provides N3Logic (Berners-Lee *et al.*, 2008) rules for each resource, which describe the method, representation, and URI structure, such that the client can reason over those rules, and execute an HTTP request according to its internal state and the satisfied rules. Moreover, it utilises link headers to guide clients to the next states.

Hyperdata (Kopecky *et al.*, 2011), on the other hand, proposes a method for updating RDF data stores. It is based on the argument that updating data via SPARQL endpoints is not sufficient because of 1) data dependencies, where updates need to be propagated to dependent data that are not expressed in the SPARQL query; 2) security issues; and 3) validation. For these reasons, Hyperdata is proposed to update RDF stores through APIs. However, instead of describing the APIs separately, the API descriptions are stored as triples with the data in the RDF store. It uses named graphs to represent API endpoints for resources in the RDF store that will then be manipulated by the API. They have four types of resources: classes, individuals, property resources, and value resources. The approach uses a custom minimal vocabulary to describe the named graphs for these resources and associated triples, as well as the triple patterns and the relationships between them. The triples and triple patterns denote what will be affected by the HTTP methods. These API descriptions are stored with the data itself and are returned with the resource when it is retrieved. Thus, the description of any one endpoint is also linked to other endpoints within the application, so a client could navigate between endpoints.

Hypermedia RDF (Kjernsmo, 2012) is a proposed vocabulary to make RDF a hypermedia type. A hypermedia type is a term defined by Amundsen (2011b) as: "MIME media types that contain native hyper-linking semantics that induce application flow. For example, HTML is a hypermedia type; XML is not". Amundsen also defines a classification scheme of hypermedia types called H Factors, which are used to measure the level of hypermedia support the media type offers. Hypermedia RDF is influenced by Amundsen's argument for making RDF sterilisations more powerful, instead of providing an API for RDF. The argument is

that the Web is more successful because messages contain not only data but application control information. The Hypermedia RDF vocabulary defines a set of predicates and instances to describe what actions are applicable in regard to a certain resource, for example that it can be updated, deleted, merged into, or accepts formats. The approach does not specify the repercussions of updating or deleting a resource.

RDF-REST (Champin, 2013) is an approach and an implementation (in Python) that provides a unified method to provide both Web APIs and Linked Data. It therefore facilitates the implementation of systems that expose both. This is done by having RDF-REST as a layer embedded in the system architecture. It abstracts the logic layer as a set of core objects or resources that expose a uniform interface. The uniform interface provides methods that correspond to the HTTP methods, and RDF representations are thus exchanged. One of the main design decisions in RDF-REST is to have RDF as the native system format. Therefore, application-specific Web APIs are provided through wrappers that interact with the core objects. The wrappers use serialisers and parsers to transform between RDF and other media types. Therefore, RDF-REST aims to comply with the Linked Data Profile specification (Nally *et al.*, 2012a) for manipulating Linked Data. One of its limitations is that it is designed for developing Web APIs from scratch, so these Web APIs would consequently be built on top of RDF-REST.

3.4 A Classification Matrix for SWS Approaches

Another way to classify SWSs is to look at the approach they take in enriching services with semantics. These are not mutually exclusive and one may build on the other (e.g. SAWSDL and OWL-S). Cabral *et al.* (2012) classify the description approach into: service ontologies, and semantic annotation extension mechanisms. The following matrix uses the meta-model classification discussed in this chapter for the horizontal axes. For the vertical axis, it extends the description classification presented in (Cabral *et al.*, 2012) by first recognising that ontologies are not only service ontologies: they can also be for describing resources in resource-oriented approaches. And secondly, it further classifies "Semantic Annotation Extension Mechanisms" into two subclasses: "Link to Concepts in Ontologies" and "Use Graph Patterns". In addition, some approaches cannot be considered description approaches, as they focus on methods for providing services, so a row has been added for "Provision Approaches" in the matrix. A brief description of these classifications is provided below.

Description Approaches

1. **Ontologies/Vocabularies: These approaches introduce ontologies or vocabularies. The majority are for describing services; however there are some that describe resources. These are different from a generic domain ontology, as they have been introduced specifically for the purpose of describing a service or resource interface.**
2. **Semantic Annotation Extension Mechanisms: These provide mechanisms to annotate a specific service or resource with descriptions. These can be categorised into approaches that:**
    a. **Link to Concepts in Ontologies: These are mainly based on linking to concepts defined in an ontology to describe inputs, outputs, preconditions, effects, groundings, etc.**
    b. **Use Graph Patterns: These approaches utilise graph patterns to describe functionality in a service. They are more flexible than the approaches that link to concepts in ontologies, and this flexibility is discussed at the end of this section.**

Provision Approaches

These provide conceptualisations architectures and implementations of methods to provide SWS.

			Type of Meta-Model		
			Service		Resource
			WSDL WS	RESTful WS	
Description Approach	Ontology/ Vocabulary		OWL-S WSMO SWSF WSMO-Lite DSD MSM	RESTfulGrounding WSMO-Lite MSM Zhao & Doshi	SSWAP ReLL Hypermedia RDF Zhao & Doshi
	Annotation Extension Mechanism	Link to Concepts in Ontologies	WSDL-S SAWSDL LOS	hRESTS SA-REST MicroWSMO SBWS LOS SADI	Hernandez & Garcia

		Use Graph Patterns	LOS SPARQL descriptions	LIDS LOS SPARQL descriptions	SSWAP ReLL RESTdesc HyperData Hernandez & Garcia
Provision Approach					Semantic REST RDF-REST Hernandez & Garcia TSC

Figure 6 Classification Matrix of SWS Approaches

SAWASDL, SA-REST, hRESTS, LIDS, LOS, HyperData and RESTdesc are classified as extension mechanisms; however, they do introduce minimal vocabularies, but because these vocabularies are minor they are considered mainly extension mechanisms.

Some approaches, such as LOS and Hernandez and Garcia (2010), fall under several classifications because they use a mixture of approaches to achieve their aims. LOS describes both RESTful Web services and WSDL ones, and in terms of extension mechanisms, links to ontologies and uses graph patterns for inputs and outputs. Hernandez and Garcia (2010) argue for the use of triple spaces, and process calculus to formally represent RESTful Web services, hence providing services using triple spaces. In addition their approach assumes that the services would be described by linking to ontologies and also using graph patterns.

As mentioned above, semantic service-oriented meta-models for WSDL-based services suffer from being too complex. This has led to new approaches shifting towards describing increasingly popular RESTful Web services. However these approaches explicitly describe their inputs and outputs, and in some cases pre- and post-conditions; thus, in addition to adding an extra description layer, they impose an RPC-mindset on these descriptions. So instead of them being conceptualised as resources, as they would be in REST, these are transformed into services. Moreover, these services focus on data retrieval and do not offer extended functionality.

By comparison, resource-oriented meta-models focus on describing resources, and all of these SWS approaches (except ReLL, RESTdesc, Hypermedia RDF and HyperData) do not consider REST's constraint of using hypermedia as the engine of the application state, which provides an alternative method for the creation of conversational interactive services for RESTful Web services. There is, however, an issue with ReLL, RESTdesc, Hypermedia RDF and HyperData, in that they still introduce vocabularies to describe how to interact with certain endpoints. ReLL and RESTdesc, in particular, introduce vocabularies for descriptions that are

already provided by HTTP and do not need to be explicitly defined. Such descriptions can be eliminated because adding them introduces redundancy and there is an overhead in keeping them consistent.

The classification presented in this chapter is based on how SWS approaches differ both conceptually and syntactically in their description of services. However, there are other ways to classify SWS, one example provided by Klusch (2008a) presented two comprehensive classifications of SWS discovery and composition methods. For the discovery approaches, he classified 27 methods according to

1. which parts of the service description are used in the matchmaking process. These could be the profile (IOPE), the process, or non-functional properties; and
2. how the matchmaking is performed, i.e. whether it is logic-based, non-logic-based (text similarity, graph matching) or a hybrid of both.

For SWS composition methods, he classifies 16 methods based on

1. whether there was interleaving between planning and execution (i.e. static or dynamic); and
2. Whether they are based on the SWS profile description (Functional Level Composition: FLC) or on the SWS process description (Process Level Composition: PLC).

An important issue to note, however, is that most of these discovery and composition methods are for the same SWS approaches, namely OWL-S, WSMO and SAWSDL, which, according to the classification presented in Figure 6, fall under service-oriented meta-models for WSDL Web services. The service is semantically annotated in these approaches by mainly linking to concepts in ontologies, which greatly influences how discovery and composition approaches are implemented.

As shown in the classification matrix, another method for annotating services is by using graph patterns, this provides greater flexibility for the descriptions. In approaches that annotate by linking to concepts, inputs and outputs either link to classes in an ontology or to simple data types, and when linking to simple data types, there is no direct mechanism for telling what that simple data type represents semantically. However, with graph patterns, inputs and outputs are variables in these patterns, either as subjects or objects of predicates/properties. This means that inputs and outputs could be simple data types while also being described as an object or range of a certain predicate. The implication of using graph patterns for description goes beyond providing more flexibility, as they

introduce new methods for matching services, which are more scalable (Stadtmüller and Norton, 2013).

3.5 Comparison of SWS Approaches Capabilities

In the previous section SWS approaches were classified according to how they conceptualised and described interfaces, in this section their capabilities are compared. The 27 SWS approaches were analysed according to which of the following capabilities they offer, and the results are shown in Table 1.

1. Discovery

One of the main goals of SWS is to facilitate automated discovery of services, therefore the purpose of many semantic description approaches is to address discoverability.

2. Composition

Composition is the process of integrating several services or resources in a workflow to achieve a certain goal. There are four main ways that composition has been addressed in SWS:

2.1 Orchestration

There is a single point of control one entity is responsible for the execution of the workflow. In the execution of the workflow, this entity is acting as a client to the services that compose the workflow. The workflow is typically known in complete to the controlling entity before it starts executing it. The Web service community have introduced several specifications to describe workflows such as the Web Services Business Execution Language (WSBPEL) (Alves *et al.*, 2007), their aim was to have interoperable descriptions of the workflows which can be processed by execution engines. One of the areas that SWS approaches targeted was to introduce vocabularies/ontologies for describing these workflows semantically, such as composite services in OWL-S and orchestrations in WSMO.

2.2 Automated Composition Planning

This is another way to achieve the orchestration of services that utilises the semantic descriptions of services. It automates the composition of services using AI planning techniques, which view the world as states, where Web services are actions that alter these states, and can be composed to achieve stated goals. As mentioned above several SWS composition techniques have been surveyed by Klusch (2008a) and has received growing attention from the SWS research community.

2.3 Choreography

The aim of choreography in Web services is to enact a global plan/workflow that is known to the participating entities, and is achieved when individual participants execute their parts/roles. There is no single point of control. In a Web environment, enacting the choreography means that the same participants will act as both clients and services in a peer-to-peer fashion. As for orchestration, the Web service community has introduced specifications for standardising choreography descriptions, such as the Web Services Choreography Description Language (WS-CDL) (Kavantzas *et al.*, 2005). In SWS, WSMO describes service interfaces as choreographies, and introduced an ontology for describing choreographies of services as states and guarded transitions. With regards to creating choreographies automatically, it can be considered a self-organisation or a multi-agent planning problem as in (Falou *et al.*, 2009). However this class of problems is not popular in the SWS research community, and as mentioned above, most of the research focused on automatically creating orchestrations by using AI planning, rather than automatically creating choreographies.

2.4 Conversational Services

In RESTful Web services, the server guides the client through the next steps, this the one of the constraints on the uniform interface in REST, namely using "hypermedia as the engine of the application state". Therefore when the client is following the steps, it is actually interacting with several endpoints (resources), and hence a form of composition. The server is controlling the workflow, however the client has the autonomy to opt out at anytime, and to interact with endpoints on other servers. Unlike orchestration and choreography in traditional Web services, there is no declarative specification of workflow; however signposting mechanisms are built into the media types. The workflow unfolds to the client, and it knows how to respond at each step, but is not aware of the complete workflow. As discussed in Chapter 2, few RESTful APIs adhere to the hypermedia constraint. However there are SWS approaches that acknowledge the hypermedia constraint and introduced vocabularies to describe possible choices to the client such as ReLL (Alarcon and Wilde, 2010), RESTdesc (Verborgh *et al.*, 2011), Hypermedia RDF (Kjernsmo, 2012), and Hyperdata (Kopecky *et al.*, 2011), these approaches were explained previously in Section 3.3.

2.5 Linked Data Integration

With Linked Data becoming increasingly popular, many recent approaches to SWS have targeted providing interface descriptions for Linked Data in the aim of facilitating access to datasets through APIs instead of using SPARQL endpoints, and to merge datasets together or with other non-linked data resources. And while the aim of these interfaces is not service composition in the strict sense, if

we take a RESTful view, the distinction between services and resources is blurred, and data integration could be regarded as integration of resources. In some of the approaches this type of integration has no side-effects, in other words it is merely data retrieval, however this does not need to be the case and there are others, where the integration of data automatically triggers real-world events. Table 1 shows each of the 27 approaches, and the capabilities they address.

Table 1 Capabilities of SWS approaches

Publication	Purpose	Capabilities					
			Composition				
		Discovery	Orchestration	Auto. Composition	Choreography	Conversational	Linked Data Int.
OWL-S (Martin *et al.*, 2004)	General	✓	✓	✓	x	x	x
WSMO (Bruijn *et al.*, 2005a)	General	✓	✓	✓	✓	x	x
SAWSDL (Farrell and Lausen, 2007)	General	✓	x	✓	x	x	x
WSDL-S (Akkiraju *et al.*, 2005)	General	✓	x	✓	x	x	x
SWSF (Battle *et al.*, 2005)	General	✓	✓	✓	✓	x	x
DSD (Klein *et al.*, 2005)	General	✓	x	✓	x	x	x
SA-REST (Lathem *et al.*, 2007)	General	✓	x	✓	x	x	x
hRESTS (Kopecky *et al.*, 2008)	General	*	x	*	x	x	x
MicroWSMO (Kopecky *et al.*, 2008)	General	*	X	*	x	x	x
WSMO-Lite (Vitvar *et al.*, 2007)	General	✓	X	✓	x	x	x
RESTfulGrounding (Filho and Ferreira, 2009)	General	✓	✓	✓	x	x	x
ReLL (Alarcon and Wilde, 2010)	Data Retrieval	✓	✓	x	x	✓	✓
SBWS (Battle and Benson, 2008)	General	✓	x	✓	x	x	✓
SPARQL descriptions (Sbodio *et al.*, 2010)	General	✓	x	✓	x	x	x
LIDS (Speiser and Harth, 2011)	Data Retrieval	✓	x	x	x	x	✓
LOS (Krummenacher *et al.*, 2010)	General	✓	✓	x	x	x	✓
Semantic REST (Battle and Benson, 2008)	General	x	x	x	x	x	✓
Zhao and Doshi (2009)	General	x	x	✓	x	x	x
Hernandez and Garcia (2010)	General	x	✓	x	✓	x	x
TSC (Riemer *et al.*, 2006)	General	✓	*	*	*	*	✓
RESTdesc (Verborgh *et al.*, 2011)	General	✓	x	✓	x	✓	x
iServe (Pedrinaci *et al.*, 2010b)	General	✓	x	✓	x	x	x
SADI (Wilkinson *et al.*, 2009)	Bioinformatics	✓	x	✓	x	x	✓
HyperData (Kopecky *et al.*, 2011)	General (LD)	x	x	x	x	✓	✓
Hypermedia RDF (Kjernsmo, 2012)	General (LD)	x	x	x	x	✓	✓
RDF-REST (Champin, 2013)	General (LD)	x	x	x	x	✓	✓
SSWAP (Gessler *et al.*, 2009)	Bioinformatics	✓	x	✓	x	x	x
✓: addressed by the approach **x**: not addressed by the approach *: assumed existing & addressed by other layers							

The purpose of the approach can be one of the following:

1. General: the approaches are not specific to a domain.
2. Generic (LD): it is the same as General except it deals with Linked Data.
3. Data Retrieval: the services targeted do not change data or the state of the world.
4. Bioinformatics: the approaches are specific to the bioinformatics domain only.

Looking at the table, most approaches 19 out of 27 target service discovery, and the ones that did not are: Zhoa and Doshi, Hernandez and Garcia, Semantic REST, Hyperdata, Hypermedia RDF and RDF-REST. The former two approaches focused on utilising semantic technologies to provide a formal definition of resource-orientation, Hernandez and Garcia (using triple spaces and process calculus) and Zhoa and Doshi (an ontology for resource types and situation calculus). The latter four approaches focused on providing platforms or interfaces for linked data. In Section 3.4, the six approaches, which did not target service discovery, had resource-oriented meta-models; suggesting that when these approaches diverted from the service-oriented mindset, they also diverted from the goals typically targeted by the service-oriented approaches. Most of the approaches that targeted discovery exposed either IO or PE, or both. Approaches that exposed IOPE are: OWL-S, WSMO, WSDL-S, SWSF, DSD, WSMO-Lite, RESTful Grounding, LOS and iServe. WSMO-Lite also added service categories. Approaches that expose IO are: SAWSDL, SA-REST SBWS, SADI and LIDS, and approaches that exposed PE are SPARQL descriptions and RESTdesc. RO approaches such as SSWAP, TSC, and ReLL exposed resources in their interfaces.

For approaches that exposed IOPE, the matchmaking techniques typically applied involve profile matching for IO or specification matching for PE or both. The matching can be logic matching checking subsumption of concepts in IO, and the entailment of PE. It can also be non-logic matching which utilises the structural textual aspects of the underlying concepts. Moreover LIDS, LOS and RESTdesc use graph patterns to describe interfaces, these graph patterns can be matched according to the similarity of their predicates and resources (Stadtmüller and Norton, 2013). The DSD described in the DIANE language enables fuzzy matching of service requests and service offers. The matching boils down to checking if the service offer's effects are a subset of the service request's effects. The IO in DSD are part of the effect definition. In SPARQL descriptions, an agent's goals are represented as ASK queries, and services as CONSTRUCT queries, the CONSTRUCT clause represents the effect of the service and the WHERE clause represents its precondition. The agent has a KB, the service matching has two steps: 1) check if the agent satisfies the preconditions, which it does if the CONSTRUCT query representing the service yields results when applied to the agent's KB. 2) after

those results are obtained check if they fulfil the client's goal by applying the ASK query to them.

Regarding orchestration, fewer approaches attempt to formulate workflows compared to discovery, they are: OWL-S with composite services (which applies also to RESTful Grounding), WSMO, using interfaces that describe choreographies and orchestrations, and SWSF, which specifically targets workflows by providing a process model based on the Process Specification Language (PSL) (ISO 18629). Hernandez & Garcia combined process calculus and triple space operations, ReLL used Petri nets (Reisig, 1985), and LOS process model and SPARQL queries.

Many approaches target automatic composition; most apply AI planning methods, where the world is modelled as states, and the services as actions that alter states and have prerequisites (i.e. have preconditions and effects). For example in OWL-S descriptions are typically transformed to PDDL, and hence, several planning algorithms can be applied (Klusch, 2008a). Approaches that targeted WSMO also converted descriptions into PDDL (Farnaghi and Mansourian, 2013) or Hierarchal Task Networks (HTN) (Tabatabaei *et al.*, 2009). There were no reported approaches for SWSF however it is very similar to the OWL-S and WSMO, and therefore the same methods can be applied. Approaches that automatically composed SAWSDL added PE to the service descriptions (Klusch, 2008a). Zhao & Doshi conceptualised RESTful Web services as actions that are comprised of the HTTP method and the resource (these actions have preconditions and effects) then modelled them in Situation Calculus and used regression to derive compositions automatically. Automatic composition in WSMO-Lite was achieved by modelling the problem as a STRIPS instance (Fikes and Nilsson, 1971), and then the Graphplan algorithm (Blum and Furst, 1997) was applied. For SPARQL descriptions, when a goal is not satisfied by one service, the precondition is relaxed by using the OPTIONAL clause for the triple patterns, resulting in a set of graph patterns which if cannot be fulfilled by a single service are adopted as new goals, and regression planning is applied. Planning in RESTdesc is provided by constructing proofs, since services are modelled as N3 rules. Although ReLL does not target automatic composition, it does however model services as Petri nets, which suggest compositions can be created using Petri net reachability algorithms.

Only three approaches target choreography these are WSMO, SWSF and Hernandez & Garcia. WSMO uses its choreography ontology, SWSF uses its FLOW ontology and

Hernandez & Garcia as mentioned above combines process calculus and triple space operations.

RO approaches: ReLL, RESTdesc, Hyperdata, Hypermedia RDF and RDF-REST provided or supported vocabularies for describing conversational mechanisms to guide clients to next states. These also support Linked Data integration together with SBWS, LIDS, LOD and TSC.

3.6 Adopted Research Methodologies in SWS Approaches

This section aims to provide an analysis of the research methodologies that the 27 SWS approaches discussed in this chapter applied to provide evidence for their viability and effectiveness, the goal is to inform the choice of methodology for evaluating the EXPRESS approach proposed in this thesis. This analysis draws on Shaw's (2002) model for analysing research strategies for software engineering. Shaw classifies research strategies employed in software engineering research papers by identifying the types of research questions they explore in the paper, the types of results produced, and the type of validation provided. Her work aimed to encourage experimental validation in software engineering research by explicitly describing generally accepted research strategies in software engineering. To analyse the research strategies in SWS approaches, 27 publications that introduce the SWS approaches were selected, in addition to five others that presented evaluation efforts for certain SWS approaches.

Research Questions in SWS Approaches

Of the types of research questions identified by Shaw, the ones that are addressed by research in SWS approaches are about:

- Design, evaluation or analysis of a particular instance:
    - What is a (better) design or implementation of the SWS approach?
    - How does an X SWS approach compare to a Y one?
- Feasibility/Viability
    - Is it possible to accomplish this SWS approach?

Other research question types mentioned by Shaw were: means of development, method for analysis, and generalisation/characterisation.

Research Results in SWS Approaches

With regard to the types of research results identified by Shaw, the SWS description approaches fall under the following the "Specific Solution" types. According to Shaw (2002) this can be any of the following:

- **design, prototype, or full implementation**
- **careful analysis of a system or its development,**
- **result of a specific analysis, evaluation, or comparison.**

Results in SWS approach papers are mainly the approach itself, its implementation and associated tools if available. In evaluation papers, these were the results of the evaluation and comparison.

Validations in SWS Approaches

Shaw (2002) categorises the validation approaches into types shown in Table 2

Table 2 Validation techniques in software engineering (Shaw, 2002)

Type of validation	Examples
Analysis	I have analysed my result and find it satisfactory through (formal analysis) … rigorous derivation and proof (empirical model) … data on controlled use (controlled experiment) … carefully designed statistical experiment
Experience	My result has been used on real examples by someone other than me, and the evidence of its correctness / usefulness / effectiveness is (qualitative model) … narrative (empirical model) … data, usually statistical, on practice (notation, tool) … comparison of this with similar results in technique actual use
Example	Here's an example of how it works on (toy example) … a toy example, perhaps motivated by reality (slice of life) …a system that I have been developing
Evaluation	Given the stated criteria, my result... (descriptive model) … adequately describes the phenomena of interest … (qualitative model) … accounts for the phenomena of interest… (empirical model) … is able to predict … because …, or … gives results that fit real data … Includes feasibility studies, pilot projects
Persuasion	I thought hard about this, and I believe (technique) … if you do it the following way, (system) … a system constructed like this would … (model) … this model seems reasonable. Note that if the original question was about feasibility, a working system, even without analysis, can be persuasive
Blatant assertion	No serious attempt to evaluate result

The validation types in the reviewed SWS publications fall under four types from the above classification: examples, persuasion and analysis and evaluation. Table 3 shows the types of validation for each publication, and is a summary of Table 25 in Appendix A, which provides a short description for each publication

detailing what the paper achieves, then states the results mentioned in the paper and their validation.

Table 3 Validation approaches in SWS

Publication	Validation Approach			
	Examples	Persuasion	Analysis	Evaluation
OWL-S (Martin *et al.*, 2004)	✓	✓		
WSMO (Bruijn *et al.*, 2005a)	✓	✓		
SAWSDL (Farrell and Lausen, 2007)	✓	✓		
SWS Coordination (Klusch, 2008a)			✓	
SWS Comparison (Cabral et al., 2004)		✓		
WSDL-S (Akkiraju *et al.*, 2005)	✓	✓		
SWSF (Battle *et al.*, 2005)	✓	✓		
DSD (Klein *et al.*, 2005)	✓	✓		
SA-REST (Lathem *et al.*, 2007)	✓	✓		
hRESTS (Kopecky *et al.*, 2008)	✓	✓		
MicroWSMO (Kopecky *et al.*, 2008)	✓	✓		
WSMO-Lite (Vitvar *et al.*, 2007)	✓	✓		
Kopecky (2012)	✓	✓	✓	
RESTfulGrounding (Filho and Ferreira, 2009)	✓	✓		
ReLL (Alarcon and Wilde, 2010)	✓	✓		
SBWS (Battle and Benson, 2008)	✓	✓		
SPARQL descriptions (Sbodio *et al.*, 2010)	✓	✓	✓	
LIDS (Speiser and Harth, 2011)	✓	✓		
LOS (Krummenacher *et al.*, 2010)	✓	✓		
Semantic REST (Battle and Benson, 2008)	✓	✓		
Zhao and Doshi (2009)	✓	✓		
Hernandez and Garcia (2010)	✓	✓		
TSC (Riemer *et al.*, 2006)		✓		
RESTdesc (Verborgh *et al.*, 2011)	✓	✓		
iServe (Pedrinaci *et al.*, 2010b)	✓	✓		
SADI (Wilkinson *et al.*, 2009)	✓	✓		
HyperData (Kopecky *et al.*, 2011)	✓	✓		
Hypermedia RDF (Kjernsmo, 2012)		✓		
RDF-REST (Champin, 2013)	✓	✓		
SSWAP (Gessler *et al.*, 2009)	✓	✓		
SWS Challenge (Petrie et al., 2009)				✓
S3 Contest[7]			✓	

According to the results of the analysis in the above table the validation undertaken by the majority of the approaches was by using examples, 81%, and persuasion, 91%. The majority of examples were "toy" examples, simplified to ease the illustration of the approach. Persuasion was achieved either by discussing a proof-of-concept implementation or providing links to online demonstrators or supporting tools.

The publications that used analysis for validation constituted only 12.5% of the papers and the type of analysis fell under "Experiment with statistically significant results". These included the SPARQL descriptions (Sbodio *et al.*, 2010),

[7] **S3 Contest** http://www-ags.dfki.uni-sb.de/~klusch/s3/

and WSMO-Lite in Kopeckey (2012), which provided matchmaking experiments as a validation of the discoverability of their proposed descriptions. This involved converting the OWL-S test collection (Klusch and Kapahnke, 2010b) to their approaches, either adapting a matchmaker or implementing one, and comparing the results to existing matchmakers on the OWL-S test collection.

OWL-S, WSMO and SAWSDL discoverability and composability have been demonstrated in the SWS Coordination (Klusch, 2008a), which surveyed several matchmaking and planning algorithms designed for these approaches. Their discoverability has also been demonstrated in the S3 contest for service matchmaking, and thus was considered a representative of approaches that used analysis for validation.

In the SWS Challenge (Petrie *et al.*, 2009), the participants are given realistic scenarios and are asked to fulfil them with their proposed SWS approaches. The challenge evaluates the approaches according to their ability to mediate between different formats, and to provide accurate descriptions for specified WSDL services. Their accuracy is tested on their ability to be selected automatically and accurately. Thus according to Shaw's classification, this was the only publication that used evaluation as a validation method. Moreover the SWS Challenge aimed to understand the trade-offs between different approaches and how much human intervention is needed to modify services to adapt to changes in the requirements. However, the results of the challenge were not promising and no participant had solved all of the problems, and they found even the simplest problems challenging (Petrie *et al.*, 2009, p.284).

The analysis shows that SWS is an emerging research area and that the community has no well-established methods for evaluating new approaches, and (as shown by the results of the SWS Challenge) approaches struggle to meet the requirements of realistic problems.

A potential solution to evaluate SWS, involves analysis of expert opinions, as undertaken by Bachlechner and Fink (2008), albeit this time to assess the viability of SWS in general, not a specific approach, and therefore not present in Table 3. Their study involved surveying and analysing opinions from both practitioners and researchers to evaluate the potential of Semantic Web services as integration architectures, and using Shaw's categories, their validation technique is considered an evaluation.

3.7 Conclusions

This chapter has discussed and analysed twenty-seven SWS approaches. These approaches impose either service-oriented or resource-oriented meta-models, and show that the focus of research activities is shifting towards RESTful Web services and resource-oriented meta-models. The description approaches also differed in their method: some introduced ontologies or vocabularies, and others were annotation extension mechanisms. They all assumed Web services are already implemented and exist as an extra semantic layer.

Some approaches were not concerned with how the services are described but in how to provide them. Approaches such as TSC, Semantic REST, RDF-REST, describe the architecture or implementation for providing resource-oriented SWS that exchange RDF messages, but they do not specify how these services are described.

Hernandez and Garcia (2010) proposed providing services using triple space computing, and suggested the existence of service descriptions and their link to existing ontologies; however, they did not specify those descriptions. Moreover, their architecture is implemented by interacting with triple spaces, which imposes a specific architecture on service providers, making it harder to adopt.

The review of these approaches raises three interesting questions:

1) Is a meta-model even needed for resource-oriented services?

On the Semantic Web, resources are described by ontologies and these ontologies are used in SWS as domain ontologies. However, as domain ontologies were seen as insufficient to describe the functionality of the service, service ontologies have been developed. REST provides a unified way to access resources, with well-defined semantics. Therefore, can the combination of both REST's unified interface and the semantic description of resources be sufficient to describe the *functionality* of the service?

2) Can the description be a result of the provision of the service?

Since the mapping between entities in the domain ontology and restful resources is seemingly straightforward, is it possible to utilise that mapping so that the description of the service is a by-product of its provision?

3) What are the types of results, and validations that are applicable for these research questions?

Research in SWS is a relatively new field that has not been heavily used in practice and the analysis of strategies adopted by SWS approaches showed that the results were mainly the approach itself and the validation was by providing examples or by persuasion by providing demonstrators, or proof of concept implementations. Nevertheless there were validations based on experiments and formal comparisons. In addition, there was qualitative analysis of expert opinions, introduced by Bachlechner and Fink (2008).

In the light of this analysis the approach taken in this thesis is therefore to view the result as the EXPRESS approach and its implementation, and to undertake a broad validation combining several methods, specifically:

1. Examples and Persuasion: To show that it works and how it works.
2. Analysis: Experiments that test its efficiency.
3. Evaluation: Qualitative analysis of expert opinions comparing the approach to others and discussing the trade-offs.

Chapter 4, presents a primary stage in answering the first question. It presents a scenario analysis to elicit the required functionality of SWS, then studies the limitations and requirements from the proposed approach that eliminate an explicit meta-model.

Chapter 4: Scenario Analysis and RO Modelling

In the previous chapter, several SWS approaches were reviewed. These approaches semantically describe functionality to automate or semi-automate their discovery and composition. They had two underlying assumptions in common: firstly, explicit interface descriptions are required to describe the functionality, and secondly, a domain ontology/vocabulary exists that describes entities/resources manipulated by these interfaces. These two assumptions exist in all of the reviewed approaches regardless of whether they were service or resource-oriented. They all imposed an explicit semantic meta-model to describe the functionally, in addition to semantic descriptions in the domain ontology. This thesis questions these two assumptions.

In the conclusions of Chapter 3, the following question is asked: If resources on the Semantic Web are described in ontologies (domain ontologies), and REST provides a uniform method for manipulating resources, can these two elements semantically describe the functionality of SWS applications?

This question can be decomposed into the following two questions:

- What sorts of functionality are SWS required to describe?
- If explicit service descriptions are eliminated, what are the requirements and limitations in both the domain ontology and HTTP (as a RESTful mechanism) when describing the required functionality?

This chapter addresses the above questions, by analysing 20 scenarios to study the requirements of SWS, and then represents them as Resource-Oriented Models to investigate the requirements and limitations in both the domain ontology and HTTP. Section 4.1 explains the method of selecting the scenarios, Section 4.2 presents the analysis and results and Section 4.3 reflects on the SWS approaches from Chapter 3, and Section 4.4 concludes the chapter.

4.1 Web Service Scenarios

The approach taken by this research is to elicit the functional requirements from real representative scenarios. However, another possible approach to gather the functional requirements could have been to study the features and functionality offered by other Web service approaches. The danger of this is of over engineering and adding unnecessary complexity. Another reason for studying the scenarios instead of technologies is discussed by Foster *et al.*(2008), when comparing modelling state in different Web service specification approaches:

> **"Ideally, we would like to evaluate the relative merits of these two positions in terms of concrete metrics such as code size. Such an evaluation, however, requires agreement on the requirements that the interfaces should support. Unfortunately, proponents of the different approaches tend to differ also in their views of requirements."**

Therefore grounding the requirements in representative scenarios will provide less subjective judgements.

4.1.1 Identifying Communities of Interest

The scenarios were selected from communities of interest where Web services are used as integration technologies. Our intention is that they form a spectrum of Web service uses. Starting from the low end of requirements and complexity, these communities are: Web mashups, Enterprise Services, Business to Business (B2B), Cloud Computing and Grid Computing. These domains are defined in Table 4.

Table 4 Communities of interest definitions

Community	Definition
Mashups	Mashups are applications that combine APIs and data sources to form new applications or new data sources (O'reilly, 2005).
Enterprise Services	Enterprise Services are concerned with integrating different systems within an organisation, with the objective of enabling independent evolution of these components (Fremantle *et al.*, 2002).
Business to Business	Business to Business (B2B) services aim to offer the ability of sharing information and performing business transactions between businesses on the Web (Kreger, 2003).
Cloud Computing	Cloud computing offers software, platforms and infrastructures as services to clients who pay to lease them. The services are dynamically scalable (Armbrust *et al.*, 2009).
Grid Computing	Grid Computing in general is concerned with enabling the utilisation of distributed and heterogeneous resources to provide a seamless platform for computational or data intensive applications. This platform can be used to enable remote collaboration and expensive instrument sharing (Foster *et al.*, 2002).

4.1.2 Selecting the Scenarios

The scenarios were selected according to the following criteria: they should be real scenarios, representative of the communities, and exist in the literature.

Scenarios in research papers can be real existent scenarios or hypothetical ones. The real scenarios are usually found in papers discussing experiences in developing a system. However there are also scenarios in the literature that are hypothetical motivating scenarios, tailored to highlight certain aspects of technological solutions.

A total of seventy research papers in the five communities of interest were reviewed. The papers were found by searching in Google Scholar for the keywords "scenario", "case study", "Web service" and the name of the community of interest. Out of the search results, seventy research papers were selected that appeared to contain a scenario or case study.

Table 5 Number of reviewed papers in each community of interest

Community of interest	# of reviewed papers
Mashups	14
Enterprise Services	14
Business to Business	14
Cloud Computing	13
Grid Computing	15

Out of those papers it was possible to find three or four real scenarios or case studies, in each community. For mashups and Cloud Computing only three scenarios were found in the literature. To provide a fourth scenario for mashups, the "Yahoo Finance Stock Quote Watch List", one of the featured pipes on Yahoo Pipes was selected. For Cloud Computing, a case study of the Google App engine LingoSpot was selected. This resulted in a total of 20 scenarios, listed in Table 4.

Table 6 List of Selected Web service Scenarios

Community of interest	Scenarios
Mashups	**M1:** Stock Quote Watch, Yahoo Pipes (Donnelly, 2010) **M2:** The MashMaker Scenario (Ennals and Garofalakis, 2007) **M3:** Displaying the time and location of a website's visitors using a layered mashup architecture (Biornstad and Pautasso, 2009) **M4:** Creating situational applications using the enterprise information mashup fabric. (Jhingran, 2006)

Community of interest	Scenarios
Enterprise Services	**E1:** SSPD (City University) (City University, 2008) **E2:** MLE (City University) (City University, 2008) **E3:** BT.com (Integrating BT's OSS) (Calladine, 2004) **E4:** SCORe (Integrating BT's OSS) (Calladine, 2004)
Business to Business	**B1:** Reverse Auctioning Service (Decker and Weske, 2007) **B2:** Telecommunications Wholesaler (Zimmermann *et al.*, 2005) **B3:** E-Procurement (Brodie, 2000) **B4:** Supply Chain Management (Preist *et al.*, 2005)
Cloud Computing	**C1:** New York Times "Times Machine" (Klems *et al.*, 2008) **C2:** MLB Website's Chat system (Klems *et al.*, 2008) **C3:** Colorado State University using Google Apps (Herrick, 2009) **C4:** LingoSpot a business built using Google App Engine[8]
Grid Computing	**G1:** NEESgrid: Grid Based System for the Earthquake Engineering Domain (Gullapalli *et al.*, 2004; Pearlman *et al.*, 2004) **G2:** DAME Distributed Aircraft (Jackson *et al.*, 2003; Austin *et al.*, 2005; Jackson *et al.*, 2006; Jackson *et al.*, 2005) **G3:** Virtual Screening with Desktop Grids (Chien *et al.*, 2003) **G4:** ChombeChem testbed on the Grid (Frey *et al.*, 2003; Taylor *et al.*, 2006)

4.1.3 Scenario example

B1: Reverse Auctioning Service (Decker and Weske, 2007) is selected as an example, the 20 scenarios are detailed in Appendix B.

> ***"A buyer (e.g., car manufacturer) uses reverse auctioning for procuring specially designed components. In order to get help with selecting the right suppliers and organizing and managing the auction, the buyer outsources these activities to an auctioning service. The auctioning service advertises the auction, before different suppliers can request the permission to participate in it. The suppliers determine the shipper that would deliver the components to the buyer or provide a list of shippers with different transport costs and quality levels, which the buyer can choose from. Once the auction has started, the suppliers can bid for the lowest price. At the end, the buyer selects the supplier according to the lowest bid. After the auction is over, the auctioning service is paid."***

[8] Google App Engine, App Engine Developer Profiles
http://code.google.com/appengine/casestudies.html

4.2 Scenario analysis

To analyse the 20 scenarios we collected, two questions were asked: 1) How to model a resource-oriented interface, mapping to an ontology, which can be used by applications or agents to achieve functionality expressed in the scenario. 2) Do similarities or patterns emerge?

To answer these questions, the requirements of the scenarios were elicited, and then the scenarios were abstracted as resource-oriented models. Resource-oriented modelling is an approach developed to represent the resources in the interface, their relationships and interactions. The approach undertaken for eliciting the requirements and resource-oriented modelling are explained below with an example.

4.2.1 Eliciting requirements

This was the first step when analysing the scenarios, it involved capturing a high level view of the requirements and abstracting them from the descriptions of the scenarios. It involved looking at issues such as: the existence of different client roles, the different systems involved, and are those systems managed by the same entity.

As an example, these are the requirements from the one of the B2B scenarios, B1: Reverse Auctioning Service, from Table 6:

1. Registration - The auctioning service deals with many participants/clients that need to register before using the service. This implies the need for authentication and authorisation.

2. Support for different client roles - There are two different roles for users of this service: buyers and suppliers.

3. The service provider and the service consumers are different entities
The service provider is the auctioning services, and the consumers are the buyer and the suppliers.

4.2.2 Resource-Oriented Modelling

Resource-Oriented Modelling is a novel approach, developed specifically for this analysis (Alowisheq *et al.*, 2011), devised to offer a formal and unified method to facilitate the scenario analysis. In this modelling approach, resources are key actors in the interfaces, in contrast to other approaches where services, messages

or objects have primacy. It aims to provide a more intuitive mapping from model to implementation than could be achieved with non-resource focused methods. Resource-Oriented Modelling is based on re-purposing UML Collaboration Diagrams.

4.2.2.1 UML Collaboration Diagrams

The UML collaboration diagram is one of the UML interaction diagrams. As well as showing the interaction between objects, it focuses on the structural organisation of these objects. Therefore it can model static and dynamic aspects of the system which correspond to the structural relationships between objects and the behaviour and exchanged messages, respectively. The following figure is a collaboration diagram taken from (Booch *et al.*, 1999)

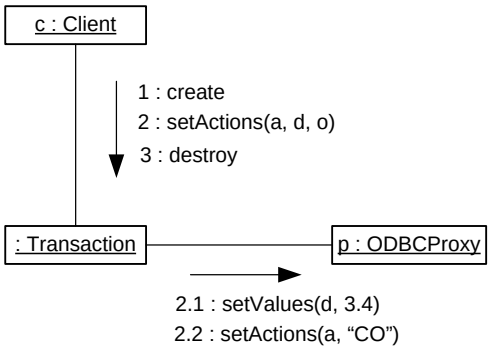


Figure 7 Collaboration Diagram

It shows objects exchanging messages. The arrows are messages, the sequence number on the arrows indicates the time order of messages, where 2 is the 2nd message and 2.1 is a message nested in 2. Both the arrows and the sequence numbers show the behaviour of the system. The links between the objects show the structural relationships, such as associations, aggregations, compositions and dependencies.

4.2.2.2 Collaboration Diagrams for RO Modelling

When building ROA and RESTful Web services, what is being created is an interface for clients, not a complete system; therefore our modelling approach focuses on the interface. The interface is formed by the resources that the server exposes to the client. The client is not modelled as a resource; however, messages that have no initiator are considered to be from the client. In our modelling approach, resources take the place of objects in collaboration diagrams.

According to ROA, these resources have a uniform interface: they can be created, read, updated or deleted, so the messages are restricted to these four actions.

Creation of resources in ROA is achieved by sending a POST request to a factory resource and, in UML terms, these can be considered as classes. In the original UML collaboration diagrams, there were no classes only objects, but later versions introduced specification level modelling that showed the structural relationships between classifiers. We do not take that approach, and instead represent factory classes as resources. This is because factory resources are not abstract in ROA but are actual elements that participate in the interaction. They need to be included so that both the static and dynamic aspects can be modelled.

The example in Figure 8 shows a simple example of the RO modelling approach. It models the B1: Reverse Auctioning Service scenario, which can be broken down into the following steps:

(1.) The buyer creates an auction
(2.) The buyer starts the auction
(3.) The suppliers place their bids
(4.) The buyer selects a bid
(5.) The buyer pays for the service
(6.) The buyer deletes the auction

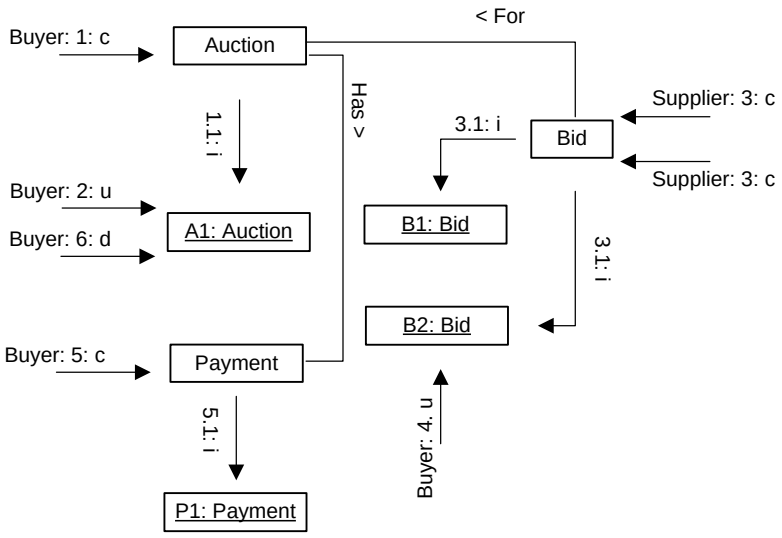


Figure 8 RO Diagram for B1: Reverse Auctioning Service

The letters c, u, d, and i on the messages respectively correspond to create, update, delete and instantiate. The links labelled `Has` and `For` are structural links that show how the resources relate to each other. We show the structural links between factory resources and non-insatiable resources, which can only be created by the server. The rationale behind this is to facilitate eliciting domain ontologies. Below is the domain ontology for B1.

```
:Auction a owl:Class.
:Bid a owl:Class;
:Payment a owl:Class.
:For a owl:ObjectProperty;
      rdfs:domain :Bid;
      rdfs:range :Auction.
:Has a owl:ObjectProperty;
      rdfs:domain :Auction;
      rdfs:range :Payment.
```

This ontology contains the classes and object properties in the scenario, however data properties would still needed to be added.

The advantages of Resource-Oriented Modelling stem from it being a more natural way to represent REST and ROA solutions, hence allowing designs to be more easily mapped to solutions. This is because it provides a simple mechanism for eliciting domain ontologies and captures both dynamic and static aspects of the interface.

4.2.3 Outcomes of the Scenario Analysis

The RO models for the twenty scenarios were created (they are included along with the Scenario descriptions in Appendix B). This was done by abstracting the interface as resources, then deciding on the structural relationships between these resources and the interactions needed to reflect the main success scenario in those scenarios.

The underlying assumption in the analysis was that resources in the scenarios were semantically described in domain ontologies, and these ontologies can be accessed by clients. Therefore, the analysis focused on finding other interaction requirements the client needed to be aware of (i.e. expressed to the client) in order to interact with the resources to achieve the scenario. The resulting interaction requirements are listed and explained below:

1. Mutability: Is the interaction with the resources to retrieve information or to update them? Most of the interactions in the scenarios were presented as updating resources. Table 7, where analysis results are compiled, shows that out of 128 interactions 89 were updating compared to 39, which were information retrieval.
   HTTP offers four main methods for interacting with resources. These are GET, PUT, POST and DELETE. A GET on a resource would be for information retrieval, and the other three methods modify, create and remove resources,

respectively. The combination of the HTTP method and the URI of the resource should be sufficient to semantically describe to the client how to formulate the request and the effects of the interaction. Therefore, there needs to be a mechanism where both the HTTP method and the URI are presented to the client prior to the interaction.

2. Atomicity: Is the interaction atomic or conversational? Conversational interactions are where the client follows a certain order of interactions to achieve the business logic. Conversational interactions are made up of several atomic ones. Conversational interactions can be achieved in HTTP by providing the client with links to the possible next steps. Out of the twenty scenarios, seventeen were conversational interactions.
3. Synchronisation: Is the interaction with the resources synchronous or asynchronous? An aspect of interaction that needs to be expressed to the client, is when the response is not immediate, e.g. a running job, hence asynchronous. The default mode of interaction is a synchronous request-response mode. This shows in the analyses of the scenarios, as only 34 out of the 128 interactions were asynchronous. Asynchronous interaction in HTTP can be achieved either through polling or pushing (notification). In polling, the client checks if the processing is completed at set intervals; using HTTP, the server can be made to respond with the status code "202 Accepted", which means that "The request has been accepted for processing, but the processing has not been completed.". However achieving pushing (notification) is not natively supported by HTTP. The analysis of the scenarios shows that in 6 interactions notification was required, compared to 26 where polling sufficed.
4. Plurality: Do the resources represent collections? In the twenty scenarios there were nineteen interactions with resources that represented collections. Five out of those nineteen can be represented as dynamic filters on collections, where the client provides values for properties on which the collection is filtered. The method for expressing these types of interactions is discussed in Chapter 5.
5. Roles: Are the types of interactions permitted by the server similar for every client, or are there different client roles? In the analysed scenarios, six out of the twenty scenarios had different client roles: for example, in the B1: Reverse Auctioning scenario, there were two types of clients: suppliers, and buyers.
6. Resource Representation: This is a fundamental feature of RESTful approaches, because interacting with the resources is performed by exchanging representations of resources.

Table 7 Interaction requirements of scenarios across communities of interest

Mutability	Atomicity	Synchronisation	Plurality	Roles

	Information Retrieval	Updating	Conversational	Polling	Notification	Collection	Filtered Collection	
Mashups	12	13	3	3	1	4	2	0
Enterprise Services	6	6	2	0	1	1	0	2
B2B	8	25	4	2	0	5	1	1
Cloud Computing	2	18	4	9	0	3	0	0
Grid Computing	11	27	4	12	4	6	2	3
Total	39	89	17	26	6	19	5	6

Table 7 shows a summary of the number of times the interaction requirements appeared in the five communities. Note that the sixth requirement is a fundamental requirement of RESTful approaches and is present in all the scenarios therefore it does not appear in the table. The full analysis per scenario is included in Appendix B.

4.3 SWS Approaches and Interaction Requirements

This section reflects on the 27 SWS approaches reviewed in Chapter 3, and asks if and how they support the six interaction requirements presented in the previous section. Table 8 shows the approaches and which requirements they fulfil.

Table 8 SWS approaches and interaction requirements

Publication	Purpose	Requirements					
		Resource Rep.	Mutability	Plurality	Atomicity	Synchronisation	Roles
OWL-S (Martin *et al.*, 2004)	Generic	*	x	*	✓	*	x
WSMO (Bruijn *et al.*, 2005a)	Generic	*	x	*	x	*	x
SAWSDL (Farrell and Lausen, 2007)	Generic	*	x	*	x	*	x
WSDL-S (Akkiraju *et al.*, 2005)	Generic	*	x	*	x	*	x
SWSF (Battle *et al.*, 2005)	Generic	*	x	*	✓	*	x
DSD (Klein *et al.*, 2005)	Generic	*	x	*	x	*	x
SA-REST (Lathem *et al.*, 2007)	Generic	*	✓	*	x	x	x
hRESTS (Kopecky *et al.*, 2008)	Generic	*	✓	*	x	x	x
MicroWSMO (Kopecky *et al.*, 2008)	Generic	*	✓	*	x	x	x
WSMO-Lite (Vitvar *et al.*, 2007)	Generic	*	x	*	x	*	x
RESTfulGrounding (Filho and Ferreira, 2009)	Generic	*	✓	*	✓	x	x
ReLL (Alarcon and Wilde, 2010)	Data Int.	*	✓	✓	✓	x	x
SBWS (Battle and Benson, 2008)	Generic	*	x	*	x	*	x
SPARQL descriptions (Sbodio *et al.*, 2010)	Generic	*	x	*	x	*	x
LIDS (Speiser and Harth, 2011)	Data Int.	*	x	x	x	x	x
LOS (Krummenacher *et al.*, 2010)	Generic	*	✓	x	x	x	x

Publication	Purpose	Requirements					
		Resource Rep.	Mutability	Plurality	Atomicity	Synchronisation	Roles
Semantic REST (Battle and Benson, 2008)	Generic	✓	✓	✓	x	x	x
Zhao and Doshi (2009)	Generic	✓	✓	✓	x	x	x
Hernandez and Garcia (2010)	Generic	✓	✓	x	x	✓	x
TSC (Riemer *et al.*, 2006)	Generic	✓	✓	*	*	✓	✓
RESTdesc (Verborgh *et al.*, 2011)	Generic	*	✓	x	✓	x	x
iServe (Pedrinaci *et al.*, 2010b)	Generic	*	✓	x	x	*	x
SADI (Wilkinson *et al.*, 2009)	Bioinformatics	*	✓	x	x	✓	x
HyperData (Kopecky *et al.*, 2011)	Generic (LD)	✓	✓	✓	✓	x	x
Hypermedia RDF (Kjernsmo, 2012)	Generic (LD)	✓	✓	x	✓	x	x
RDF-REST (Champin, 2013)	Generic (LD)	✓	✓	✓	✓	✓	x
SSWAP (Gessler *et al.*, 2009)	Bioinformatics	✓	✓	x	x	x	x
✓: addressed by the approach **x**: not addressed by the approach *: assumed existing & addressed by other layers							

In SWS that are classified as service-oriented meta-models the resource representation requirement is addressed by other layers typically the domain ontology which describes the resources manipulated by the service, and in approaches that have WSDL groundings, such as OWL-S, WSMO, SAWSDL, WSDL-S, SWSF, DIANE, WSMO-Lite, the resources manipulated are also described in WSDL types. This is also the case for plurality.

As for mutability, approaches that described RESTful Web services, whether they adopted RO or SO meta-models, all provided mechanisms for specifying which HTTP method to be used.

Regarding atomicity this is either fulfilled by providing the capability to describe composite services such as in OWL-S, SWSF and RESTful Grounding, and Hernandez & Garcia or providing methods for guiding the clients to the next state as in ReLL, RESTdesc, Hyperdata, Hypermedia RDF and RDF-REST.

Synchronisation can be targeted in other layers such as Message Exchange Patterns (MEP) in WSDL/SOAP based services, however it is ignored by most RESTful and RO approaches, with the exception of Triple space approaches such as TSC and Hernandez & Garcia. RDF-REST and SADI address this by providing "202 Accepted" status codes.

The roles requirement is addressed by only one of the SWS approaches, TSC. The triple space provides the definition of roles an permissions. Possibly, the reason

that the roles interaction requirement is ignored by the SWS frameworks is that these are regarded to be application-specific.

RDF-REST fulfils all the requirements except roles because it proposes to implement the Linked Data Platform (LDP), which targets these requirements in the interface descriptions.

4.4 Conclusions

This chapter set out to conceptualise Web service scenarios as RESTful interactions with resources, and to understand their requirements as a result of this conceptualisation.

It presented the compilation and analysis of a total of twenty representative Web service scenarios from five communities of interest. RO models were introduced to aid in the analysis and abstract resources in the interaction. The underlying assumption was that the resources were semantically described in domain ontologies; therefore the aim was to investigate other aspects that needed to be expressed in the interface, so that the client can interact with the interface to fulfil a specific scenario, and how to achieve those using only REST and the domain ontology.

The need for five main interaction requirements emerged from the analysis. These are mutability, atomicity, synchronisation, plurality and roles, in addition to the underlying requirement assumed during the analysis, which is resource representation. These requirements were used to reflect on the SWS approaches reviewed in Chapter 3.

These requirements informed the design of the proposed RESTful SWS approach, EXPRESS, and the next chapter discusses how they are implemented in EXPRESS.

Chapter 5: EXPRESS: EXPressing REstful Semantic Services

This chapter introduces EXPRESS, an approach to both describing and providing RESTful Semantic Web services.

EXPRESS eliminates explicit service descriptions and vocabularies, and shows how RESTful Semantic Web services can be created semi-automatically by combining the expressivity and semantics in ontologies and providing a uniform interface for them. This requires the conceptualisation of problems as interactions with semantically described resources, rather than services. So instead of semantically describing a temperature service as a service that takes a location as input and returns the degree as output, it is conceptualised as a temperature resource that is filtered by a location. The difference is subtle, but this chapter shows how it enables the elimination of explicit service descriptions and vocabularies.

Section 5.1 presents an overview of EXPRESS and a simple example of how it works; Section 5.2 shows how EXPRESS describes and provides the interaction requirements discussed in Chapter 4, Section 5.3 presents a proof-of-concept demonstrator for EXPRESS that shows how RESTful Services can be provided semi-automatically, and Section 5.4 discusses how EXPRESS compares to other SWS approaches.

5.1 Overview of EXPRESS

The processes of semantically describing and providing services in EXPRESS are intertwined and are undertaken in six steps. In the first of these, the developer provides a domain ontology that describes the resources in the interface. All the steps are illustrated in Figure 9 below.

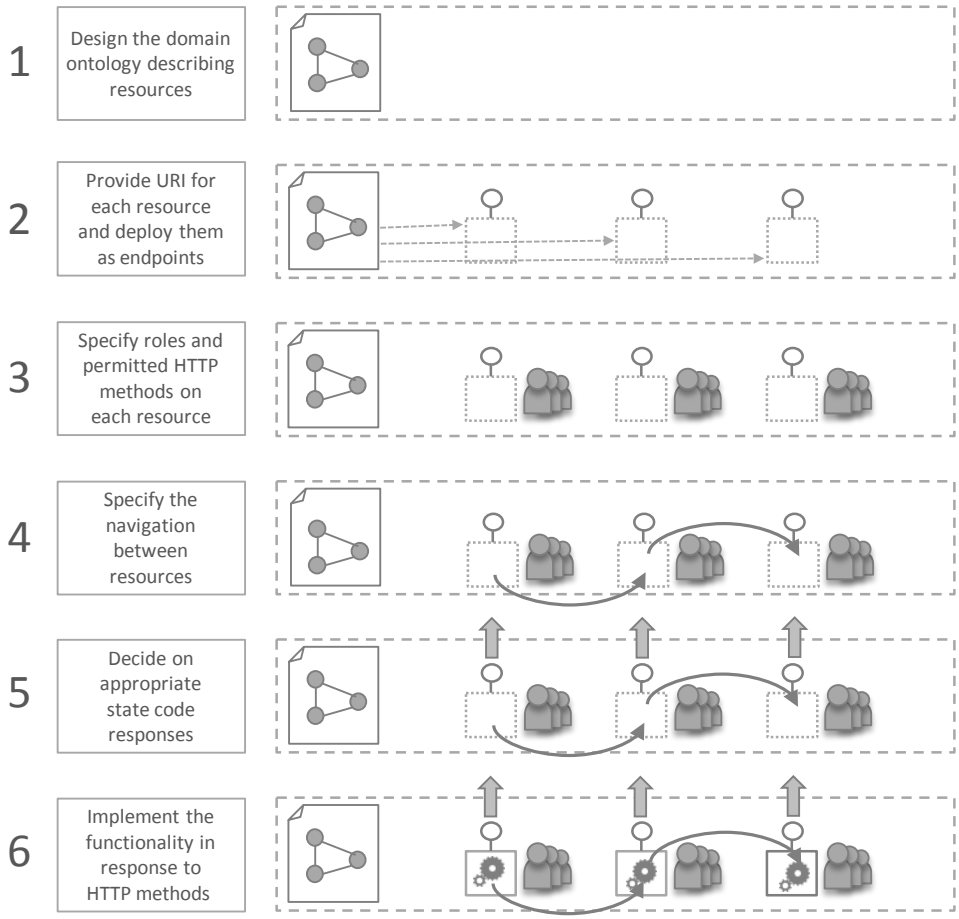


Figure 9 Steps for describing and providing a RESTful interface in EXPRESS

A Simple Example

Following is a simple example to demonstrate the primary concepts in EXPRESS. This example presents a bookstore Web service. The bookstore wants to enable the ordering of books, and is referred to as the service provider. There are two types of clients: customers and an independent delivery service.

(Step 1) The service provider needs to provide an ontology describing entities it wants clients to deal with. In this case they are: book, order, and person. The following listing describes the relevant parts of the ontology formatted in N3

```
:Book              a     owl:Class.
:title             a     owl:DatatypeProperty;
   rdfs:domain     :Book;
   rdfs:range      xsd:string.
:author            a     owl:ObjectProperty;
   rdfs:domain     :Book;
   rdfs:range      :Person.
:Person            a     owl:Class.
:isbn              a     owl:DatatypeProperty;
```

```
    rdfs:domain     :Book;
    rdfs:range      xsd:string.
:Order              a     owl:Class.
:containsItem       a     owl:ObjectProperty;
    rdfs:domain     :Order;
    rdfs:range      :Book.
 :orderedBy         a     owl:ObjectProperty;
    rdfs:domain     :Order;
    rdfs:range      :Customer.
 :creationdate      a     owl:DatatypeProperty;
     rdfs:domain    :Order;
     rdfs:range     xsd:dateTime.
 :Customer          a     owl:Class.
 :hasAddress        a     owl:DatatypeProperty;
     rdfs:domain    :Customer;
     rdfs:range     xsd:string.
```

(Step 2) The OWL file is used to create a RESTful interface for the resources. The file is parsed; classes, properties and individuals are given URIs based on their names in the file. The following are examples of generated URIs.

```
http://bookstore.com/Book  (URI for a class)
http://bookstore.com/Book/DBSys  (URI for a book instance)
http://bookstore.com/Order/Or11233  (URI for an order instance)
```

The book's properties also have URIs, for example the book's title has this URI

```
http://bookstore.com/Book/DBSys/title
```

The URIs are designed to include the types of the requested resources as shown above; this is consistent with the W3C note on cool URIs[9].

(Step 3) The service provider then states, via mechanisms later discussed in the chapter, which methods (GET, PUT, POST and DELETE) can be applied to each URI. If the server provider has several types of clients, it can state that permitted methods on a URI differ depending on what type of client is accessing it.

The interface is deployed after specifying the access control lists, as stubs are automatically created.

(Step 4) In this case the interaction is conversational, so the client would be guided by the server on which links to follow next. This means that, in the implementation, the developer would specify that when a book is retrieved, a link to order a book is presented to the client.

[9] Cool URIs for the Semantic Web, W3C, http://www.w3.org/TR/cooluris/

(Step 5) The developer does not need to specify default HTTP response codes, such as 200 OK or 201 Created, when retrieving or creating a resource respectively, which is the case in this example; however, to make the interaction synchronous the developer would need to change the response. This is explained further in Section 5.2.5.

(Step 6) The service provider maps these stubs to existing services or codes the business logic in them.

To illustrate how a client customer interacts with the interface to place an order, we assume the client has already discovered the service[10]. For the client to invoke the service it needs to have the OWL file. It can access this from the service in the same way it `GET`s any other resource.

The purpose of the OWL file is to show the resource representation and thus the exchanged messages format, relationships, and special instances. The client also needs to know how to invoke HTTP methods on resources. After the client has got the OWL file, to place an order it sends a `POST` request to

```
http://bookstore.com/Order
```

with the following payload

```
_:a223     a       :Order;
     :containsItem <http://bookstore.com/Book/DBSys>;
     :hasTime "2013-04-23T11:19:35"^^xsd:dateTime;
     :orderedBy :c1245.
```

The server will respond by creating a new order and sending back its URI to the client. For example `http://bookstore.com/Order/Order11233`. The `orderedBy` property indicates which customer placed the order.

As an example of how role-based access control (RBAC) is applied, on the URI

```
http://bookstore.com/Order/Order11233/hasStatus
```

customer clients can only invoke `GET`. The delivery service, which is also a client of the bookstore service can invoke `GET` or `PUT` to modify the status, but cannot modify other `Order` properties, however customers can. This is explained further in Section 5.2.6.

[10] Another assumption is that the client knows the URI of the book it wants to order. This example is extended in Section 5.2

5.2 Semantic Description

This section shows how EXPRESS provides the interaction requirements discussed in Chapter 4. These interaction requirements are first listed in Table 9, which states the means of description or provision and the step in which they occur (as illustrated in Figure 9).

Table 9 Interaction requirements and the step in which they are expressed

Interaction requirement	Means						Step					
	Domain ontology	Resource URI	HTTP Methods	Links	HTTP Status Codes	RBAC	1	2	3	4	5	6
Resource Representation	✓	✓					✓	✓				
Mutability		✓	✓					✓	✓			
Plurality		✓						✓				
Atomicity				✓						✓		
Synchronisation					✓						✓	
Roles						✓			✓			

5.2.1 Resource Representation

The main argument of this thesis is that the resource representation in the domain ontology and the standard interface are sufficient to describe the required functionality. Therefore the resource representation requirement plays the main role in the design of EXPRESS approach, and it is the foundation upon which the other five requirements stand, hence a thorough explanation of the resource representation requirement is substantially longer than the other requirements.

Referring to the bookstore example in the previous section, the representation of a `Book` individual is as follows:

```
<http://bookstore.com/Book/DBSys>  a :Book;
     :isbn      "0123735564"^^xsd:string;
     :title     "Database Systems"^^xsd:string;
     :author    <http://bookstore.com/Person/JSmith>.
```

The resource representation is constructed from the specified properties of the resource and their values. In EXPRESS `rdfs:domain` that links a property to class, specifies that individuals of the class would have those associated properties in the representation, and the types of those properties would depend on the `rdfs:range` statements in the ontology.

Typically, the server is responsible for minting URIs for newly created resources. In a case where the client is creating a new resource such as the `Order` in Section 5.1, EXPRESS requires the client to send URIs as blank nodes (bNodes), then creates the resource and sends the URIs back to the client.

This section explains the resource types, their representations, and the rationales for the design decisions. In a RESTful interface, clients and service providers interact by exchanging resource representations. Specifying the resource representation is important because it sets restrictions on the exchanges between the server and the client, this establishes a common language that manages expectations and hence enables validation and facilitates future automation of the interaction.

5.2.1.1 Resource Types

The messages exchanged in EXPRESS are in RDF. The interface is described by the domain ontology, which can contain several resource types. The main resource types are classes, individuals, or properties of individuals. Each one of these resources types has a different URI pattern that corresponds to a graph pattern (shown in Table 10). This graph pattern, together with the domain ontology, dictate the format of the resource representation.

The aim is to enable automated generation of URIs endpoints and server-side stubs from the description of resources in the ontology.

Table 10 Resource types and corresponding URI and graph patterns

Resource Type	URI Pattern	Corresponding Graph Pattern
Class	`/AClass`	`?x a AClass` `?x ?p ?o`
Individual	`/AClass/Individual`	`Individual a AClass` `Individual ?x ?y`
Object Property	`/AClass/Individual/Property`	`Individual Property ?x` `?x ?p ?o`
Data Property	`/AClass/Individual/Property`	`Individual Property ?x`

Filtered Individuals	`/AClass?Property={`*value*`}`	`?x a AClass`
		`?x ?p ?o`
		`?x Property value`
		`value ?y ?z`

Properties of Filtered Individuals	`/AClass/Property1?Property2={`*value*`}`	`?x a AClass`
		`?x Property1 ?y`
		`?x Property2 value`
		`?y ?p ?o`

The six resource types in Table 10 are explained in further detail below.

1. **Class**

This resource type serves as a factory endpoint for creating new individuals of this class, or listing existing ones. Having a factory endpoint for creating resources is a well-known convention in the design of Web applications and Web API, and the URI for such an endpoint typically ends with the name of the class or resource type. For example in the Facebook Graph API[11], the following endpoint is used to add photos

```
/{album-id}/photos
```

To form the newly added photo's URI, the photo ID will be appended to the URI above, and EXPRESS follows the same convention, by having the URI pattern representing a class end with the class name.

The corresponding graph pattern aims to match any individual of this class, together with the individual's properties. The notion of properties here is also influenced by object-oriented design. That is the properties assume a direction; therefore, in the design of the ontologies for EXPRESS the developer should define the domain and range of the property. This is one of the requirements EXPRESS imposes on the design of ontologies. Thus when manipulating an individual of this class, or returning its properties, only properties which have been defined to have this class as a domain will be considered as part of the result. This functions to manage server and client expectations.

2. Individual

This resource type represents an individual of a class, the corresponding URI pattern is also in line with cool URIs and conventions and practices in the design of Web APIs. When a resource is created of a certain type, its URI is formed by appending its ID to the URI of the Class it belongs to. The corresponding graph pattern represents a single resource as well as its associated properties and their values.

3. Object Property

The object property resource type accesses the values of object properties for a certain individual. This fine-grained access allows the client to retrieve or manipulate a property of the resource, rather than the whole resource, thereby increasing the efficiency of the interaction when resources are large, and enabling different levels of access control over resource properties.

The URI pattern of an object property, since it accesses part of a resource, becomes an extension to the resource's URI and takes the following form, as shown in Table 10:

[11] Facebook Graph API https://developers.facebook.com/docs/graph-api/using-graph-api/v2.1

```
/AClass/Individual/Property
```

The corresponding graph pattern matches the individual which is the value of the property as well as its associated properties and their values.

4. Data Property

The data property resource type is very similar to the object property resource type, the only difference is in the corresponding graph pattern, which matches the triple connecting the individual to the value of the data property.

5. Filtered Individuals

This resource type represents individuals that are filtered by one or more property values. It is intended to provide an efficient mechanism for retrieving and creating resources. This is a factory endpoint, like the Class resource type.

The class resource type had two main functionalities:

1. To create a new individual of this class.
2. To provide access to all individuals of this class.

The Filtered Individuals resource type is a special case of the Class resource type. It enables both the creation and retrieval of individuals, however unlike the Class resource types, these individuals are filtered by property values during retrieval. Individuals created by this resource type can have certain property values specified by the client. Examples include a server generated ID, or a creation date. Let us assume that the `Order` resource in the bookstore has a creation date, which is created by the server. This is indicated to the client by specifying the all the properties needed to create the `Order` in the query string, and leaving out the properties that the server would create, as shown below.

```
Link: <http://bookstore.com/Order?orderedby={}&containsItem={}>;  rel="POST"
```

This would tell the client that values for both `orderedBy` and `containsItem` are required for creating an `Order`, and as a result an `Order` would be created, that has the client provided values for both `orderedBy` and `containsItem`, and a creation date specified by the server.

The URI pattern for the Filtered Individuals resource type is comprised of the name of the Class and a query string with name-value pairs for the filtering properties. This offers flexibility for defining endpoints of this type, so that several endpoints may exist to filter individuals by different combinations of

properties. Moreover the structure of the URI pattern, being a query string, differs from those of other resource types, so it is not confused with them.

The corresponding graph pattern matches individuals of the Class, that have the given value for the specified property, as well as those individuals associated properties and their values.

6. Properties of filtered individuals

In a case where the client only wants a value of a certain property for filtered individuals, such as titles of books by a certain author, it is inefficient to return all the properties of those individuals.

This follows the convention established in the previous resource type; however only the required property of the individual is returned, not all the properties of the individuals.

Therefore, the URI pattern for this resource type is similar to the Filtered Individuals URI pattern in terms of the query string and name-value pairs, but it differs in that it has the the required property before the query string. The corresponding graph pattern matches the individuals, their required properties and the properties used for filtering.

Property paths are a new feature in SPARQL 1.1 (Harris and Seaborne, 2013). They enable the specification of an arbitrary length route between two resources; triple patterns are paths of length 1.

For example

```
?order     :containsItem/:title  ?title
```

would return the titles of books in orders. Property paths make writing graph patterns more concise, allow resources connected by arbitrary length paths to be matched and support inverse paths where roles of subject and object are reversed.

The use of property paths is a potential future extension for EXPRESS, which would add greater flexibility to the introduced resource types. However this is currently out of the scope of this thesis. For example

```
/Order/{OrderID}/containsItem/title
```

would be the URI for the pattern above.

5.2.1.2 Resource Types in RO SWS Approaches

Having different resource types that exhibit differ in behaviour as a result of applying HTTP methods, or differ in their representation is common in the design of resource-oriented SWS. For example in Semantic REST (Battle and Benson, 2008) they had two main endpoint types, a class-level and a resource-level endpoint. Each has a different URI pattern. The URI form for a class-level resource is:

```
/ResourceType/
```

The URI form for a resource-level endpoint is:

```
/ResourceType/ResourceID
```

Moreover these endpoints accept SPARQL queries to be sent and resolved, in EXPRESS this is not allowed for security reasons, however basic filtering is offered by introducing the two other resource types Filtered Individuals, and Properties of filtered individuals. In EXPRESS, if advanced SPARQL queries are required they should be defined explicitly as a resource in the ontology.

Zhao and Doshi (2009) identified three types of resources, these are: resource set, individual resource and transitional service. Each of these types has an associated URI pattern. A resource set type, represents a collection of resources of a certain type therefore the HTTP methods applied to it will manipulate all individuals in the set. In addition this type of endpoint serves as a factory endpoint to create new individuals of this type. An individual resource represents one resource, and hence the HTTP methods affect a single resource. The third type is different, it is loosely defined, to encompass all functionally that does not map directly to manipulating sets, or individuals, and that is considered more transformation-oriented, or resources that update other resources. They provide examples such as ShipOrder, and SubmitPayment. EXPRESS's alternative for this, is to represent the functionality as an update of resource's property, this way EXPRESS provides a unified view of resources.

RDF-REST (Champin, 2013) proposes to implement the Linked Data Platform (LDP) (Speicher *et al.*, 2014). In LDP there is a notion of LDP Resources (LDPR) and LDP Containers (LDPC), these two types of resources respond differently to HTTP methods.

Hyperdata (Kopecky *et al.*, 2011) uses named graphs to represent API endpoints for resources in the RDF store. They have four types of resources: classes, individuals, property resources, and value resources. These are defined as named graphs, and in Hyperdata are considered as endpoints, which accept HTTP

methods. These definitions also include the resource description, so that the boundaries of the resources are defined.

In TSC approaches such as the one by Hernandez and Garcia (2010), they assumed that there were domain ontologies that define the classes of individuals, and that each triple space has a URI and corresponds to a certain class, therefore individuals that exist in a triple space, are all of the same class, and each one of those individuals had a specific URI. Thus the underlying assumption is that there are two different resource types (class and individual), which exhibit different behaviours when HTTP methods were applied.

SSWAP (Gessler *et al.*, 2009) on the other hand was developed for bioinformatics applications, and the functionality is conceptualised as mapping an input provided by the client to a related output provided by the service, therefore it defines only one resource type. The client can send `POST` request with the input to the endpoint URI, or perform a `GET` request with the input value appended to the endpoint URI by means of a query string.

These RO SWS approaches, used resource types to specify:

1. How the server would respond: Would it manipulate a list of resources, a single resource, or a part of a resource?
2. What the payload looks like: What does a resource contain and what are its boundaries, in other words what is the payload structure?

Three other RO SWS reviewed in Chapter 3 do not use resource types for the purposes above these are RESTdesc (Verborgh *et al.*, 2011), ReLL (Alarcon and Wilde, 2010), and Hypermedia RDF (Kjernsmo, 2012). In RESTdesc there is no RDF serialization of the resource representations; graph patterns represent resources that are necessary for the composition or discovery of the APIs. Therefore, those graph patterns provide a flexible way to define the expectations from the endpoint, but only the ones necessary to compose or discover them, not to represent the resource. In other words, the resource representation is left to the lower layers. ReLL is similar in this sense, where the resource representation is left to the schema and media types. Hypermedia RDF is a proposed vocabulary to make RDF a hypermedia type. The approach does not specify the repercussions of updating or deleting a resource. So in a sense the resource representation does not define the resource boundaries, therefore there are no resource types.

Ultimately, RO SWS can take two methods, either they impose general types of resources or endpoints with similar behaviours and rules for payload structures, or theyoffer more flexibility and define implications for each endpoint separately,

such as the three approaches discussed in the last paragraph. EXPRESS adopts the first method. The whole purpose of resource types in EXPRESS is to strike a balance between imposed restrictions and generality so that several applications can be fulfilled, while being easy for developers to understand the concepts.

5.2.1.3 EXPRESSive Ontologies

As explained above EXPRESS utilises mechanisms in the ontology itself to represent requirements in the interaction. This imposes assumptions on the design and the interpretation of the ontology. An example discussed above was the specification of the domain and range for each property. Two other assumptions are explained below. This requires designing the ontology with EXPRESS in mind. EXPRESS also assumes ontologies are in OWL DL, however since no reasoning is required at this stage, the OWL profile of less importance.

1. Potential addition of new concepts

In service-oriented approaches to SWS, domain ontologies are mainly used to specify inputs and outputs for the services. A resource-oriented approach requires a different conceptualisation of the problem, as any resource the client may interact with would need to be specified. For example in a resource-oriented approach if you want customers to be able to order books you need to have an Order class, whereas in a service-oriented approach there would typically be a Book Order service, described using a service ontology, this service may have a book as input and an order ID as output. For that reason a resource-oriented approach such as EXPRESS may require the addition of new concepts to the domain ontology.

2. Alignment to classes and properties in popular ontologies or vocabularies

Although this step is not necessary for an ontology to become an EXPRESSive ontology (and is usually a part of designing any ontology) it serves the purpose of service matchmaking in EXPRESS. Using `owl:equivalentClass`, and `owl:equivalentProperty` enables linking the definitions of classes and properties in an EXPRESSive ontology to other ontologies. For example, consider the `Book` class, from the example used earlier in the chapter

```
:Book   a  owl:Class;
                owl:equivalentClass  dbpedia:Book.
```

The Book class is now mapped to the `dbpedia:Book` class.

5.2.1.4 Open Issues

There are two open issues related to resource representation, solutions for these issues are suggested below. For a client to interpret and interact with EXPRESS services autonomously these issues must be further explored.

1. Which resource properties are required from the client and which are optional?

This could be defined using the cardinality restrictions in OWL, specifically owl:minCardinality. For example if `author` was an optional property for `Book`, it could be expressed as follows:

$$Book \sqsubseteq \geq 0\ author$$

which is a cardinality restriction that at least zero authors are required for a book.

2. Which resource properties link to sub-objects or dependent ones (weak entities)?

Taking for example, a book's author, and assume that the server would allow clients to create books. The client would be allowed to create an author individual when creating the book, these resources would not have been created yet, and would be sent as bNodes: then the server would create them and send back their URIs. However if the client wishes to link to an existing author, it can provide their URIs instead, and the server would understand not to create them. So what if the server would not allow the client to create a `Book` without having an `Author`. How would that be conveyed to the client? This could be conveyed using OWL restrictions. For example:

$$Book \sqsubseteq \exists\ author.Person$$

This would tell the client that an author would need to exist, before creating a Book.

However the issue with using restrictions, either existential (such as the one above) or cardinality restrictions (such as in point 1) are not enforced in OWL, because its standard semantics adhere to Open World Assumption (OWA), therefore reasoners do not notify if an Book instance exists without having an author.

Therefore EXPRESS aware clients need to interpret these as restrictions and use other mechanisms to extract and deal with these restrictions accordingly. This is out of the scope of this thesis, and two potential solutions are discussed as future work in Section 8.4.

5.2.2 Mutability

Resources in EXPRESS can be created, read, updated and deleted; the offered functionality is indicated by which HTTP method the resource accepts: these are `POST`**,** `GET`**,** `PUT` **and** `DELETE`**. The effects of applying these methods to the resource types are shown in the following table.**

Table 11 Resource types and the effects of HTTP methods

Resource Type	GET	PUT	POST	DELETE
Class	Gets information about all individuals of this class	Creates a named individual: the client states the identifier	Creates an individual: the server decides the identifier	Deletes individuals of this class
Individual	Gets all properties of the individual	Updates individual's properties values	N/A	Deletes individual and its properties
Object Property	Gets the value of this property	Updates the value of this property	N/A	Deletes the relationship between the property value and the individual; the decision whether the value is deleted is left to the implementation
Data Property	Gets the value of this property	Updates the value of this property	N/A	Deletes the property value
Filtered Individuals	Gets individuals that have the given property value	Updates individuals that have the given property value	Creates individual(s) with the given property value	Deletes all individuals that have the given property value
Properties of Filtered Individuals	Gets property1, of all individuals that have the given value for property2	Updates property1, of all individuals that have the given value for property2	N/A	Deletes property1, of all individuals that have the given value for property2

In EXPRESS, as in ROA, the HTTP methods `POST`**,** `GET`**,** `PUT`**, and** `DELETE` **map to Create, Read, Update and Delete (CRUD), respectively. To be meaningful in the context of EXPRESS, the** `POST` **method, which creates new instances, can only be applied to the factory resource types: class and filtered individuals. In all the other cases,** `GET` **retrieves,** `PUT` **updates, and** `DELETE` **deletes the associated graph pattern represented by the resource type, as explained in Table 10.** `PUT` **is used for creating individuals only when applied to the Class resource type, this means the server permits the client to provide the identifier, which is consistent with ROA practices (Richardson and Ruby, 2007, p99, p220).**

It is possible to formalise each request as a SPARQL query. This formalisation provides a specification of the request's behaviour, or effects. To represent the `GET` **method SPARQL CONSTRUCT queries are used. To enable the representation of the** `PUT`**,** `POST` **and** `DELETE` **methods the SPARQL Update Language (Gearon *et al.*, 2013) specifically DELETE and INSERT operations are used.**

The mapping to SPARQL queries has the following assumptions:

1. **The service provider has an internal RDF graph named** `<Server>`
2. **The resource type and the HTTP method determine the SPARQL query**
3. **In** `PUT` **and** `POST` **the payload, is considered another RDF graph** `<Payload>`

The following example in

Table 12 illustrates the mapping to SPARQL queries, it maps the HTTP methods' effects on the "individual" resource type, shown in the second row in Table 11. The rest of the mappings are in Appendix C.

This example represents the mapping of HTTP methods into SPARQL queries on a book individual from the bookstore example in Section 5.1. The URI pattern and corresponding graph pattern for an individual is as follows.

URI Pattern

```
AClass/Individual
```

Graph Pattern

```
Individual    a    AClass;
Individual    ?x   ?y.
```

And in the case of a specific book, DBSys, this would be:

URI

```
http://bookstore.com/Book/DBSys
```

RDF Graph

```
<http://bookstore.com/Book/DBSys>  a     :Book;
         :isbn     "0123735564"^^xsd:string;
         :title    "Database Systems"^^xsd:string;
         :author   <http://bookstore.com/Person/JSmith>.
```

Table 12 Formalisation of HTTP methods in SPARQL queries for a book individual

GET	
Description	Retrieves information about DBSys at this URI http://bookstore.com/Book/DBSys
Corresponding SPARQL Query	`CONSTRUCT { <http://bookstore.com/Book/DBSys> ?p ?o } WHERE { <http://bookstore.com/Book/DBSys> ?p ?o }`
Result	`<http://bookstore.com/Book/DBSys> a :Book; :isbn "0123735564"^^xsd:string; :title "Database Systems"^^xsd:string; :author <http://bookstore.com/Person/JSmith>.`
Explanation	The CONSTRCUT query returns triples in the format specified by the graph pattern associated with the individual resource type (see Table 10), which returns the values of the associated triples.
PUT	
Description	Updating the ISBN of the book at this URI http://bookstore.com/Book/DBSys
Payload	`<http://bookstore.com/Book/DBSys>`

	:isbn "1334340005"^^xsd:string.

Corresponding SPARQL Query	WITH <Server>
	DELETE
	{ <http://bookstore.com/Book/DBSys> ?p ?oOld}
	INSERT
	{ <http://bookstore.com/Book/DBSys> ?p ?oNew}
	WHERE
	{ GRAPH <Payload> {
	<http://bookstore.com/Book/DBSys> ?p ?oNew }
	GRAPH <Server> {
	<http://bookstore.com/Book/DBSys> ?p ?oOld }}

Result	<http://bookstore.com/Book/DBSys> a :Book; :isbn "1334340005"^^xsd:string; :title "Database Systems"^^xsd:string; :author <http://bookstore.com/Person/JSmith>.
Explanation	To update an individual, this is mapped to a DELETE/INSERT operation, the payload contains the triples that specify the properties that will be updated and their new values. The DELETE/INSERT operation deletes from the server the triples that match the pattern : Individual ?p ?old But since there is a WHERE clause, this pattern also matches the triples provided in the payload. Therefore only triples containing properties provided in the payload will be affected in the server, and replaced by the triples provided in the payload which is the effect of the INSERT clause.
DELETE	
Description	Deletes the individual and associated properties.

Corresponding SPARQL Query	DELETE {
	GRAPH <Server> {
	<http://bookstore.com/Book/DBSys> ?p ?o. }}
	WHERE
	{ <http://bookstore.com/Book/DBSys> ?p ?o. }

Explanation	The triple <http://bookstore.com/Book/DBSys> ?p ?o. matches the individual and its properties at the server, and the DELETE operation removes those triples.

When designing Web services in EXPRESS, a developer specifies through the interface (explained in Section 5.3) which methods can be applied to which resources. The client discovers this from the HTTP Link Header when retrieving the ontology. Below are some examples:

```
Link:     <http://bookstore.com/Order?containsItem={}>;        rel="POST"
Link:     <http://bookstore.com/Book?isbn={}>;                 rel="GET"
```

The Link Header is explained in more detail in Section 5.2.4

Of course the client could know through sending an `OPTIONS` **request to a certain resource, but that would mean an extra roundtrip to the server for each interaction. It is more efficient to provide the client with the possible next actions as soon as it receives a response from the server, rather than blindly sending** `OPTIONS` **requests to resources to know what method is allowed.**

5.2.3 Plurality

EXPRESS provides multiple mechanisms to represent and manipulate collections. In Table 11, all the resource types except "Individual" can be used to represent collections. The "Class" and "Filtered Individuals" resource types represent factory endpoints for creating new individuals using the `POST` method. However, when applying `GET`, `PUT` or `DELETE` to them, these endpoints represent collections, and would affect all individuals which "Class" or "Filtered Individuals" represent. For example, if a client performs a `GET` on the following URI

```
http://bookstore.com/Book/
```

all instances of books at the bookstore would be returned. However, the functionality of returning all the books would not be likely to be provided by the bookstore. Instead there would be a mechanism to look up books by title or author. This is provided by the "Filtered Individuals" resource type. For example, performing a `GET` on the following resource

```
http://bookstore.com/Book?title="Database Systems"
```

returns books with the title "Database Systems".

The importance of whether a resource is a collection or not, is for managing client expectations, so the client should be prepared to deal with multiple individuals when performing `GET`, `PUT` or `DELETE` on the two resource types mentioned above, and multiple property values in the other three resource types, which are "Data Properties", "Object Properties" and "Properties of Filtered Individuals".

5.2.4 Atomicity

As explained in Chapter 4, most of the scenarios in the analysis were conversational, meaning the client interacted with the server in several steps to achieve the business logic. In RESTful applications the server guides the client by providing hypermedia controls (discussed in Chapter 2): these controls provide the resource location and state how it can be manipulated. In EXPRESS, a possible method for achieving this, without introducing new vocabularies, is to use the HTTP Link header.

The Link Header was in the HTTP/1.1 2068 1997 protocol (Fielding *et al.*, 1997), but was not specified in the later version HTTP/1.1 2616 1999 (Fielding *et al.*, 1999). However, it was argued for by Connolly and Hickson (1999), and more

recently in (Nottingham, 2010). Using the Link headers enables the linking of resources regardless of their representation format (i.e. serialisation).

To show how this is achieved in EXPRESS, an example is presented from the bookstore scenario mentioned in this chapter. When the client retrieves the ontology, it also receives, in the header, a link for the next possible action(s) and associated HTTP method.

```
HTTP/1.1 200 OK
Link: <http://bookstore.com/Book?isbn={}>;  rel="GET"
```

The client then knows that the next possible action it can take is to perform a `GET` on the following resource `/Book?isbn={}` .

EXPRESS repurposes the use of the link relations (rel) to specify the HTTP method. In RESTful applications such as APP (Gregorio and de hOra, 2007), possible values of link relations are defined in the media type specification, and are used not only to specify the HTTP method, but also the expectations in terms of payload structure (Webber *et al.*, 2010, p116). Since in EXPRESS the payload structure is specified by the resource type, what is left is the HTTP method.

In RESTful practices, link headers have been proposed to be used to fulfil the uniform interface constraint "hypermedia as the engine of application state" for media types that are not hypertext. In EXPRESS using link headers was one of three possible solutions:

1. Embedding the links in the RDF representations returned from the server. This would mean adding or using other vocabularies or ontologies to define the links, and EXPRESS actively avoids using or introducing interaction vocabularies.
2. Returning multipart messages from the server, the first part would be the RDF representation of the resource and the other would be in either HTML or ATOM containing the links. This would be a less elegant solution, due to the overhead of providing manipulating messages with different media types.
3. Using link headers.

Using Link headers is proposed to fulfil conversational services, (Appendix E provides an example of a conversational service that has been used in the expert reviews in Chapter 7) however the practicality of this approach has yet to be assessed. Section 8.4 discusses future work, which aims to provide automated conversational services and to use case studies to assess the practicality of

solutions. However, below is an open issue that would need to be addressed to achieve this.

Identifying which resources must be created first

For example, when creating an Order, the client should already have a created Customer, otherwise it would need to create one. In the interaction the server presents the client with the following options.

```
Link: <http://bookstore.com/Order>;       rel="POST"
Link: <http://bookstore.com/Customer>;    rel="POST"
```

Although this issue seems different than point two in Section 5.2.1.4, they are actually similar. In both cases the client would be allowed to create the related or required individual when creating the main one. So in the previous point, point 2, the client would send the author's information when creating a `Book`, and in this point, it would send the customer's information when creating the `Order`. As explained in point 2 these would be sent as bNodes: then the server would create them and send back their URIs. So what if the server would not allow the client to create an `Order` without having a `Customer`. How would that be conveyed to the client? This could be conveyed using OWL restrictions, as in point 2. For example:

$$Order \sqsubseteq \exists\, orderedBy.Customer$$

As explained before, restrictions are not enforced in OWL, because its standard semantics adhere to OWA, and potential solutions for this are discussed in future work in Section 8.4.

5.2.5 Synchronisation

Synchronisation is discussed in Chapter 4, and while there is no native support for notification in HTTP, polling can be achieved by implementing clients that interpret the HTTP code Accepted (202). This means that, in the resource implementation, if the response to the client would not be immediate (i.e. it needs processing) the server should return Accepted (202), and this would tell the client to try again later. In a case of a `POST`, when the resource needs processing before being created, the URI of this new resource would be returned in the location header. The client should be designed to poll this new URI at intervals using `GET` until it gets a Created (201) response from the server, with a representation of the newly created resource. This supports polling, but not pushing which is a one of the limitations in HTTP and consequently of EXPRESS.

5.2.6 Roles

EXPRESS enables simple yet fine-grained, Role-Based Access Control (RBAC). Service providers can specify which client roles are permitted to apply to which HTTP methods on which resource. In the bookstore example, a delivery service, (which is a bookstore client) was permitted to update the status of an order, so it was permitted to apply `PUT` to the following URI pattern

```
http://bookstore.com/Order/{OrderID}/hasStatus
```

However a customer was only permitted to apply a `GET`. In Section 5.3, the implementation of this requirement is discussed.

EXPRESS Design Principles

EXPRESS aims to take intuitive prevalent familiar conventions and map them into semantic structures. The design decisions aim to:

1. Minimise roundtrips to the server
2. Control granularity
3. Give resources cool URIs
4. Actively avoid adding interaction vocabularies, or ontologies, that either describe the resources or services.

5.3 EXPRESS Online Demonstrator

This section discusses the design and implementation of the EXPRESS deployment system. The deployment system aids in the creation of Semantic and RESTful Web services. The following figure illustrates the steps involved:

1. An OWL file describing entities in the existing system is given.
2. The deployment engine extracts resources from the OWL file and assigns URIs.
3. The roles and access control are specified on URIs and stubs are generated.
4. Stubs are connected to existing business logic, coded, or the code is generated.
5. Clients can access the Web service.

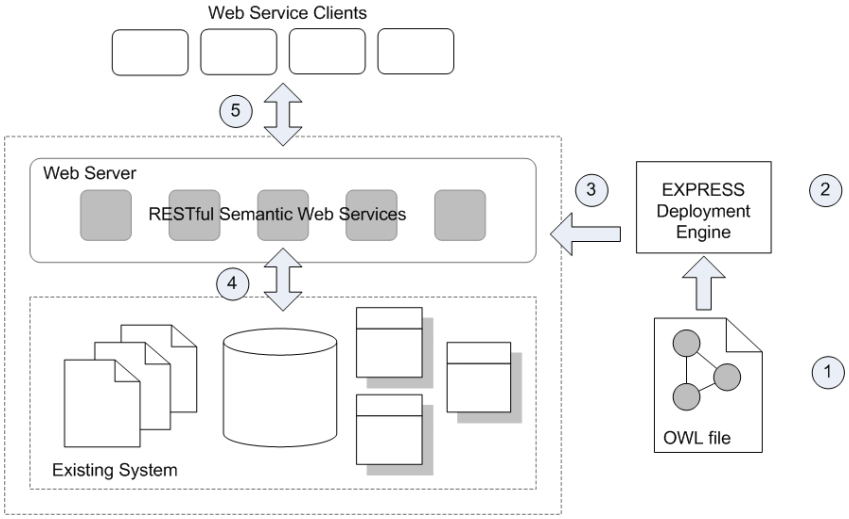


Figure 10 Steps for Deploying Web services in EXPRESS

We can envision the use of EXPRESS in three cases, depending on the type of the existing system:

1. To provide a RESTful interface for Semantic datasets;
2. To make existing Web services RESTful and Semantic;
3. To provide legacy systems with Semantic and RESTful Web services;

Table 13 describes what EXPRESS offers for these systems and the tasks required.

Table 13 Uses of EXPRESS

Existing System	What EXPRESS offers	Tasks Required
Semantic datasets (Linked Data)	▪ Data manipulation through a RESTful interface ▪ Access Control	OWL file exists EXPRESS : Generates Stubs Developer : Specifies Access Control EXPRESS : Generates the code in the stubs because it is direct data manipulation
Web service	▪ Makes the Web service RESTful and Semantic	Developer : Creates OWL file Developer : Specifies Access Control EXPRESS : Generates Stubs Developer : Links the generated stubs to business logic in existing Web services
Legacy System No Web service	▪ A RESTful and Semantic Web service	Developer : Creates OWL file Developer : Specifies Access Control EXPRESS : Generates Stubs Developer : Links the generated stubs to business logic or codes it in the stubs

A prototype EXPRESS deployment engine was developed. The aim was to assess the applicability of EXPRESS and identify potential problems. The engine parses the OWL file then assigns for each class, property or individual a URI or a URI pattern. It then enables the user to specify which URIs can be accessed, by which type of clients, and which methods (GET, PUT, POST, or DELETE) the clients can apply to those URIs. After that the stubs that respond to the HTTP methods for these URIs are created.

Jena[12] was used for parsing the OWL files and generating the URIs and the URI patterns. To generate the stubs Restlet[13] was used. Restlet is a REST framework in Java. Using the Restlet API it enables the creation of stubs called restlets that respond to HTTP methods. These restlets represent resources or classes of resources. It also provides a routing mechanism to forward requests, based on the URI structure, to appropriate restlets. In terms of security, it offers several authentication and authorisation methods. The stubs generated by the EXPRESS deployment engine are restlets. The routing and authorisation code is generated based on the information about the types of clients and the methods they are authorised to perform on the URIs. The type of authorisation needed in EXPRESS is a fine-grained RBAC. For instance, in the Bookstore example, the Customer can only perform a `GET` on this type of URI

```
http://bookstore.com/Order/{OrderID}/hasStatus
```

At the same time a delivery service can perform `GET` and `PUT`. This kind of fine-grained access control is not directly supported by Restlet, so its authorisation mechanisms were extended to implement it. The following figure shows the steps a developer should follow to deploy Web services in EXPRESS.

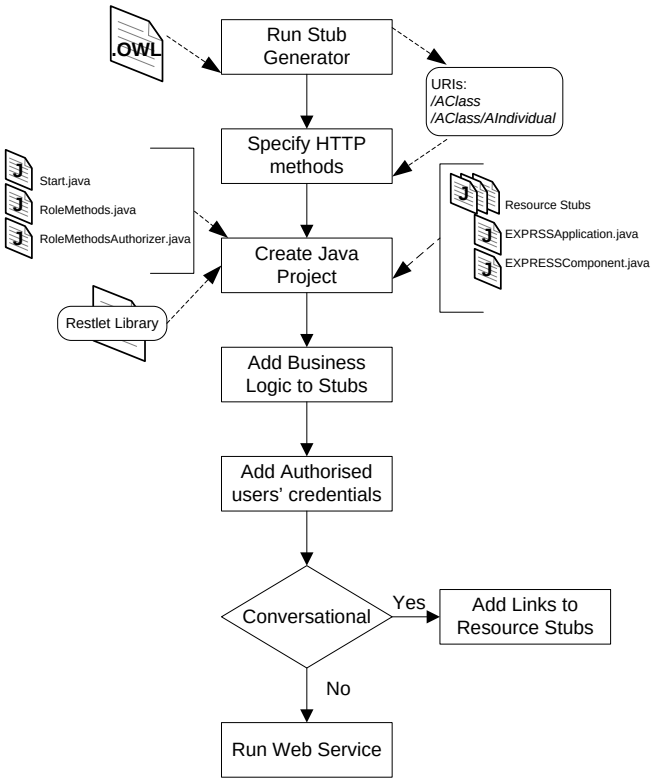


Figure 11 Steps to deploy a Web service using the stub generator

[12] Jena, Semantic Web Framework for Java, http://jena.sourceforge.net/
[13] Restlet, Lightweight REST framework, http://www.restlet.org/

The EXPRESS Prototype is also available online at (http://express.ecs.soton.ac.uk/), a user can upload an ontology, and configure the Web stubs through a webpage; as a result the online engine will generate the stubs. The generated stubs can be downloaded, or deployed temporarily at the server (run in a sandbox). If they are deployed at the server, they can be tested them using either using a browser for `GET` requests, developing a client that performs the calls, or more conveniently test them using tools such as Poster[14], a Firefox plug-in developer tool to facilitate interacting with Web services, by constructing HTTP requests from within the browser.

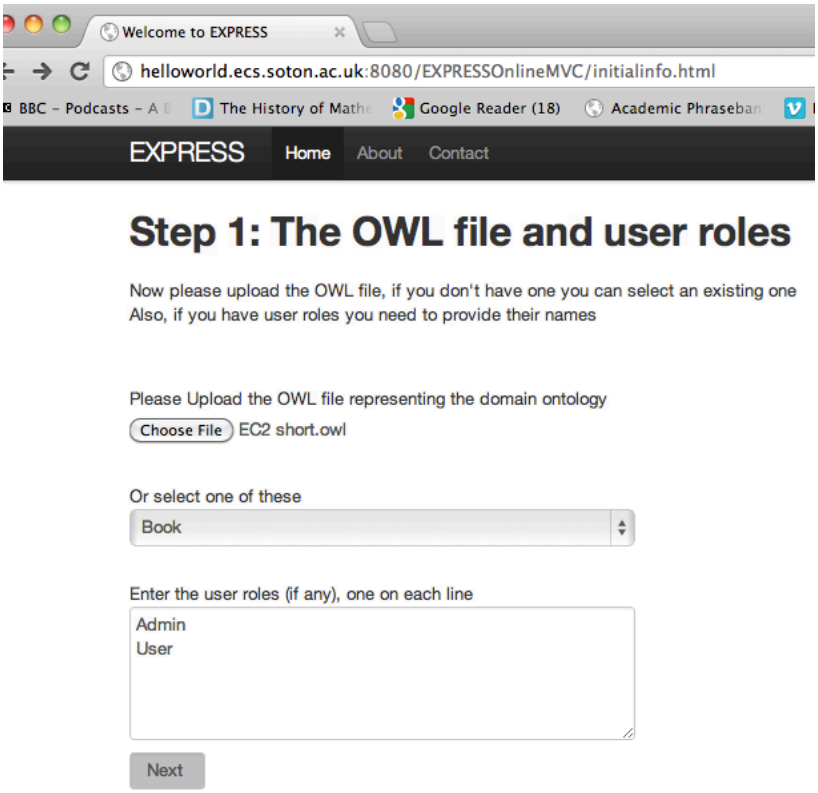


Figure 12 Online EXPRESS, the 1st step providing an OWL file and the roles

Figure 12 shows the webpage where the user can upload the ontology and provide the user roles for the EXPRESSive service; this is the first step. Based on the information provided in Step 1, the second webpage, shown in Figure 13, shows the resource URIs obtained from the uploaded ontology, and enables the user to specify, the interaction requirements, access control and allowed HTTP methods on each one of them.

[14] Poster Firefox Extension https://code.google.com/p/poster-extension/

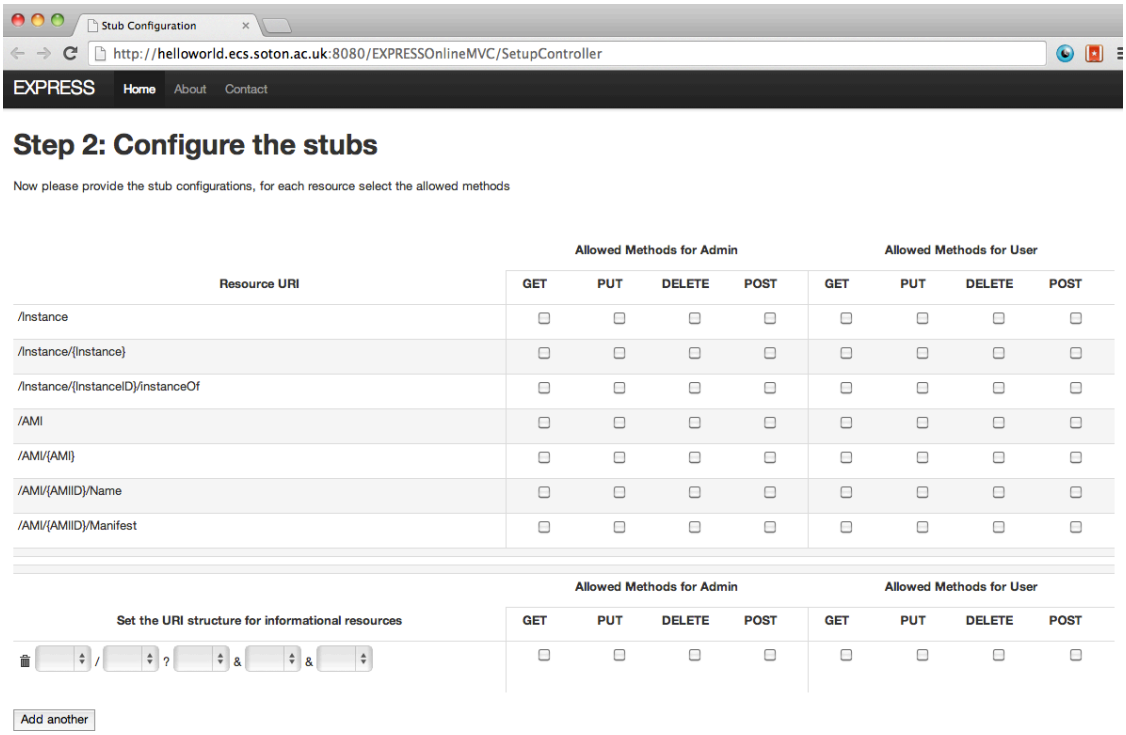


Figure 13 Online EXPRESS, the 2nd step configuring the stubs

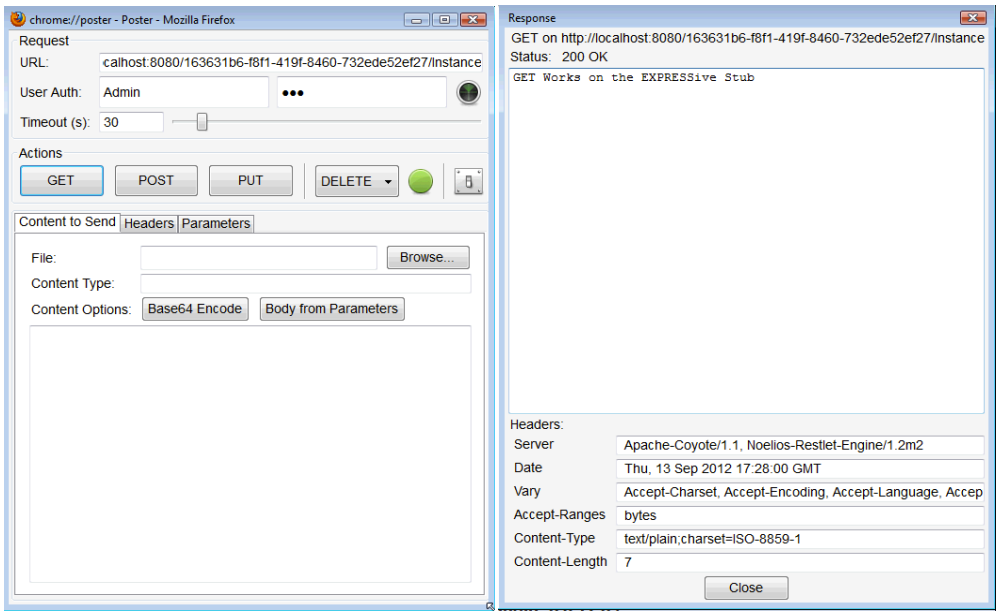


Figure 14 Using Poster to interact with the generated Stubs

After the stubs are generated and deployed at the server, they are given a temporary URI. In the example in Figure 13 it was in the path /163631b6-f8f1-419f-8460-732ede52ef27/ at the server. The stubs deployed there can be accessed via Poster. Figure 14 shows a `GET` request on a protected resource /163631b6-f8f1-419f-8460-732ede52ef27/Instance -in which a username and password were provided- and the server's response.

5.4 EXPRESS and SWS approaches

Section 3.5 compared the 27 SWS approaches that were reviewed in Chapter 3 in terms of the capabilities they offered, and Section 4.3 compared them according to the interaction requirements they fulfilled. This section compares EXPRESS to these SWS approaches. Table 14 is the combination of Table 1 and Table 8 with the addition of EXPRESS in the last row, which shows the capabilities it supports and interaction requirements it fulfils.

Table 14 Comparison of SWS including EXPRESS

Publication	Capabilities						Requirements					
		Composition										
	Discovery	Orchestration	Auto. Composition	Choreography	Conversational	Data Integration	Resource Rep.	Mutability	Plurality	Atomicity	Synchronisation	Roles
OWL-S (Martin *et al.*, 2004)	✓	✓	✓	x	x	x	*	x	*	✓	*	x
WSMO (Bruijn *et al.*, 2005a)	✓	✓	✓	✓	x	x	*	x	*	x	*	x
SAWSDL (Farrell and Lausen, 2007)	✓	x	✓	x	x	x	*	x	*	x	*	x
WSDL-S (Akkiraju *et al.*, 2005)	✓	x	✓	x	x	x	*	x	*	x	*	x
SWSF (Battle *et al.*, 2005)	✓	✓	✓	✓	x	x	*	x	*	✓	*	x
DSD (Klein *et al.*, 2005)	✓	x	✓	x	x	x	*	x	*	x	*	x
SA-REST (Lathem *et al.*, 2007)	✓	x	✓	x	x	x	*	✓	*	x	x	x
hRESTS (Kopecky *et al.*, 2008)	*	x	*	x	x	x	*	✓	*	x	x	x
MicroWSMO (Kopecky *et al.*, 2008)	*	X	*	x	x	x	*	✓	*	x	x	x
WSMO-Lite (Vitvar *et al.*, 2007)	✓	X	✓	x	x	x	*	x	*	x	*	x
RESTfulGrounding (Filho and Ferreira, 2009)	✓	✓	✓	x	x	x	*	✓	*	✓	x	x
ReLL (Alarcon and Wilde, 2010)	✓	✓	x	x	✓	✓	*	✓	✓	✓	x	x
SBWS (Battle and Benson, 2008)	✓	x	✓	x	x	✓	*	x	*	x	*	x
SPARQL descriptions (Sbodio *et al.*, 2010)	✓	x	✓	x	x	x	*	x	*	x	*	x
LIDS (Speiser and Harth, 2011)	✓	x	x	x	x	✓	*	x	x	x	x	x
LOS (Krummenacher *et al.*, 2010)	✓	✓	x	x	x	✓	*	✓	x	x	x	x
Semantic REST (Battle and Benson, 2008)	x	x	x	x	x	✓	✓	✓	✓	x	x	x
Zhao and Doshi (2009)	x	x	✓	x	x	x	✓	✓	✓	x	x	x
Hernandez and Garcia (2010)	x	✓	x	✓	x	x	✓	✓	x	x	✓	x
TSC (Riemer *et al.*, 2006)	✓	*	*	*	*	✓	✓	✓	*	*	✓	✓
RESTdesc (Verborgh *et al.*, 2011)	✓	x	✓	x	✓	x	*	✓	x	✓	x	x
iServe (Pedrinaci *et al.*, 2010b)	✓	x	✓	x	x	x	*	✓	x	x	*	x
SADI (Wilkinson *et al.*, 2009)	✓	x	✓	x	x	✓	*	✓	x	x	✓	x
HyperData (Kopecky *et al.*, 2011)	x	x	x	x	✓	✓	✓	✓	✓	✓	x	x
Hypermedia RDF (Kjernsmo, 2012)	x	x	x	x	✓	✓	✓	✓	x	✓	x	x
RDF-REST (Champin, 2013)	x	x	x	x	✓	✓	✓	✓	✓	✓	✓	x
SSWAP (Gessler *et al.*, 2009)	✓	x	✓	x	x	x	✓	✓	x	x	x	x
EXPRESS	✓	✓	✓	x	x	x	✓	✓	✓	✓	✓	✓

✓: addressed by the approach **x**: not addressed by the approach *****: assumed existing & addressed by other layers

As the table shows EXPRESS fulfils the six interaction requirements, which was shown in Section 5.2, the closest approach to EXPRESS in terms of interaction requirements is RDF-REST, which fulfils them all except roles. As for capabilities, EXPRESS addresses discovery, which is demonstrated in the next chapter. It also supports data integration, because it consumes and produces RDF, which makes it suitable for providing interfaces for datasets. Conversational services are a goal for EXPRESS, which it supports by using Link Headers, the practicality of this solution is left for future work, and discussed in Section 8.4.

5.5 Conclusions

This chapter presented EXPRESS, a RESTful Semantic Web service approach which aims to eliminate explicit service descriptions for describing services. EXPRESS works by providing a straightforward mapping between resources (described in an ontology) and URIs that respond to HTTP requests. This chapter also shows how such mapping can facilitate stub generation in the aim to reduce implementation effort.

The design of EXPRESS is based on the argument that the Web's infrastructure has more to offer than mere data retrieval, and achieving extended functionality does not mean that extra layers of definitions are required, or a new infrastructure. Instead, EXPRESS suggests that what is needed is a different conceptualisation of the problem, and although this conceptualisation may in itself impose something of an overhead, this is outweighed by the simpler relationship between ontology, service and protocol that we have achieved with EXPRESS. This method allows ontologies to be transformed into SWS without the need for additional meta-models or vocabularies.

The next two chapters present evaluations of EXPRESS: in Chapter 6 the discoverability of EXPRESS's semantic description is evaluated, and in Chapter 7 it is evaluated in terms of development effort and practicality.

Chapter 6: Semantic Matchmaking in EXPRESS

Chapter 5 presented EXPRESS and demonstrated how it provides and semantically describes Web services. This chapter assesses the discoverability of the semantic descriptions, using a standardised test-collection and evaluation environment. This chapter will discuss service matchmaking in EXPRESS, the methodology for evaluation and the results. It addresses the third research question: Can EXPRESS provide a similar level of semantic expressivity to existing approaches?

Section 6.1 provides an overview of semantic service matchmaking, Section 6.2 discusses semantic service matchmaking in EXPRESS. The experimental design is explained in Section 6.3, Section 6.4 presents and discusses the results and 6.5 concludes this chapter.

6.1 Semantic Service Matchmaking

This section provides a brief overview of semantic service matchmaking. According to Klusch (2008a), semantic service discovery is: "the process of locating existing Web services based on the description of their functional and non-functional semantics."

Dong *et al.* (2012) identify six dimensions for analysing SWS matchmakers. These are

1. The languages used for describing the semantics of Web services.
   These differ among the SWS approaches, for example OWL and RDF are used in OWL-S, WSML is used in WSMO and N3 in RESTdesc.
2. The SWS matching parts or parameters.

Different parts/parameters of service description are used for matchmaking. These can be: the service profile, i.e. inputs, outputs and/or preconditions and effects (IOPE), the service process, and non-functional properties.

3. Matching approaches and matching degrees.
   The matching approaches can be logic-based, non-logic-based (e.g. text similarity or graph matching) or a hybrid of both. The mechanism is considered adaptive if it involves learning (Klusch, 2008a).
   As for matching degrees or degrees of logical relevance, these are usually specified for logic-based matching. These differ slightly from one approach to another, but in general are: exact, plug-in, subsume, intersection and fail. (Paolucci *et al.*, 2002; Dong *et al.*, 2012; Klusch, 2008a).
4. The testing platforms and collections.
   The two main evaluation platforms for SWS are SWS Challenge and Semantic Service Selection (S3) contest: their goals are mentioned in Chapter 3. The approach used in S3 is adopted in the evaluation of EXPRESS and is further discussed in this chapter.
5. The SWS discovery mechanisms.
   This concerns where and how information such as service descriptions, ontologies and registries are stored, published and discovered.
6. The SWS discovery architecture.
   The architecture can be centralised or decentralised, as in P2P.

The SWS discovery mechanism and architecture (the fifth and sixth dimension) are of less concern in the scope of this thesis, as the matchmaking process is bound to happen, regardless of where the service descriptions are assumed to reside, as even in the case where there is no dedicated architecture for discovery, the service consumer (or client) would be performing some form of matchmaking, locally.

6.2 Matchmaking in EXPRESS

Starting with the first dimension mentioned above, the language used in EXPRESS is OWL, as explained in Chapter 5. The second and third dimension, the matching parts and the potential matching approaches, are discussed below.

A service in EXPRESS is mainly described by two elements:

1. The URI of the endpoint, that maps to a resource or several resources in the domain ontology provides three main aspects:

a. As discussed in Chapter 5, the URI templates correspond to graph patterns, hence, graph matching methods can be applied, such as the approach by Stadtmüller and Norton (2013).
b. In cases where a URI refers to a class, the monolithic DL matching techniques can be applied. In monolithic DL services, the whole service is defined as a concept. Examples of such definitions, from (Grimm, 2007) are:

$$S = Flight \sqcap \forall\, from.USCity$$
$$R = Flight \sqcap \forall\, from.UKCity$$

S and R represent service and request definitions, respectively. Few matchmakers assume this way of defining services, only four out of the 27 classified by Klusch (2008a).
c. From the filtering resources' URIs, inputs and outputs can be extracted. Hence profile-matching techniques can be applied, which is the method adopted in this evaluation.

2. The HTTP method allowed on the endpoint.
   An effect or postcondition in EXPRESS is a direct function of the HTTP method and the resource represented by the endpoint, and hence, there is no need to explicitly state the postconditions. This is one of the ways EXPRESS reduces the complexity of service descriptions. However, as a consequence EXPRESS is less flexible than OWL-S in defining postconditions, because, in OWL-S, postconditions are logical expressions and the number and type of variables are not restricted.
   The advantage EXPRESS has over OWL-S is its utilisation of the HTTP methods' semantics in the semantic service description. In OWL-S, the semantic service description builds on the basic description of inputs and outputs only, and as result, there is a need for other means to describe what the service does with those inputs and outputs: that is why explicit preconditions and effects needed to be introduced, to describe the state of the world required before and resulting after the service is executed.

The approach that EXPRESS takes is that a service request will be formulated in a similar fashion to the service offer. Hence, the matchmaking between service request and the service offer is a matching based on the two elements mentioned above.

As for the fourth dimension, the testing platforms and collections are discussed in the following section, the Experimental Design.

6.3 Experimental Design

This experiment is designed to test whether the semantic elements exposed by EXPRESSive service descriptions are sufficient to be consumed/utilised by well-performing matchmaker algorithms, and hence enable similar matching quality to other semantic service approaches, while minimising the required semantic descriptions. The approach we take is similar to (Sbodio *et al.*, 2010).

Three main components were required to perform this experiment:

1. A well-performing matchmaker, adapted to be used with EXPRESSive descriptions.
2. A test collection of EXPRESSive services and an equivalent test collection in another SWS approach (OWL-S is chosen for this experiment) to compare the effect of the service descriptions on the performance of the matchmaker.
3. An evaluation environment (which serves as a benchmarking platform), used to run the matchmaker on both test collections, and calculate results. The Semantic Web service Matchmaking Evaluation Environment SME^2 is used in this experiment. It is designed so that matchmakers and test-collections can be plugged in, and provides a platform for evaluating the matchmakers' performances.

These components are discussed in further detail in the following subsections.

6.3.1 Adapting the iSeM Matchmaker

The iSeM (Klusch and Kapahnke, 2010a) matchmaker was chosen, because it fulfils the following requirements:

1. Has a good performance on OWL-S and SAWSDL descriptions.
2. Implements an interface for SME^2.
3. Access to source code, and hence can be adapted to EXPRESSive service descriptions.

It was developed by experts in the field, and it has a better performance than other matchmakers, according to the S3 2010 and 2012 competitions[15], and since the aim of the experiment is to compare the expressiveness of the semantic descriptions, having a fixed matchmaker algorithm is more objective.

[15] Annual International Contest S3 on Semantic Service Selection
2010 http://www-ags.dfki.uni-sb.de/~klusch/s3/html/2010.html
2012 http://www-ags.dfki.uni-sb.de/~klusch/s3/html/2012.html

The iSeM matchmaker is a hybrid and adaptive matchmaker for both OWL-S and SAWSDL descriptions. It matches service functional descriptions and has the following features (Klusch and Kapahnke, 2010a):

1. Signature Matching (IO).

iSeM deploys several matching methods for the services' inputs and outputs. These matching methods are: strict-logical, approximated-logical, structural and textual. Approximate-logical, structural and textual matching methods aim to compensate for the strict-logical-matching false negatives.

The strict-logical matching performs subsumption checks on input and output classes: this causes some matches to fail. Approximate logical matching assumes that the parts of class definitions causing the match failure are unnecessary, and matches concepts accordingly. This approximation also enables the ranking of services according to the resulting information gain and loss in the redefined concepts. The structural and textual matching methods are non-logical ones. The structural match is calculated according to the typology of the ontology containing the defining concepts. The textual match, on the other hand, is calculated according to the weighted keyword vectors containing the concepts' unfolding (i.e. their primitive concept definitions).

2. Specification Postconditions and Effects (PE) matching.

This matches postconditions and effects written in PDDL. It checks if a service plugs in a request, i.e. that the preconditions of the request entail the preconditions of the service and the effects of the service entail the effects of the request.

3. SVM (support vector machine)-based semantic relevance learning.

The SVM learns the weighted aggregation of the matching methods mentioned above. It uses 5% of the test collection as a training set.

Both the source code and the binary version of the iSeM v1.1[16] are implemented to work on OWL-S. iSeM v1.1 contains several variants (matchmaking methods) implemented as modularised filters. The variants and their types are presented in the following table.

Table 15 iSeM matchmaker variants

The iSeM matchmaker variant	IO	PE	SVM
Logic-based	✓		
Approximate logic-based	✓		
Structure	✓		

[16] Adaptive, hybrid semantic service profile (IOPE) matchmaker iSeM V1.1 (OWL-S) http://www.semwebcentral.org/projects/isem/

Text similarity	✓		
SVM logic-based, text-similarity, structure	✓		✓
SVM logic-based, text-similarity, structure, specification	✓	✓	✓
SVM logic-based, text-similarity, structure, approx. logic-based, specification	✓	✓	✓

The iSeM source code was used to develop an EXPRESSive version by modifying the service manipulation package to extract the service signature concepts from a service written in EXPRESS instead of OWL-S. To distinguish between the EXPRESSive version of iSeM and the original one throughout this chapter, they will be referred to as iSeM EXPRESS and iSeM OWL-S respectively.

6.3.2 Creating the EXPRESSive Test Collection (EXPRESS-TC)

OWLS-TC (Klusch and Kapahnke, 2010b) is a test collection for semantic matchmaking evaluations. It has been used in the S3 contests and is widely accepted by the SWS community. Version 4.0 contains 1083 services, 42 queries (service requests), and 48 ontologies. The relevance of services with respect to queries is also provided as binary and graded judgements. These judgements are not complete, as only 10% of the request-service combination has been judged, using a pooling strategy adopted by Text Retrieval Conference (TREC). The judgments are derived from the top 100 results from matchmakers in the 2008 S3 contest (Klusch *et al.*, 2010b).

The services are grounded in WSDL 1.1, and the test collection includes the WSDL files as well. 160 services and 18 requests out of the total have been modified to include preconditions and effects expressed in the Planning Domain Definition Language (PDDL) 2.1.

The OWLS-TC was chosen for the experiment because in addition to it being widely accepted by the community, the source code available for the iSeM matchmaker is developed for OWL-S services.

There are two methods to create an EXPRESSive test collection, and a decision had to be made between:

1. **Manual Conversion**
   Selecting a subset of the OWL-S test collection services to be converted manually into EXPRESSive descriptions and performing the experiment on a subset of the test collection.
2. **Automatic Conversion**

Finding an approach to automatically convert the whole OWL-S test collection to an EXPRESSive test collection, both the 42 requests and the 1083 services.

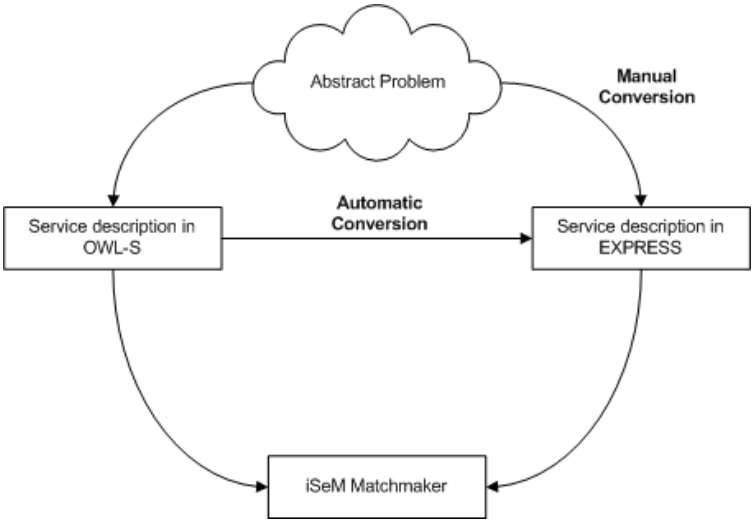


Figure 15 The manual and automatic approaches to generate the test collection

Figure 15 illustrates the two approaches. They both have advantages and risks, which are discussed next.

Advantages and Risks of the Manual Conversion

The manual conversion is achieved by reading the OWL-S description, reverting to the actual problem it aims to solve and then using that abstract problem to create a description in EXPRESS.

This ensures that the EXPRESSive description is not influenced by another approach's conceptualization of the problem, in this case OWL-S, hence the semantic elements exposed by EXPRESS truly reflect what would be reached if there was no OWL-S description, However there are two risks, with this approach:

1. The size of the test collection will be considerably smaller, and as a result the reliability of the experiment will be weaker.
2. There is more chance of bias when converting the queries and services. The bias could occur by making the services closer to matching the queries; however this could be overcome by asking impartial/neutral participants to perform the semantic description of both queries and services.

Advantages and Risks of the Automatic Conversion

The advantages of the automatic conversion approach over the manual one is that it results in a considerably larger test collection, which increases the reliability of the results. The automatic conversion however also introduces a risk that could weaken the argument for the experiment.

In the automatic approach, the risk is that the automatic conversion may render an EXPRESSive description that would not occur as a natural process of conceptualising the problem in EXPRESS, and will only be an OWL-S description coerced into an EXPRESSive representation. Hence the semantic elements exposed by the EXPRESSive version would be the same as the ones exposed by the OWL-S ones, and this may not have occurred if we started with the abstract problem, and took a manual approach to the conversion instead of an automatic one. The reason this is a risk, is that the matchmaking capabilities of EXPRESS, would not be a result of following the approach itself, instead they would exist because of the conversion from OWL-S.

However, there are multiple ways to design either an OWL-S or EXPRESSive representation of the same service. A reasonable assumption to make, is that at least one OWL-S representation and one EXPRESSive one would expose the same inputs and outputs.

The discussion above has raised issues with both the automatic and the manual conversion from OWL-S service descriptions to EXPRESSive ones. The automatic conversion was preferred, because it would render a considerably larger test collection, and was achieved by the following methods:

1. For each OWL-S description, whether a request or a service, extracting the semantic elements, in this case the inputs and outputs.
2. Providing an EXPRESSive semantic description template, where those elements could be plugged in.

To minimise the risk of the automatic conversion (i.e. the EXPRESSive descriptions not occurring naturally), a subset of the services were converted manually to inform the design of the automatic conversion method. The 42 queries from the OWLS-TC were chosen to be converted manually, as they can be considered a representative subset of the test collection they will be matched against.

The 42 queries have inputs and outputs, 37 of the queries are read-only (informational) services, and 5 of them are updating queries. The read-only services in EXPRESS, are modelled by applying a `GET` method to a resource and, as discussed in Chapter 5, these resources represent either a class, an individual, a property of a named individual, or a filter on a collection. With class, individual and property, resource types, the client does not provide any inputs. So to represent the read-only queries in the test collection, we needed to represent them as filters on a collection.

The method was to take each one of the queries and to conceptualise the problems they represent as EXPRESSive services, then analyse how they relate to the OWL-S service. This resulted in the realisation of several approximations required for automatic conversion:

1. An issue that causes a mismatch between EXPRESSive and OWL-S descriptions is a design decision of EXPRESS, discussed in Chapter 5, which restricts the representation of multiple outputs. This means if an OWL-S service has multiple outputs, EXPRESS will represent them as one output, which is the union of those outputs.
2. In some cases, such as when a service returns a price of merchandise, the intuitive conceptualisation is to have the price as a property of the merchandise (for example the query named "2For 1 DVD/MP3 player price service"). However, it is also possible to reverse the relationship and to have the merchandise as properties of the price.
3. The preconditions and effects (PE) in the OWL-S service are ignored because, in EXPRESS, preconditions are not specified, and as for effects, the semantics are described by the method and the type of resource. However, as discussed by Klusch and Kapahnke (2010a), the effect on the results is minor because only 17% of the services in the OWL-S test collection have PE.
4. As discussed in Chapter 5, the HTTP method is a part of the service definition in EXPRESS; in Section 6.2, the method can also be used for matchmaking. However in the OWL-S test collection, 37 out of 42 of the queries were read-only services and the others were updating services. As for the services, only 47 out of the 1083 services are judged to be relevant to these queries. Since these form only a very small percentage of services, we assumed that all the services, after transforming them into EXPRESS, are to be retrieved with a `GET`.

In addition to undertaking this manual process to guide the automatic conversion, in Chapter 7 (Expert Reviews), experts are asked to compare two versions of an EXPRESSive service, a manually created one, and another which is automatically converted from an OWL-S service. Results are discussed in Section 7.3.1.3.

Taking into consideration the approximations above, the following steps were taken to transform an OWL-S service into an EXPRESSive one:

1. Create an ontology containing the inputs and outputs of the OWL-S service.
2. If more than one output exists, a new class is created which is the union of all the outputs.
3. Create Properties, where the domains of the properties is the OWL-S output, and their ranges are the OWL-S inputs.

4. **Create the URI of the endpoint in the following form**
   `Output?hasInput1={}&hasInput2={}&…`

The following is the conversion of the "2 For 1 DVD/MP3 player price service", which is described by "This service returns prices of a given pair MP3 Player brand and DVD Player brand". It has the following inputs: MP3PLAYER and DVDPLAYER, and this output: PRICE. The full OWL-S service is in Appendix D.

The EXPRESSive version of this service is below

The endpoint is

```
Price?hasMP3player={}&hasDVDplayer{}
```

and the following ontology represents the EXPRESSive interface

```
:MP3player      a owl:Class;
          owl:equivalentClass
          <http://127.0.0.1/ontology/my_ontology.owl#MP3player> .
:DVDplayer      a owl:Class;
          owl:equivalentClass
          <http://127.0.0.1/ontology/my_ontology.owl#DVDplayer> .


:Price      a owl:Class;
          owl:equivalentClass
          <http://127.0.0.1/ontology/concept.owl#Price> .
:hasDVDplayer      a owl:ObjectProperty;
          rdfs:domain :Price;
          rdfs:range :DVDplayer .
:hasMP3player      a owl:ObjectProperty;
          rdfs:domain :Price;
          rdfs:range :MP3player .
```

The restrictions EXPRESS imposes on ontology design are shown in the example above: for example, the domain and range have had to be stated for each property.

6.3.3 Evaluation Environment

The matchmaking experiment is conducted using the Semantic Web service Matchmaking Evaluation Environment[17] SME². This environment is used in the annual Semantic Service Selection (S3) contest. SME² provides an extensible framework for testing different matchmaking approaches (algorithms). It enables

[17] The Semantic Web Service Matchmaker Evaluation Environment (SME2)
http://projects.semwebcentral.org/projects/sme2/

developers of matchmaking approaches to plug-in their matchmakers and run them against the provided test collections of services. A service test collection is made up of service requests (called queries), service offers, referenced ontologies, and the result set, i.e. the correct answers. Two test collections are shipped with SME², the OWLS-TC mentioned above, and SAWSDL-TC (Klusch and Kapahnke, 2010c), a SAWSDL version of almost all of the services in the OWLS-TC. In addition, new test collections can be plugged in.

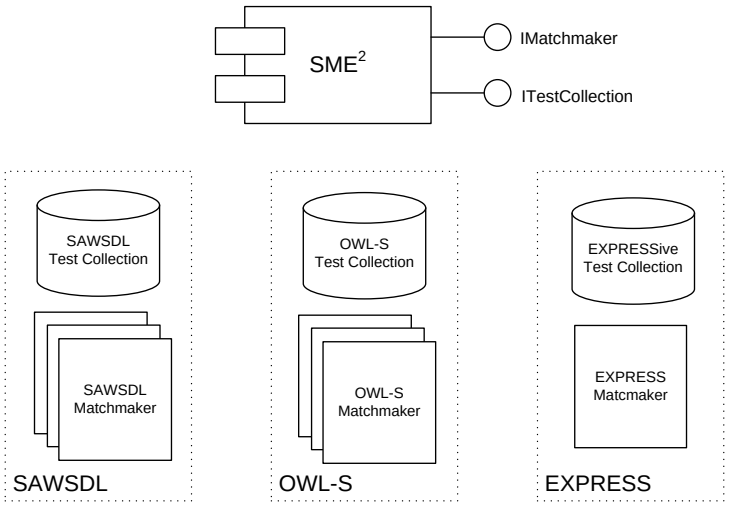


Figure 16 Architecture of SME²

SME² calculates several information retrieval (IR) measures, for binary relevance, graded relevance and time consumption. The main measures presented in S3 are:

1. For binary relevance:
    a. Macro-averaging for Precision/Recall measures.
    Precision and recall in IR are defined as follows, where A is a set of relevant documents in the dataset, and B is the set of retrieved results.

$$Precision = |A \cap B|/|B|$$

$$Recall = |A \cap B|/|A|$$

A method for averaging these values is called macro-averaging, and it is calculated for precision as follows (Klusch *et al.*, 2010a):

$$Precision_{macro}(i) = \frac{1}{|Q|} \cdot \sum_{q \in Q} \max\{P_O | R_O \geq Recall_i \wedge (R_O, P_O) \in O_q\}$$

where $0 \leq i \leq \lambda$, in SME² $\lambda = 20$,

O_q is the set of observed precision/recall values for true positives and

Q is the set queries

For each query, the maximum precision at an i level of recall is taken (i.e. after a certain percentage of documents have been retrieved), summed then averaged over the total number of queries. This means each query will have an equal weight. An alternative method for averaging is called micro-averaging, where each document (service) has an equal weight; however, since it is not presented in the results, it is not discussed here. The results of macro-averaging are usually presented as a graph such as in Figure 17.

b. Average Precision (AP)

AP involves precision, recall, and ranking in the measure of performance.

$$AP = \frac{1}{R}\sum_{n=1}^{N} rel(d_n)\,\frac{\sum_{i=1}^{N} rel(d_i)}{n}$$

where $rel(d_n) = 1$ if the document at rank n is relevant and 0 if it is not. R is the total number of relevant documents.

AP is the average of the precision value after each relevant document is retrieved. After AP is calculated for each query the mean for all queries is calculated, to obtain a single score for the matchmaker.

2. For graded relevance:

In these measures the degree of relevance (ranking) of services is taken into consideration. Unlike binary relevance, where a service is either relevant or not, graded relevance assumes varied degrees of relevance. In the test collections used with SME^2, there are four degrees of relevance: highly relevant, relevant, partially relevant, and not relevant. The graded relevance measured used are:

a. normalised Discounted Cumulative Gain (nDCG):

This is based on discounting the gains (value) according to the ranking of documents. The cumulative gain (cg) is the sum of relevance weights of retrieved documents. The discounted cg (DCG) takes the rank into consideration and reduces the weight of lower ranked documents, usually be dividing them by $\log_2(rank)$. There is usually a cut-off rank ***p***, where DCG_p is calculated. The normalised DCG, is obtained in order to average the DCG values at a specific rank across a set of queries with different numbers of relevant documents. nDCG is the result of dividing the DCG value by the ideal DCG value; the nDCG values can then be averaged for all queries.

b. Q-Measure:

Q-measure is a generalisation of AP to accommodate graded relevance and it is a modification of weighted average precision.

$$Q = \frac{1}{R}\sum_{n=1}^{N} rel(d_n)\,\frac{cbg}{cg_I + n}$$

where cg_I is the ideal cumulative gain at rank n
and cbg is the cumulative-bonused gain

cbg is similar to cg: instead of just summing the weights of relevant documents, it adds an extra reward for each relevant document.

For AP, nDCG and Q-Measure, SME² calculates two scores, for both complete and incomplete judgements. The measures for incomplete judgments are the ones reported in the S3 contest, and are named AP', nDCG' and Q'. These are calculated using only the results that are rated in the judgement sets (as mentioned in Section 6.3.2, these are incomplete). Zhou and Yao (2010) provide a detailed explanation of these measures and a discussion of their effectiveness.

3. Time consumption: Average Query Response Time (AQRT)
AQRT is the average time a matchmaker takes to return results for a query, and it is calculated in seconds.

6.4 Results and Analysis

The seven variants of iSeM EXPRESS, and iSeM OWL-S were run on their corresponding test collections. Table 16 shows the results from the runs.

Table 16 Results of running iSeM OWL-S and iSeM EXPRESS on SME²

	Matchmaker Variant (Filter)	AQRT (s)	AP'	Q'	nDCG'
iSeM OWL-S	Logic-based	0.190	0.699	**0.726**	**0.807**
	approx. Logic-based	0.702	**0.696**	**0.701**	**0.748**
	Structure	0.303	**0.747**	**0.734**	0.783
	Text-similarity	1.517	0.800	0.804	0.891
	SVM logic-based, text-similarity, structure	3.056	**0.821**	**0.751**	**0.790**
	SVM logic-based, text-similarity, structure, specification	3.211	**0.840**	**0.782**	**0.829**
	SVM logic-based, text-similarity, structure, approx. logic-based, specification	3.942	**0.839**	**0.783**	**0.820**
iSeM	Logic-based	**0.153**	**0.700**	0.724	0.815
	approx. Logic-based	**0.592**	0.681	0.690	0.739
	Structure	**0.290**	0.717	0.712	**0.755**
	Text-similarity	**0.942**	**0.811**	**0.812**	**0.895**
	SVM logic-based, text-similarity, structure	**1.398**	*0.411*	*0.463*	*0.519*

	SVM logic-based, text-similarity, structure, specification	**1.507**	*0.309*	*0.365*	*0.387*
	SVM logic-based, text-similarity, structure, approx. logic-based, specification	**2.211**	*0.309*	*0.381*	*0.400*

The table shows the AQRT, AP, Q and nDCG for iSeM EXPRESS and iSeM OWL-S. The values in bold indicate better performance. From the AQRT results it is clear that the iSeM EXPRESS filters are faster than the iSeM OWL-S ones.

The values of AP, Q and nDCG for the first four variants (the non-SVM ones) are very close for iSeM OWL-S and iSeM EXPRESS. For text similarity and logic based iSeM, EXPRESS performs slightly better in terms of AP', and slightly worse in approximated logic-based and structure. The highest performing variant for iSeM EXPRESS out of the seven variants, in terms of precision-recall, is text-similarity. For iSeM OWL-S text similarity is the highest of the non-SVM ones.

On the other hand, for the SVM variants (the last three variants), iSeM EXPRESS performs much worse. This is due to the SVM variants being trained on an OWL-S sample of services rather than on EXPRESS sample, and hence tuned towards OWL-S services. Figure 17 and Figure 18 show the macro-averaged precision-recall curves. Figure 17 shows the non-SVM variant's performance, and shows the very close similarity between the iSeM OWL-S variants and the iSeM EXPRESS ones. Figure 18 shows how iSeM EXPRESS variants perform considerably worse than the iSeM OWL-S ones, due the SVM learning effect, as discussed above.

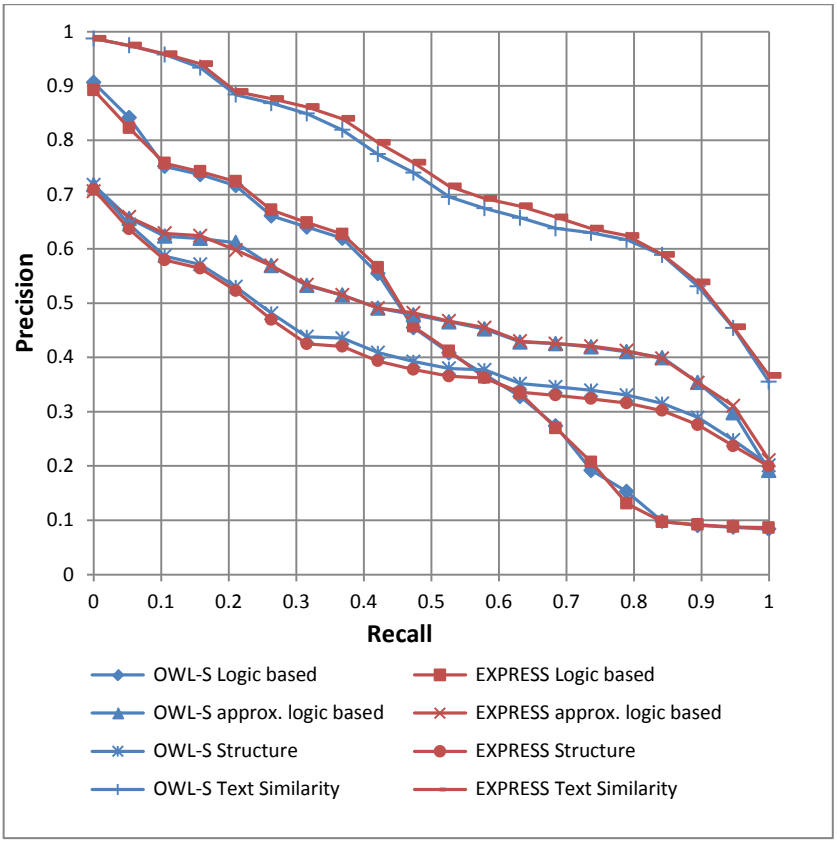


Figure 17 Macro-averaged Precision-Recall Curve for non-SVM variants

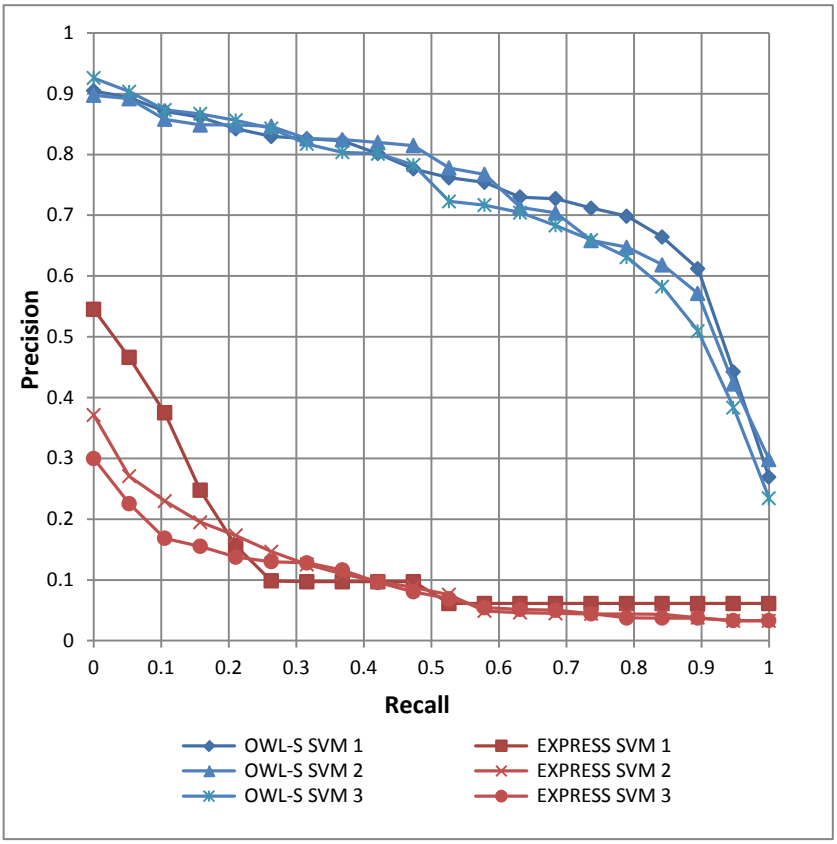


Figure 18 Macro-averaged Precision-Recall Curve for SVM variants

Figure 19 shows the AQRT differences between the approximated logic-based iSeM EXPRESS and iSeM OWL-S. The approximated logic-based variant is used as a representative of the other variants because, as shown in Figure 19 all the iSeM EXPRESS variants outperform the OWL-S ones in terms of speed.

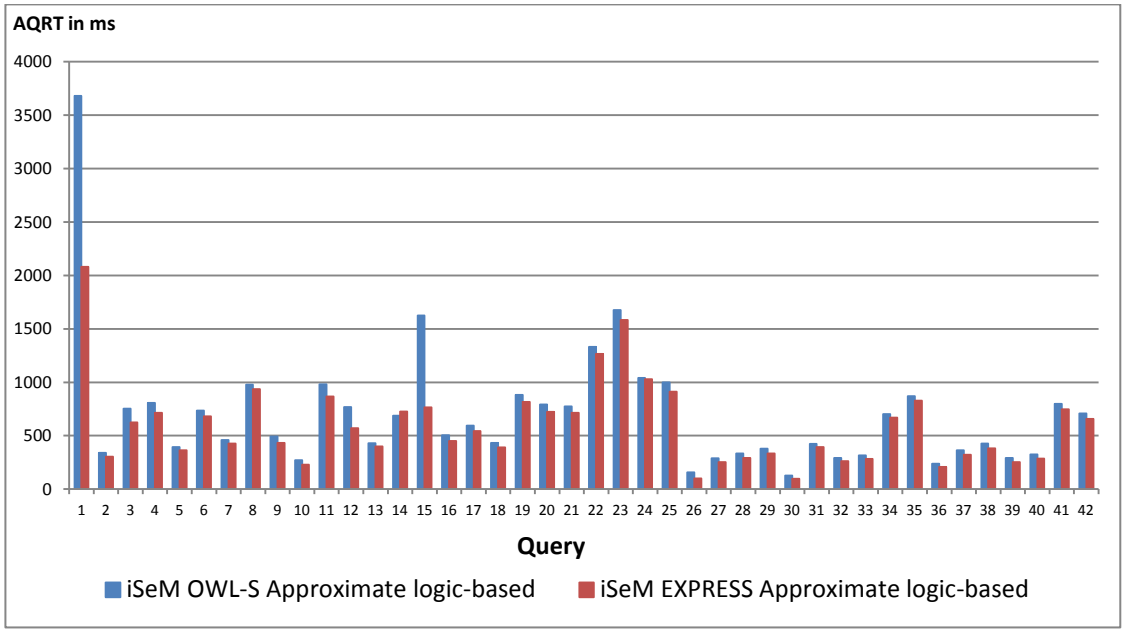


Figure 19 AQRT for iSeM OWL-S and iSeM EXPRESS (Approximate Logic-based)

The statistical significance of the results was measured for two variants, the approximated logic-based, and the text similarity, by conducting the Friedman test for the AP and AQRT for the variants, as shown in Table 17

Table 17 Friedman test for approximated logic-based and text similarity variants

	Approximated logic-based		Text similarity	
	AQRT	AP'	AQRT	AP'
iSeM OWL-S	0.701	0.696	1.517	0.800
iSeM EXPRESS	0.592	0.681	0.942	0.811
P =	0.000	0.028	0.000	0.317

The values of p in Table 17 show that the AQRT improvements in the EXPRESS variants are statistically significant for $p<0.05$. However, this differs for the AP', the approximate logic-based variant, where iSeM EXPRESS performs slightly worse, with a statistical significance $p=0.028<0.05$, meaning that this performance is consistently worse, albeit the difference is small. In the text similarity variance, although the performance of iSeM EXPRESS seems to be slightly better, it is not statistically significant $p=0.317>0.05$.

The objective of this experiment was to show whether EXPRESSive descriptions are as discoverable as other SWS descriptions such as OWL-S. This experiment clearly shows very close performances in terms of precision and recall in the non-

SVM variants and a slightly better performance in speed, ranging from 4% to 38%, as shown in Table 18.

Table 18 % of Improvements of iSeM EXPRESS over OWL-S in terms of AQRT

	iSeM OWL-S	iSeM EXPRESS	%
Logic-based	0.19	0.153	19%
Approximated logic-based	0.702	0.593	15%
Structure	0.303	0.29	4%
Text similarity	1.517	0.942	38%

The table does not list the SVM variants because, although EXPRESS performance is better in terms of AQRT (around 50%), the SVM precision and recall values are much worse, as discussed before, and speed alone is not a gain if those values are not comparable. However, as mentioned before, this is due to the SVM training effect.

Moreover EXPRESS considerably reduces the descriptions sizes, Table 19 shows the means and medians for service descriptions (lines of code (LOC) and size in bytes) in the test collections, it shows that EXPRESSive descriptions are %78-%79 smaller on average.

Table 19 Service description size in LOC and bytes

File size in	Description approach	Mean	Median
LOC	OWL-S	117.89	116
	EXPRESS	25.23	24
	%	**%79**	**%79**
Bytes	OWL-S	6653.71	6422
	EXPRESS	1467.28	1354
	%	**%78**	**%79**

6.5 Conclusions

This chapter assessed the discoverability of EXPRESSive descriptions. It provided an overview of SWS matchmaking and explained how to achieve it in EXPRESS, then discussed the matchmaking experimental design and the results.

The results of the experiment show that EXPRESS descriptions offer very close semantic expressivity to the OWL-S ones. This is indicated by the adapted iSeM matchmaker performance, which yielded very close precision-recall measures with an improvement in speed ranging from 4% to 38%, depending on the matchmaker variant. However, the SVM variants did not work as well with EXPRESS, as they have been trained with OWL-S. This is a promising result considering that EXPRESS massively reduces the size of the service descriptions. However, it also raises an important question: Having demonstrated EXPRESS's competence for semantic matchmaking, what are the trade-offs, i.e. how does this affect the ease

of development, practicality, and semantic richness? This is further investigated in the next chapter.

Chapter 7: Expert Reviews

As a Semantic Web service approach EXPRESS aims to provide semantic descriptions of services while minimising their development effort. Chapter 5 and 6 discussed the functional aspects of EXPRESS, in terms of description, development and matchmaking; this chapter aims to discuss and provide evidence on how EXPRESS reduces the development effort, compared to other Semantic Web service approaches.

Development effort is a non-functional, subjective aspect. Moreover, EXPRESS and the other approaches in the study (OWL-S and RESTdesc) are still research prototypes, which have not been used yet in practice. Therefore, as there is no user base in relation to which a questionnaire or observation can be used to assess development effort, the research method that is applicable in this case is to undertake an expert review. To achieve this, feedback was solicited on the development effort and practicality from experts, by showing them the development process in regard to a specific scenario in different approaches including EXPRESS, and asking them open-ended questions on the development effort required in these approaches.

As a follow-up to Chapter 6's matchmaking experiment, the interviews also explored the experts' opinions on the representativeness of the results of the automatic conversion that created the EXPRESS test collection (EXPRESS-TC) used in the experiment.

In this chapter, the experimental design is explained in Section 7.1, Section 7.2 presents the results and analysis, Section 7.3 discusses the results and how they relate to the research questions and 7.4 concludes the chapter.

7.1 Experimental Design

Semi-structured interviews with six experts were conducted. Each participant was presented with a scenario, which was shown in three Semantic Web service approaches (EXPRESS, OWL-S and RESTdesc). This section explains the methodology, the scenario and material design, the interview design and how the interviews were analysed.

7.1.1 Method

The main aim of the expert reviews was to get the expert's assessment of EXPRESS in terms of development effort, a sense of how it compares to other approaches, and where or if the intended simplicity of EXPRESS compromises its functionality.

The other aim concerned the matchmaking experiment in Chapter 6. EXPRESS-TC was generated from the OWL-S test collection (OWLS-TC) and used in an experiment to evaluate the semantic expressiveness of the EXPRESS service descriptions. Therefore, it is important to verify that the automatically generated descriptions are one of the possible and plausible solutions that a developer could come up with manually.

The interviews were designed in two parts. The first aim (i.e. assessing EXPRESS in terms of development effort and practicality) was addressed in part one of the interviews, while the second aim (i.e. verifying the plausibility of automatically generated EXPRESS descriptions) was addressed in part two.

For the first part, two Semantic Web service approaches, OWL-S (Martin *et al.*, 2004) and RESTdesc (Verborgh *et al.*, 2011), were selected to compare EXPRESS against. The reasons for selecting these are listed below.

For OWL-S:

1. As explained in Chapter 3, it is one of the most actively researched Semantic Web service approaches.
2. It is a W3C submission, which is indicative of a community investment and higher maturity level.

The matchmaking experiment in Chapter 6 compared the descriptive power of OWL-S and EXPRESS's semantic descriptions, so comparing the development effort provides a broader examination of the impacts of the design decisions in both approaches.

However, a difficulty arises in that OWL-S was not designed to work with RESTful Web services, and although there is one paper introducing RESTful groundings for OWL-S (Filho and Ferreira, 2009), WSDL groundings dominate the research mainstream. Therefore we also selected a RESTful Semantic Web services approach, RESTdesc. The reasons for selecting RESTdesc were:

1. Like EXPRESS it is a RESTful approach.
2. RESTdesc provides minimal descriptions and compared to other RESTful Semantic Web service approaches, uses a smaller vocabulary.
3. The research on it is still active, indicating the potential for it to mature. For example, a recent publication from the approach's author about RESTdesc was published in 2013 (Verborgh *et al.*, 2013).
4. It is a general purpose approach, compared to other RESTful approaches like LIDS (Speiser and Harth, 2011), which focus on integrating Web APIs with Linked Data.
5. Unlike some RESTful approaches such as RESTler (Alarcon and Wilde, 2010), which supports only the `GET` method, RESTdesc can support `GET`, `PUT`, `POST` and `DELETE`.

Details of how the materials were designed for parts one and two are explained in section 7.1.2.

Six experts in Semantic Web technologies were recruited from the School of Electronics and Computer Science at the University of Southampton. These experts are involved in the research and development of applications using Semantic Web technologies. Hence they had both a theoretical and practical background in semantic technologies. The experts included two PhD candidates, two research staff, one senior developer and one senior academic. The following table explains their range of expertise:

Table 20 Interviewed Experts' Areas of Expertise

Expert	Area of Expertise in Semantic Technologies
Expert one	Distributed SPARQL queries
Expert two	Ontologies for multimedia, semantic annotation
Expert three	Linked Data, annotating multimedia, and media fragments
Expert four	Publishing Linked Data, developing libraries for handling RDF and SPARQL
Expert five	Social media, semantic annotation
Expert six	Publishing and advocating Linked Open Data

The selection of experts aimed to focus on their familiarity with Semantic Web technologies in general, while also deliberately avoiding people with a high level of familiarity with any of the Semantic Web service approaches used in the

interviews. This was to reduce the possibility of their bias towards an approach they were more familiar with.

7.1.2 Scenario and Material Design

A bookstore scenario was designed; it involved retrieving a book by its ISBN, then ordering the book. The aim was to make the scenario simple, so it would be easy for the experts to focus on understanding the approaches and differences between them, and then provide feedback in a reasonable amount of time (forty to eighty minutes). Another consideration in the selection of the scenario was that the scenario involved not only data retrieval but also updating.

Having both data retrieval and updating services corresponds to the mutability requirement mentioned in Chapter 4. The atomicity requirement for RESTdesc and EXPRESS is shown in the interaction phase of the scenario, and as a composite service in OWL-S. Therefore the scenario covers two interaction requirements from Chapter 4. With regard to the other requirements: synchronicity, plurality, and roles, RESTdesc and OWL-S do not introduce mechanisms for expressing them. In addition the chosen scenario is a typical one used in the literature see for example the one used by Decker *et al.* (2008). The materials were presented to the experts on paper. The interviews involved two parts, and developing the materials for them are explained below.

7.1.2.1 Part One: Comparison of Semantic Web Service Approaches

The bookstore scenario was designed in the three Semantic Web service approaches: EXPRESS, OWL-S and RESTdesc. Both RESTdesc and OWL-S do not involve the steps in deploying the Web service, with both coming after the design and deployment phase. Because we are comparing them to EXPRESS, and it is involved in the design and deployment, it was necessary to discuss the tasks RESTdesc and OWL-S assume are done. However, it was emphasised in the material and when explaining the approaches to the experts that the service design and deployment are not part of RESTdesc and OWL-S. Moreover, the first page in each approach had a small activity diagram emphasising the different steps involved and which steps are not part of the approach itself but are assumed as being done. The figures below reproduce these activity diagrams.

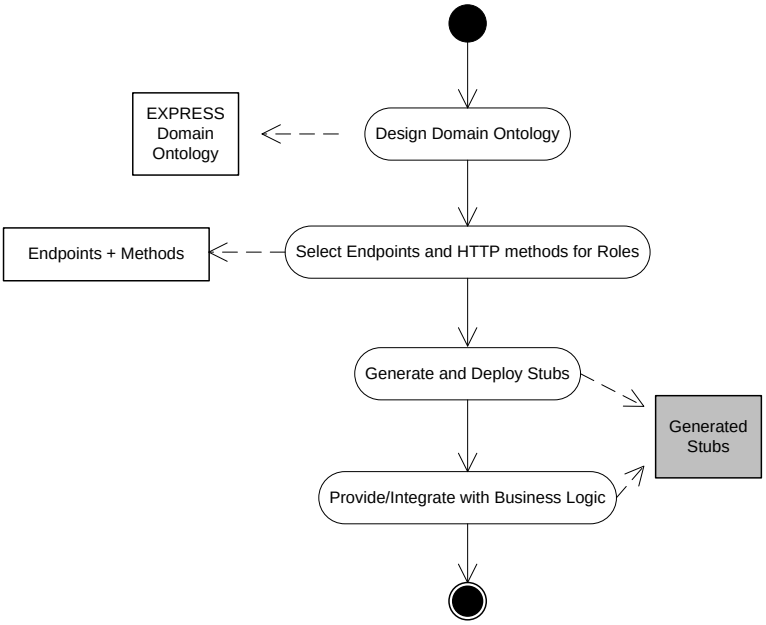


Figure 20 Activity Diagram for EXPRESS

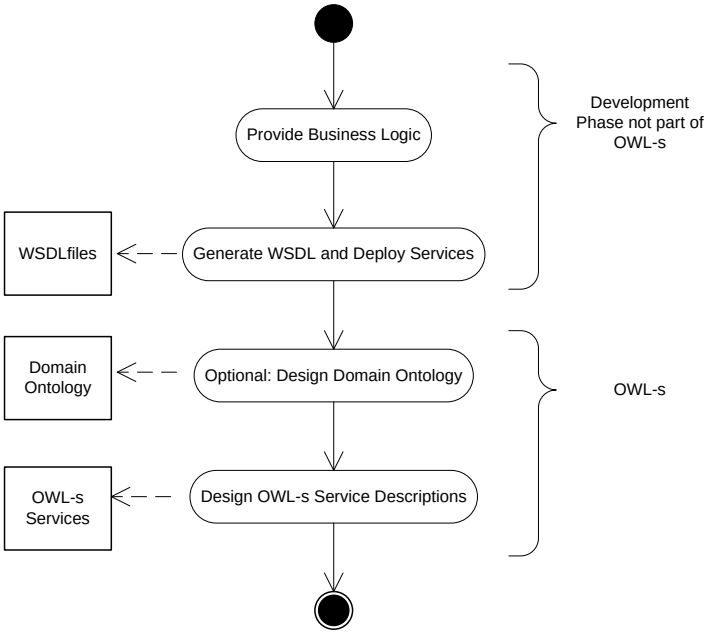


Figure 21 Activity Diagram for OWL-S

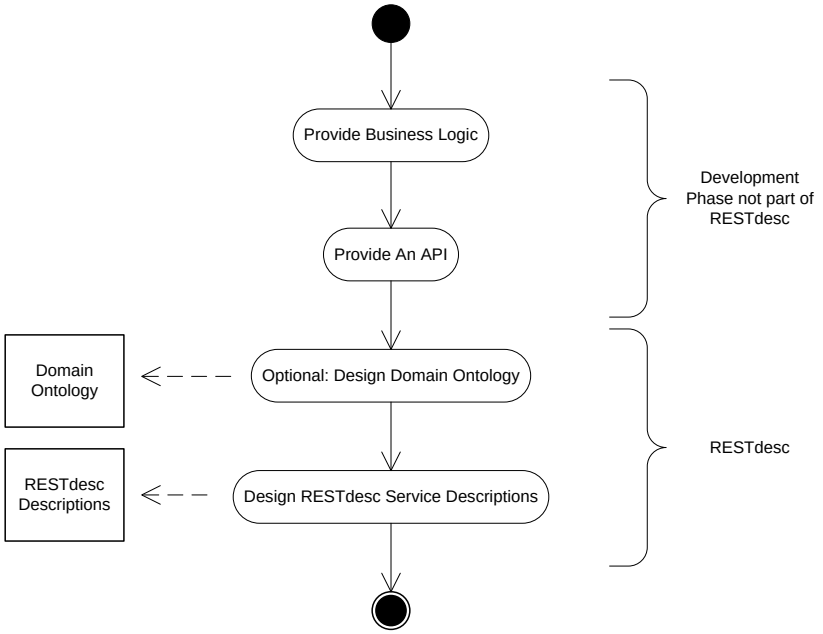


Figure 22 Activity Diagram for RESTdesc

For each approach, three main phases of the service life cycle were shown:

1. **The service design and deployment.**
2. **The semantic description.**
3. **The interaction with the client.**

The materials shown in each phase are described briefly in the following table.

Table 21 Summary of material presented to the experts

Approach	Design and Deployment	Service Description	Interaction
EXPRESS	A domain ontology containing the classes and properties relevant to this bookstore scenario, the endpoints, and a brief explanation of how EXPRESS works	None, because the description is a by-product of the deployment	The retrieval of the ontology, and the exchange of RDF
OWL-S	A brief description of OWL-S and two WSDL files, one for retrieving the book's details by its ISBN and the other for ordering a book	Two OWL-S files (one for each service) and a domain ontology	The retrieval of service descriptions, ontologies and the exchange of SOAP messages
RESTdesc	A brief description of RESTdesc, and a human-readable description of the API, as usually provided by Web APIs, including two JSON versions of the same services	Two versions of the RESTdesc descriptions in N3 rules	The retrieval of service descriptions, ontology and the exchange of JSON messages

The complete material examined by the experts is in Appendix E.

The OWL-S descriptions were generated from the WSDL files using the OWL-S Protégé plug-in. This created the structure of the OWL-S files which were then edited manually to link to the domain ontologies.

RESTdesc materials were developed by consulting its author and developer Ruben Verborgh. I contacted Ruben with an initial draft of the RESTdesc material and he suggested minor modifications. He also mentioned that there is a more recent RESTdesc version, in which URI templates are deliberately avoided, he requested that I show the scenario in the two versions of RESTdesc, I agreed because it would provide a fairer comparison, and more insight into the experts' opinions about URI templates. Ruben also answered the interview questions, which provided an initial verification of the interview questions.

The vocabularies used to describe the domain concepts such as book, author, title, ISBN, are the same across the three approaches, this was to reduce the variance between the scenario versions, making it easier for the experts to focus on the actual differences in the approaches.

7.1.2.2 Part Two: Comparing an EXPRESS description generated from an automatic conversion of an OWL-S version, to a manually written EXPRESS description.

Considering the time limitation of the interviews, and to build on the familiarity the experts gained by participating in part one, I chose to use the bookstore scenario again in part two. The service retrieving the book by its ISBN was selected, since it is a data retrieval service, and the services used in the matchmaking experiment are all considered as data retrieval services.

The OWL-S service was run through the OWL-S to EXPRESS conversion program. This provided one version; the other version was the EXPRESS version of the 'retrieving the book by its ISBN' service created for part one.

7.1.3 Interview Design

The process of the interview went as follows: I asked the participants to sign a consent form, after they read the participant information sheet. The interviews were conducted individually with each participant, and the interview was recorded. They were between forty to eighty minutes long. In the first part of the

interview, the experts were shown and walked through the materials of the three Semantic Web service approaches that were discussed in the previous section. They were given time to read them and enquire about issues they did not find clear. They were then asked the three open-ended questions for part one, discussed in the next paragraph. After that they were shown the material for part two, and asked the last interview question.

To design the interview questions effectively, they are derived from the research questions. The interview questions are listed below and their mapping to research questions is shown in Figure 23.

Part One
Question One: **Using EXPRESS means that the URIs of your services will be generated automatically, how might that affect the flexibility and ease of deployment?**
Question Two: **You are required to provide a Semantic API for a bookstore, to provide information about books and search for books by title or author. If you had to use one of these approaches, how long would it take you?**
Question Three: **Imagine you were developing clients for those services, how would you describe the descriptions in terms of** **1. Practical quality: ease of use, development speed** **2. Semantic quality: semantic richness, ability to infer over**
Part Two
Question Four: **Given these two EXPRESS descriptions how similar/different do they seem?**

Question four is not linked directly to the research questions, and therefore is not present in Figure 23. However it is related indirectly to the third research question, because it aims to assess the representativeness of EXPRESS-TC used in the matchmaking experiment, and the experiment was designed to answer the third research question.

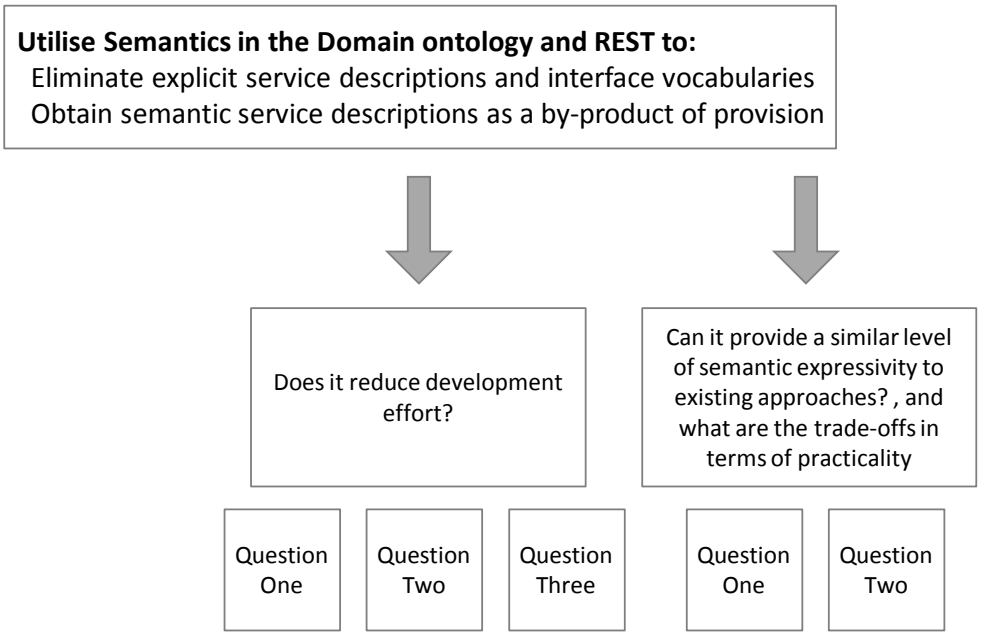


Figure 23 Derivation of interview questions

Questions Two and Three were designed so that the experts would need to think about using these approaches to design a specific service and a client, respectively, and hence, make it easier for them to provide a fairly grounded judgement.

The interviews were semi-structured, and the questions were open-ended questions, so follow-up questions were asked. For example, in Question 4, after the experts had tried to compare the two versions of the service and listed some similarities or differences, they were asked the question, "If I explained how EXPRESS works to a developer, which one of these two examples are they more likely to come up with?"

7.1.4 Interview Analysis

The interviews were qualitatively analysed, involving the following steps:

1. **Transcribing the interviews.**
2. **Reading the interviews and highlighting individual quotes that appeared related to research questions. These individual quotes were numbered sequentially for cross referencing: for example 5-12 indicated that it was quote number 12 from the 5th expert.**
3. **Deciding on preliminary codes, from highlighted text and research questions.**
4. **Three transcripts were coded with preliminary codes, then used to code the rest of the interviews, adding new codes when needed. The quotes**

were copied into a spreadsheet, and marked with a code and corresponding theme.

5. Arranging the codes into themes.
6. Categorising the quotes according to the themes.
7. Summarising the arguments and opinions in each theme.
8. Identifying agreements or disagreements between experts, and their explanation for their opinions and unexpected and interesting comments.
9. Examining and analysing the issues identified in step 8, to draw out results.

A sample of an expert review transcript is in Appendix F. Screenshots of the coded transcript document and the spreadsheet are available in Appendix G.

7.2 Experimental Results

This section presents the results of the experiment, section 7.2.1 discusses the different themes that emerged from the interviews, and section 7.2.2 provides a description for each theme and overview of the experts' responses.

7.2.1 Themes

Nine themes were elicited from the transcript analysis. The table below lists the themes and the number of quotes about them; some quotes were categorised under more than one theme. Some themes were discussed by all the experts, such as Ease of Development, Flexibility, Manual vs. Automatic Descriptions, Semantic Quality. An interesting theme that arose unexpectedly was, "The aim of SWS": in which experts questioned the practicality of SWS in general. In total there were 136 coded quotes, and since some discussed more than one theme, the total number of quotes was 179.

Table 22 Themes and the number of quotes about them

Theme	# of quotes	# of quotes about the theme from expert					
		one	two	three	four	five	six
Development Speed	25	3	8	7	-	3	4
Ease of Development	64	7	15	3	16	10	13
Flexibility	16	2	2	1	4	5	2
Linked Data	5	-	1	1	2	-	1
Man. vs. Automatic Descriptions	16	2	2	1	4	5	2
Semantic Quality	24	3	9	1	5	1	5
The aim of SWS	13	-	-	-	7	3	3
Underspecified	11	-	1	3	7	-	-
Extra features	5	-	-	2	2	-	1
Total	179	17	38	19	47	27	31
Total # of quotes from experts	136	11	26	15	36	21	27

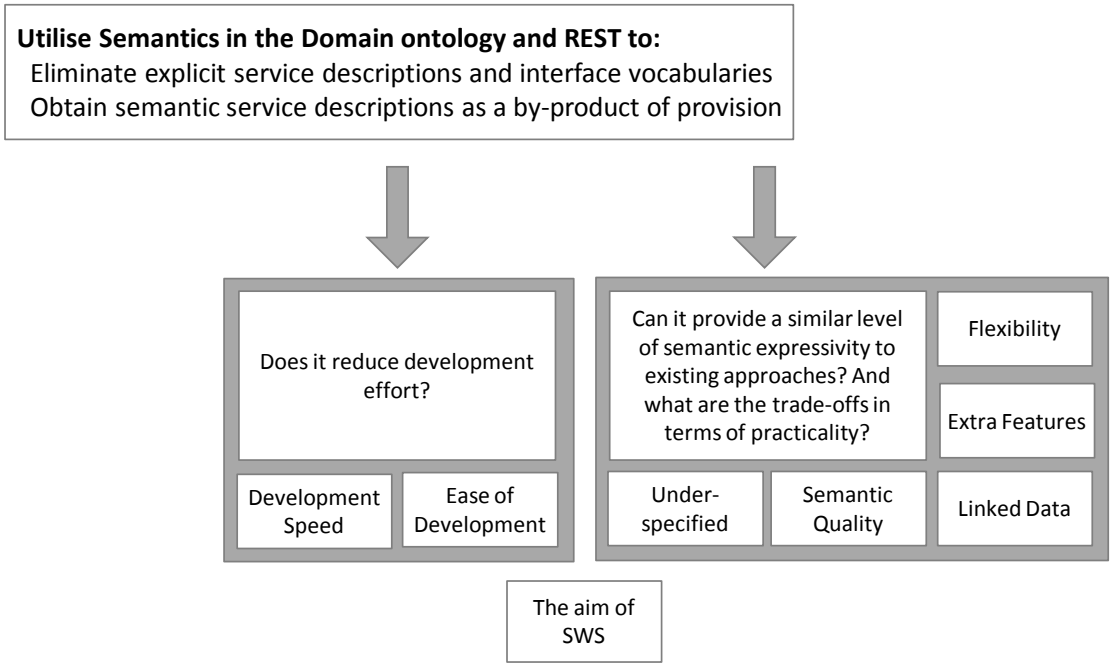


Figure 24 Themes related to research questions

Figure 23 is similar to Figure 24; however, instead of the interview questions, it shows the themes. It also shows how they are related to the research questions; this relationship is manifested in the discussions in section 7.3.

7.2.2 Summary of Experts' Responses by Theme

7.2.2.1 Theme 1: Development Speed

Development Speed refers to what the experts thought about the time it would take them to develop a Semantic API and/or a client in the three approaches presented to them. The focus of the question was on the first time they would develop in these approaches, so it involves the learning time.

Expert one preferred RESTdesc because it uses N3 rules, and thought it would be very fast to develop a Semantic API or a client. He also thought that EXPRESS would be very fast too, but felt it provided less semantic quality. As for OWL-S as he thought it was "heavy duty". Expert two thought that development times in ascending order would be RESTdesc, EXPRESS and then OWL-S. He thought that EXPRESS would be slower than RESTdesc, because you need EXPRESS in mind when developing, and that OWL-S would be slowest because WSDL services are more complex and need more debugging time. He also stated that developing clients in OWL-s may be quicker because of WSDL/SOAP tool support, and the exchange would be in SOAP messages not RDF.

Expert three said, “OWL-S will take me a really long time”, and when comparing EXPRESS and RESTdesc, he perceived that if a new service was added, a new RESTdesc description would be needed. However, with EXPRESS, the same ontology would be used. Expert four was enthusiastic in discussing the aim of SWS in general (one of the themes that emerged from the analysis), and avoided commenting on the development speed in particular.

Expert five mentioned that the OWL-S will take him the longest, and attributed that to it being built on WSDL descriptions. Comparing EXPRESS to RESTdesc, he said that, if the services were built from scratch, EXPRESS would be the fastest. Expert six agreed with the other experts about OWL-S. He said starting from scratch EXPRESS is much simpler. He also said that he would be able to understand RESTdesc quickly, but in terms of typing, it would take him a long time.

7.2.2.2 Theme 2: Ease of Development

Ease of development, encompasses aspects concerning the comprehensibility of the approaches and the effort required to learn and develop solutions in them.

Expert one felt OWL-S required a lot of work to provide OWL-S descriptions. He preferred RESTdesc, and described it as tidy, neat and very straightforward to build on top of HTTP APIs. He mentioned that EXPRESS would be very convenient in a small organization and a relatively simple service. However when things scale, it wouldn’t be very convenient, because having all the possible links in the header is a constraint. He concluded by saying that EXPRESS would be convenient for a beginner to semantic technologies, but because he is not, he prefers RESTdesc. Expert two also regarded OWL-S as more complex than EXPRESS and REST. However, he also stated that it depends whether you are building a project from scratch. In that case, both OWL-S and EXPRESS would be suitable, because RESTdesc “doesn’t rely on the business logic so much, which I guess is good; it is a lot simpler to work with”. He liked the way RESTdesc used the implies “=>” to define the services, and thought it was “simpler and cleaner”. However, he mentioned that one of downsides compared with OWL-S was dealing with the URI templates. For EXPRESS he compared it to a schema: “So you’ve got it on top of the schema, so once you’ve got the schema there, you can control it the way you want”. However, in the order book example, he considered passing a URI as part of the URL to be complex. In terms of creating clients, he mentioned that needing an RDF handling library for EXPRESS adds extra complexity, whereas in RESTdesc it would be easy, because there are many libraries that support JSON. He also stated that it depends on the programming languages used. An issue with OWL-S

he considered complex was translating the messages from XML to RDF, and OWL-S provided more semantic information but it was less easy to explore.

Expert three viewed EXPRESS as a "very standard expression of the API", and thought it was very simple compared to OWL-S and RESTdesc. He felt there would not be a problem implementing it. He liked that RESTdesc returned JSON and suggested that EXPRESS provide content negotiation to provide JSON, too. Expert four preferred EXPRESS because it is succinct and less verbose, meaning fewer errors, and also because it uses CRUD. However, he noted that people will find the equivalent classes hard to learn, and although he preferred EXPRESS to RESTdesc and OWL-S it is still hard, and he added, "Why should I bother marking up my endpoints with it?" This point of view is discussed in Theme 6: The aim of SWS. He commented that OWL-S was very verbose, and he thought the XSLT conversions for grounding were verbose, fragile and were neither readable nor debuggable. As for RESTdesc, he was not familiar with the implies (=>) in N3 and thought that the N3 descriptions were not clear enough to state that when a book is retrieved that it was not actually created then retrieved.

When explaining EXPRESS in the beginning, Expert five asked about the stub generation and commented that EXPRESS, compared to RESTdesc and OWL-S, is a much simpler and nicer system and that EXPRESS is for building a service from the ground up. He mentioned that OWL-S would be the hardest to deal with. He summarised his opinion in the following quote: "What would I develop in, if I was writing it from scratch? Yes, I would write in EXPRESS. But what would I expect to be more useful in the real world? RESTdesc. And what I think is, we should never ever use OWL-S, WSDL is such as waste of time". Expert six highlighted several issues with RESTdesc. One was the ambiguity of version two of the scenario, where a `POST` on a book's URI created an order; however, he preferred version two because he thought that the templating in version one was challenging. This is because it is encoded in strings and therefore, it is harder to debug, as a mistake would not be picked up by an RDF parser. He commented, "What is the support that is going to help me get that right and not get bugs in it? On a very pragmatic level, what happens when I make a syntax error?" When comparing EXPRESS to RESTdesc, he mentioned that version two of RESTdesc (which has no URI templates) looked easier and is probably comparable to EXPRESS.

7.2.2.3 Theme 3: Flexibility

This includes the experts' opinion on the flexibility of the approaches: whether EXPRESS was less practical than the other approaches because of the way it controls the structure of the URIs.

Expert one preferred RESTdesc in terms of flexibility, and for EXPRESS, he regarded having all the possible links in the header a constraint. Expert two pointed out that RESTdesc does not rely on the business logic, compared to EXPRESS, and regarding the URIs being controlled by EXPRESS, he said, "It is nice to have some control on the URIs", indicating that EXPRESS was restrictive and that makes it less application-specific. Expert three suggested adding content negotiation and returning JSON. Expert four talked about adding sub-objects and reverse properties to the specification, to make EXPRESS more flexible, and when asked about the way EXPRESS controls the URI structure, and whether or not it was a limitation, he said, "Definitely yes, limitations are how you survive, limitations are fine if people can see them for what they are".

Expert five, in the beginning of the interview, said, "It is clear that RESTdesc is more flexible, but as I was saying earlier on, being flexible doesn't give you that much help". This was because he was comparing it to the generation of stubs that EXPRESS offers. Commenting on EXPRESS's control of URIs, he said, "At least it will be semi-standard, at least folks will start to understand what they could expect from your Web service [...] in a way, it will be more predictable". However when I explained that the N3 RESTdesc file does not need to be in a specific location, he picked up on the fact that both RESTdesc and OWL-S can be used to describe third-party services. "So that is what is interesting about this, because I could write a set of N3 rules for a third party API like Twitter, for example, and it will describe what the Twitter API means in the semantic sense, OK, which is quite cool". He also pointed out that EXPRESS is different because it develops Web services from scratch: "Well, clearly EXPRESS is for building services from the ground up, to be in some sense semantically aware. Obviously, if you had a handful of EXPRESS services, that would be remarkably useful". Expert six did not regard EXPRESS's control over the URI structure as limiting, and commented, "So, as a service provider, you are going to provide a system where I can just throw this [the OWL file] at it, and it generates this [the services]. Sounds great to me. [...] So essentially I just edit an RDF description of my service, which is nice and flexible. If I want to change my service I edit the RDF, and press the button again, so that is very flexible, isn't it?" When I explained that EXPRESS imposes a certain URI structure that cannot be changed, he said, "Oh, I don't want to do that [change the URI structure], why do I want to do that? That is the last thing I want to do, I've got customers who care about my URI structure. As a Linked Data person, one of the things I know is the first thing you need is to work out your URI structure [...] you need to get them right first time". He further explained that URI structure is good because it becomes a language that users can learn and have

expectations about, and added, "If you give the service provider complete freedom, then it is harder for the user".

7.2.2.4 Theme 4: Linked Data

The Linked Data theme includes what the experts said about the relationship between Linked Data and the presented approaches.

Expert two regarded RESTdesc as a way to layer Linked Data on top of an API. Expert three asked whether all the URIs returned by EXPRESS were resolvable and followed Linked Data Principles. He further explained the question by asking about the implementation details, and how the URI structure that EXPRESS imposes can be propagated to the backend storage systems. He drew parallels to the D2R[18] server and how to provide the correct URI structure on the fly. Expert four saw similarities between EXPRESS and the Linked Data API[19]. The Linked Data API provided an RDF configuration file to specify the API and the endpoints to retrieve and format Linked Data. Expert six compared the way EXPRESS controls the URI structure to the process of minting URIs for Linked Data.

7.2.2.5 Theme 5: Semantic Quality

This includes the experts' opinions about whether the descriptions were unambiguous, and the ability to infer over them.

Expert one regarded EXPRESS as a more structured way of providing HTTP URIs; however, he said "it doesn't provide much semantics". On the other hand, for RESTdesc, he said, "You can see the potential of providing semantics here". Expert two initially regarded RESTdesc as not as semantically rich as the other two and said that compared to EXPRESS it requires more work to investigate its richness and to handle the logic. However, he later stated that the three approaches provide similar information about inputs and outputs, and that RESTdesc is simpler and cleaner, and while OWL-S would probably provide more semantic information, it is less easy to explore. As for EXPRESS, he liked the URI structure and that it provided fine-grained access, and he also appreciated retrieving the RDF data: "I guess EXPRESS is good because you get the raw RDF back, so you can actually put into a reasoner, you have access to the domain ontology on top of that". However, he thought the lack of explicit process definitions was a downside. Expert three discussed tool support for parsing semantic information and how it depended on the programming language used. If it was PHP or Java,

[18] D2R Server http://d2rq.org/d2r-server
[19] Linked Data API https://code.google.com/p/linked-data-api/

there would not be a problem. He also highlighted as a factor the developer's familiarity with semantic technologies.

Expert four said that RESTdesc does not state whether a service has a side effect or whether it is merely a query. He stated that EXPRESS was missing a human-readable description, and also, as mentioned before, he thought EXPRESS was missing reverse properties and sub-objects as well as the ability to describe complex data structures. Expert five thought that these approaches were similar in terms of the semantic descriptions they provide, as they all described the inputs and the outputs and how to interact with the service. Expert six stated that three approaches describe how to interact with the service, but not what the service actually does, he suggested using Good Relations[20] to describe the type of business the Web service represents. With RESTdesc he regarded the way version 2 works as ambiguous, because it meant POSTing to a book's URI to create an order. He also noted that RESTdesc was enforcing the use of their template ontology, which might not work for him and that he required something simpler. As for EXPRESS, he said, "I have a suspicion if I was to gather descriptions from a number of places and put them in a store and then try and do clever reasoning, this will be the hardest, this has the least information [...] normally when people write clients they don't do that, there is nobody doing that, everybody just wants this information so they can write their PHP or Python to use it, and that is probably why this [EXPRESS] appeals to me more because that tends to be what I do".

7.2.2.6 Theme 6: The aim of Semantic Web services

This includes comments about what experts thought of the viability of Semantic Web services in general. This is an interesting theme as it shows that some experts value practicality over semantic richness, and that the advantages of SWS are not of value to them.

Expert four was discussing the aim of Semantic Web services (SWS) and was sceptical of their value. He said, "The problem is, my gut says that this starts with a solution rather than starting with a problem, and this feels very academic, this doesn't feel like someone who has got a problem and is trying to solve it, it feels like somebody is writing a paper". He went on to say, "Well all these are solving a question that I didn't think anyone asked. That is the problem. It seems a long way ahead from where the actual real-world problems are". When I explained that SWS aimed to offer automatic discovery and composition, he said, "No one has ever asked for one of these, no programmer has ever said: Why don't you have an

[20] Good Relations http://www.heppnetz.de/projects/goodrelations/

auto-discovery mechanism for your APIs?" and "When you say discover a service, and who would want to discover a service, why would you? I mean I wouldn't trust something that said it could do something, I am much more interested in knowing about the people who wrote it, is this John Smith's third-year project or is it Amazon or the British Library? If it is the British library I am probably going to be more interested in using their API".

Expert five initially doubted the usefulness of Semantic Web services. He said, "Like in reality that is not how you would interact with these Web services, right [...] you are not building like this robot and you are telling the robot, 'Hey I want to order a book and that's it', and it goes, 'OK, I know what a book is, I know what it means to order a book, here is my list of my Web services, can you do this? Can you? Oh you can do it OK'". However as he tried to think of applications, he seemed to realise the benefits of semi-automation: "Practically speaking, you are going to be writing a client that can interact with a particular service, or a set of services, and to tell the client how to interact with them individually, so that is where the richness comes in, that is where the benefit comes in, so here you have a generic behaviour which works across a set of services, and as long as the services tell you how to interact with them potentially your client can make sense of that and interact with it in a way that is useful for its task". He went further by providing a use case from his experience, where he saw that RESTdesc provided a better solution. "Say you are doing social media analysis, say you are interacting with these five or six different social media platforms and under the platform they all have users and the users will have geolocations, and if somehow you could interact with a semantic layer of these services and these services tell you [...] this is the information we provide and then your client can go through [...] so if I want a service that provides geolocation this is what I have to do for Facebook, this what I have to do for Flickr and this is what I have to do for Twitter, and then that means that you can write a client that sits there and churns through user geolocations, but exactly how it gets it from each service is done automatically. That is quite cool, you can imagine that saving a load of work. Again, that is if they all provided that semantic information, but I suppose this is what RESTdesc gives, the ability to describe that semantic information for services you didn't write".

Expert six believed SWS described how to interact with the service. He was sceptical they captured the actual semantics. For example, in creating an order, he asked, "The other thing that is missing from all this, I don't actually know what the service is doing, nobody attempts to tell me what the service does, in some sense what is an order? Does an order buy me a chicken or does it sell something,

or give me a description of something?" I explained that this is resolved by agreeing on vocabularies or ontologies to describe shared concepts. However, he still continued to consider SWS as an interaction layer, and suggested adding a Good Relations description for what the service does, and a mechanism to advertise trust.

7.2.2.7 Theme 7: Underspecified

Experts' opinions about aspects that were missing from EXPRESS are included in this theme.

Expert two asked if EXPRESS gave any other filtering options, such as searching for a keyword within the text, and also asked about sorting options. Expert three, when discussing EXPRESS, questioned the lack of complex data structures in the examples and asked how they will be encoded and transferred. He also thought that the author should be a data property, not an on object property. He asked about whether there were guidelines for writing the ontology. Regarding EXPRESS, expert four asked about error handling and what will happen if someone tried to add arbitrary triples. He discussed the lack of explicit support for sequences, containers, sub-objects, and reverse properties. With regards to RESTdesc, he asked if there was a mapping between the returned JSON and the N3, and said, "Well if the client can't tell that the ISBN here [the N3 description] is the ISBN here [in the JSON response], then it is useless, you know you are not getting anything semantic you are only getting JSON, you might as well have done a `GET` query for. I don't see how this will work".

7.2.2.8 Theme 8: Extra features

Any extra features that were suggested by the experts are included here.

Expert three was interested in how EXPRESS would interact with a conventional database and asked if there was an association between the database design and the interface design. He suggested the option of creating the ontology from the database schema; he also discussed methods for generating the URIs for the entities in the database and suggested looking at the D2R server and integrating it with EXPRESS. He also suggested versioning for the endpoint and URIs to maintain backward compatibility. Expert four suggested adding mechanisms to query for reverse properties, and also emphasised the importance of trusting a service. Expert six also suggested adding a mechanism to convey the trust level of the service, "and then you might have something like this is my trust service where you can find something about my trust".

7.2.2.9 Theme 9: Manual vs. Automatic Descriptions

This includes the experts' opinions about the differences between the manual and automatically generated version of EXPRESS.

Expert one thought both descriptions were equally possible, and that it depends on the publisher's preference. He also thought both have the same semantic power. Expert two highlighted the syntactic differences such as the underscore and the different namespaces. He also noted the existence of extra properties the manual description. However, he was not very decisive, and seemed to agree on the semantic similarities, and remarked that the automatic one takes more work.

Expert three highlighted the syntactic differences and different properties. He thought that the automated one was more flexible and the manual one was easier for implementation, and when asked which one a developer is more likely to come up with he said it depends on the complexity of the problem; in example two [the automatic one], it just returns the book, whereas in example one [the manual one], it returns its attributes, too. Expert four thought the underscore was ugly, and said about the automatically converted version, "The ontology in example two [the automated one] looks like it describes a single lookup. This seems slightly more verbose for a worst result", and "This second layer of stuff has its strength and weaknesses, the strength being separating your ontology from your markup, as they are two different things." However when asked which one a developer is more likely to come up with, he said, "Well, I don't know from the information available. To be honest I've only seen fairly small amount".

Expert five also highlighted the syntactic differences and the difference in properties, and when asked which one a developer is more likely to come up with, he said, "I think this really depends on what the developer understands the application to be, so if the developer says 'OK, all we want you to give you back is a unique URI of the book', then I understand most applications don't care about the author, they don't care about the title, they just want to know they can get a unique URI of a book, so I'll just tell them that [...] Alternatively, if a developer knows that, OK, the reason most of the time people ask for a book is they want a title, to put on a website somewhere then I should tell them". However, he later said, "It is interesting that these descriptions are technically both valid, you can then go off and resolve that book URI and get the extra information if you wanted, it is interesting how different they are". Like the other experts, Expert six highlighted the syntactic differences and the difference in properties, and when asked about which one a developer is more likely to come up with, he said, "If they are very keen on ontologies, this one [the automatically converted one], but

the standard developer will understand this better [the manual one]". However, as he spent more time examining them, he said, "Ah sorry, this just looks so nice and clean [the manual one], I can't see why anyone would do that [the automatically converted one] if they can do that".

7.3 Discussion

In this section the results of the interviews are presented and discussed, Section 7.3.1 uses the interview results to answer the research questions and Section 7.3.2 discusses the link between the expert's expertise and their views.

7.3.1 Research Questions

The research questions are answered by synthesising the results from its associated themes, in addition to analysing the experts views on the representativeness of EXPRESSive descriptions that are automatically converted from OWL-S versions.

7.3.1.1 Does EXPRESS reduce the development effort?

The themes associated to this question are: Theme 1: Development Speed and Theme 2: Ease of Development. The experts agreed that developing in EXPRESS or RESTdesc is both faster and easier than OWL-S; they considered OWL-S verbose, complex, and harder to debug. Moreover, being built on WSDL descriptions increases the complexity. However, their opinions differed when comparing EXPRESS to RESTdesc. Half of the experts preferred RESTdesc (Experts one, two and five) and the other half favoured EXPRESS (Experts three, four and six). Below I present the experts' opinions on the pros and cons of RESTdesc and EXPRESS in terms of development effort.

Table 23 Expert opinions on development effort

	Expert Opinions	
	PROS	CONS
RESTdesc	• Follows N3 which is a widely accepted format **(Expert 1)** • Would be very fast for providing a building on top of HTTP APIs **(Expert 1)** • Tidy, neat and straightforward **(Expert 1)** • It would be the fastest for developing semantic APIs **(Expert 2)** • Use of N3 implies symbol (=>) makes service definitions simpler & cleaner **(Expert 2)** • Easier if you have an existing API **(Experts 3, 5)** • Less work than OWL-S and EXPRESS, just hosting an N3 file **(Expert 5)**	• A new RESTdesc description has to be added each time a new service is added **(Expert 3)** • Dealing with URI templates is difficult **(Expert 2, 6)** • Typing the descriptions will take a long time **(Expert 6)** • Use of N3 implies symbol (=>) to define the services is intimidating **(Expert 4)** • In version 2 of RESTdesc, creating an order by POSTing to a book URI is unexpected **(Experts 4, 6)** • String encoding of the URI templates, makes it harder to debug **(Expert 6)**

EXPRESS	• Convenient in a small organisation **(Expert 1)** • Convenient for a beginner in semantic technologies **(Expert 1)** • Faster and easier for building an API from scratch **(Experts 1,2,5,6)** • Simple compared to RESTdesc and EXPRESS **(Expert 3)** • Only an endpoint has to be added each time a new service is added **(Expert 3)** • Succinct and achieves goals with less verbosity **(Experts 4,6)** • Easier to debug **(Experts 4,6)** • It can be completely automated, simpler and nicer **(Expert 5)** • Easier to understand: less cognitive models required **(Expert 6)**	• When it scales, it is not convenient to have all the possible links in the header **(Expert 1)** • You have to have EXPRESS in mind when developing for it **(Expert 2)** • Passing URIs as part of the URL is complex **(Expert 2)** • An RDF handling library would be needed to parse the results **(Expert 2)** • EXPRESS has equivalent classes in the ontology which people will find hard to learn **(Expert 4)** • The use of query strings complicates EXPRESS **(Expert 6)**

To answer the question, EXPRESS was clearly perceived to reduce the development effort compared to OWL-S; however, compared to RESTdesc, there is a consensus from the interviewed experts that EXPRESS only reduces the development effort for developing an API from scratch, and this is evident from the number of experts who have mentioned this explicitly, even if they preferred RESTdesc.

7.3.1.2 Can it provide a similar level of semantic expressivity to existing approaches? And what are the trade-offs in terms of practicality?

Referring back to Figure 24, the related themes to this question are Theme 3: flexibility, Theme 4: Linked Data, Theme 5: semantic quality, Theme 7: underspecified, and Theme 9: extra features. Starting with OWL-S, in terms of semantic quality, Expert two said that although the three approaches provide similar content about the service, OWL-S would probably provide more semantic information, but it is less easy to explore. In terms of flexibility Expert five noted that the service descriptions could be written by third parties, which makes it useful for Semantic Web experts interacting with existing APIs. In general the experts found OWL-S overwhelming and mostly dismissed OWL-S from the comparison, by providing short comments on its complexity. As for RESTdesc and EXPRESS, expert opinions were divided, as they were about development effort. Experts one, two and five preferred RESTdesc and Experts three, four and six preferred EXPRESS. Table 18 summarises their opinions.

Table 24 Expert opinions on semantic expressivity and practicality

	Expert Opinions	
	PROS	**CONS**

RESTdesc	• Experts would prefer RESTdesc to write service descriptions **(Expert 1)** • It has a better potential for providing semantics **(Expert 1)** • Does not rely on the business logic **(Expert 2)** • A good way of layering Linked Data on top of your service **(Expert 2)** • JSON is better supported **(Expert 3)** • Useful for writing descriptions for third party APIs **(Expert 5)** • RESTdesc has more semantic information about the service **(Expert 6)**	• There is ambiguity in determining if a service has a side-effect **(Expert 4)** • It is not clear how RESTdesc will deal with objects such as lists and containers **(Expert 4)** • No mapping between the JSON responses and the N3 descriptions, hence less useful **(Expert 4)** • RESTdesc V2.0 POSTing to a book's URI to create an order is ambiguous **(Experts 4,6)** • Enforces ontologies for service descriptions, (HTTP and HTTP template vocabulary) **(Expert 6)** • Although it provides more semantic information, that information is not useful **(Expert 6)**
EXPRESS	• Raw RDF is returned, so it can put in the reasoner with the ontology **(Expert 2)** • It is more predictable, you know what to expect from a service **(Experts 3,4,5,6)** • Having the URI structure controlled is a limitation, but a good one **(Expert 4)** • It could become a semi-standard **(Expert 5)** • It is flexible, all is needed is editing the ontology **(Expert 6)** • Once the URI structure is right it should not be changed **(Expert6)** • The information EXPRESS provides is more useful for writing clients **(Expert 6)**	• Does not provide much semantics **(Expert 1)** • Would be hard to scale if all the URIs are in the header **(Expert 1)** • Not having control over the URI structure would be restrictive **(Expert 2)** • Does not provide definitions for the services **(Expert 2)** • Does not support sequences and containers **(Experts 3,4)** • No human readable description **(Expert 4)** • Does not support sub-objects or reverse properties **(Expert 4)** • Cannot be used for 3rd party services **(Expert 5)**

Most of the interviewed experts agreed that RESTdesc in general provided more semantic information than EXPRESS. However, they considered differences between the semantic information offered minimal. Experts who preferred EXPRESS saw that any more semantic information provided was unnecessary.

In terms of flexibility, in general, they also considered RESTdesc more flexible. Expert two appreciated that it did not rely on the business logic, and Expert five was particularly keen on its potential for describing APIs of third parties. The experts who preferred EXPRESS did not see the flexibility of RESTdesc as useful, and appreciated the predictability of EXPRESS.

Experts also discussed areas where EXPRESS was underspecified, such as in providing sequences, containers, sub-objects, reverse properties and error handling.

Extra features suggested by the experts included aspects such as advance filtering, content negotiation, trust, versioning, and the association between the database design and the interface design.

7.3.1.3 Is an EXPRESSive service description that is automatically converted from an OWL-S version comparable to a EXPRESSive service description designed manually?

The experts agreed that there were syntactic differences between the two versions, like the underscore and the different namespaces, which were a result of how the conversion was implemented and could have been changed easily.

In terms of semantic differences, they also agreed that the results would be similar. They highlighted the fact that the manual one specified the return of extra information (properties), but that the automatically generated one could also do this.

As for the likeliness of a programmer to come up with the automatic version, there were mixed responses. Experts attributed the differences between the two versions to differences in the developers' style. These experts described a developer who would come up with the automated version, as someone who is keen on ontologies, lazy, reflecting their understanding of the problem complexity, or attempting to separate the ontology from the markup.

As for individual responses, Expert one thought both versions were equally possible, Experts three and Expert five said it depended on the developer's understanding of the application: whether only the book was required or its attributes (title and author) were required too, and that either way the former leads to the latter, so once the book's URI is retrieved, other attributes can be retrieved too. Their responses indicate that they believed it is plausible that a developer could have written the automatically converted version.

Experts two and four were uncertain. After discussing the syntactic and semantic aspects of the two versions, Expert four explicitly said, "Well, I don't know from the information available. To be honest I've only seen fairly small amount". Expert two kept repeating the syntactic differences and was hesitant to give a verdict.

Expert six, began by entertaining the plausibility of a developer producing the automatically converted version. However he ended his observation by saying, "Ah sorry this just looks so nice and clean [the manual one], I can't see why anyone would do that [the automatically converted one], if they can do that"

In general, the experts found the fourth question difficult to answer, and seemed to seek guidance and approval for their answers. They also asked questions such as, "So what am I looking for?", and "Is that correct?" However I emphasised that their opinion is what matters, and gave neutral responses.

The experts did find it plausible that a human developer would have created something similar to the automated description, but that there was less certainty over whether this was likely.

7.3.2 Area of Expertise Influence on Results

As mentioned in section 7.3.1 half of the experts preferred RESTdesc (Experts one, two and five) and the other half favoured EXPRESS (Experts three, four and six). In this section, I reflect on how their area of expertise influenced their preferences.

Expert one is researching effective methods for distributed SPARQL queries. This involves intensive study of graph patterns. Service descriptions in RESTdesc are in N3 rules; an N3 rule constitutes two graph patterns, one for the rule head and the other for the body. Expert one described RESTdesc as tidy and neat, and said if he was a beginner in Semantic Web technologies he would have preferred EXPRESS. However, since he is an expert, he prefers RESTdesc.

Experts two and five are involved in researching effective methods for multimedia retrieval. They have both worked on designing ontologies for multimedia, and providing mechanisms for annotating its data. Expert five, who also researches social media retrieval, appreciated that RESTdesc descriptions can be written by 3rd parties.

Experts four and six, who preferred EXPRESS, are both heavily involved with Linked Data, developing systems and discussing standards. They expressed their familiarity with the concepts EXPRESS incorporates, such as minting URIs and RESTful interaction. They discussed projects they worked on which had similar concepts to EXPRESS. Expert three, who preferred EXPRESS too, works on using Linked Data to publish information about media fragments.

Therefore, it seems that the interviewed experts who had worked on publishing Linked Data preferred EXPRESS, and appreciated its applicability.

A possible extrapolation from the results would be that experts who tended to develop APIs for Linked Data preferred EXPRESS, and those who used existing APIs preferred RESTdesc.

7.3.3 Related Results

A research paper by Bachlechner and Fink (2008) involved surveying experts' opinions on Semantic Web services. The main aim of the research was to collect opinions from both practitioners and researchers about the potential of Semantic

Web services as integration architectures. The authors conducted a Delphi study with experts from industry and academia. The study involved providing the experts with two questionnaires, in two stages. The first questionnaire contained open-ended questions to gain experts' views on Semantic Web services in general. In the second stage, the results from the first were used to design a second questionnaire, where experts were asked to rate statements on a scale from 1-5.

Amongst the challenges that the experts agreed on, was the grounding of the research in reality, and the proof of cost-effectiveness. The author suggested that the industry is not yet convinced of the potential of Semantic Web services.

This finding is in line with some of the comments from the study conducted here. Experts four, five and six were sceptical about the practicality of Semantic Web services in general.

Expert four, for example, said: "The problem is, my gut says that this starts with a solution rather than starting with a problem, and this feels very academic, this doesn't feel like someone who has got a problem and is trying to solve it, it feels like somebody is writing a paper". He went on to say, "Well all these are solving a question that I didn't think anyone asked. That is the problem. It seems a long way ahead from where the actual real-world problems are".

Another similar finding from the paper, in explaining the lack of industrial adoption, respondents mentioned "the lack of skilled developers and effective tools". Expert four from the interviews also provided the following comment: "making the Semantic Web more accessible for programmers who don't have PhDs. So one in 50 computer science graduates may know about this stuff, or even one in 10: it is not enough, it is not enough to make this technology stable, so we have to make it as easy as possible to do it badly and until the people who knock up WordPress sites can make bad RDF links, we are not there".

"High complexity" was one of the challenges that both academics and practitioners in the paper ranked high. This coincides with the experts' view of OWL-S as they preferred RESTdesc and EXPRESS because OWL-S was too complex.

7.4 Conclusions

This chapter discussed the Expert Review experiment: the methodology and results. The aim was to evaluate EXPRESS as a Semantic Web service approach in comparison to two other approaches: OWL-S and RESTdesc. Six experts in Semantic Web technologies were recruited and presented with a scenario of providing a Semantic Web service designed in each one of the approaches.

The experts' preferences were divided between EXPRESS and RESTdesc. The experts who preferred EXPRESS have expertise in publishing Linked Data, and they discussed similarities to Linked Data, this suggests that their familiarity with concepts of Linked Data has influenced their preference.

However, all the experts agreed that EXPRESS is a more suitable solution for providing SWS from scratch, and for that purpose they considered it is easier than both RESTdesc and OWL-S.

They also noted that there were some areas in which EXPRESS was underspecified, such as dealing with lists, sequences and complex filtering mechanisms.

An interesting insight was some of the experts' scepticism about the viability of Semantic Web services in practice.

As for the representativeness of the EXPRESS-TC in the matchmaking experiment, the experts did find it plausible that a human developer would have created something similar to the automated description, but that there was less certainty over whether this was likely. This validates the choice to use the automatically created test, as the experts viewed this as plausible (if inelegant) approach.

Chapter 8: Conclusions and Future Work

The complexity of Semantic Web services is one of the main obstacles to their adoption in the industry. This thesis demonstrated that an alternative approach is possible which does not require additional meta-models about services. This was achieved by the development and validation of EXPRESS – a SWS approach where the semantics are derived from the domain ontology and the standard interface offered by REST. The goal has been to see to what extent this approach is feasible, what compromises need to be made to make it work in practice, and to explore how successful it is in terms of reducing development effort while providing semantic richness. This chapter summarises the work done, discusses the explicit contributions made, and suggests areas for future work.

8.1 Summary

EXPRESS utilises the similarities between REST and the Semantic Web, such as resource realisation, self-describing representations, and uniform interfaces. The semantics of a service is elicited from a resource's semantic description in the domain ontology and the semantics of the uniform interface, hence eliminating the need for semantically describing services. Moreover stub-generation is a by-product of the mapping between entities in the domain ontology and resources.

Chapter 2 provided a background on middleware, the Web, Web services, REST and the Semantic Web. It highlighted the influence of middleware approaches on Web services and SWS, and the similarities between the Web, REST and the Semantic Web. Chapter 3 analysed existing SWS approaches, both in the way they describe services and in the research strategies that they have adopted. The aim of Chapter 4 was to avoid over-engineering the approach, by grounding the design decisions on the analysis of real scenarios to see if and how an approach could describe them, and what interaction requirements would need to be described. Chapter 5 discussed and demonstrated the development of EXPRESS, and provided a detailed description of it and the online deployment engine.

Chapter 6 assessed the discoverability of EXPRESSive descriptions. The results of the experiment show that EXPRESS descriptions offer very close semantic expressivity to the OWL-S ones: this is indicated by the adapted iSeM matchmaker performance, which yielded very close precision-recall measures, with an improvement in speed ranging from 4% to 38%, depending on the matchmaker variant. Chapter 7 presented the methodology and results of an Expert Review experiment, in which EXPRESS was compared to two other SWS approaches: OWL-S and RESTdesc, by providing the experts with the same scenario designed in the three approaches. The results show that experts' preferences were divided between EXPRESS and RESTdesc. Moreover, experts who tended to develop APIs for Linked Data preferred EXPRESS, while those who used existing APIs preferred RESTdesc. However, all the interviewed experts agreed that EXPRESS is a more suitable solution for providing SWS from scratch, and for that purpose they considered it easier than both RESTdesc and OWL-S.

8.2 Contributions

The work described in this thesis has made the following contributions to the field of Semantic Web services:

1. A new approach called EXPRESS, for offering Semantic RESTful Web services from domain ontologies, which eliminates service descriptions and interface vocabularies; an online demonstrator of an EXPRESS deployment engine shows how the semantic descriptions are a result of the service provision.
2. An analysis of 20 real scenarios in five Web service communities of interest, resulting in the identification of interaction requirements that guide the design of EXPRESS.
3. A Resource-Oriented Modelling approach based on UML collaboration diagrams.
4. A mapping between EXPRESSive descriptions and OWL-S descriptions.
5. The evaluation of EXPRESS in both a matchmaker experiment, which required the creation of an EXPRESSive service test collection (EXPRESS-TC) and the adaptation of a semantic matchmaker, and in an expert review, in which experts were asked to compare EXPRESS to two other SWS approaches in terms of development effort and practicality.

The research hypothesis was as follows:

Utilising the semantics in the domain ontology and REST can provide a RESTful SWS approach that (1) eliminates service ontologies/vocabularies and explicit descriptions of interfaces, and (2) generates semantic descriptions as a by-

product of its provision, and can simplify the development of SWS while preserving a similar level of semantic expressivity to existing SWS approaches.

The hypothesis led to the following research questions:

1. **Is it possible to eliminate explicit service descriptions and service ontologies/vocabularies while their semantic descriptions become a by-product of their provision?**

An online demonstrator for EXPRESS that generates and deploys Semantic RESTful Web was explained in Chapter 5. It showed how EXPRESS requires no explicit service descriptions or service vocabularies. Moreover, Chapter 5 showed how EXPRESS fulfils the six interaction requirements derived from the scenario analysis in Chapter 4, in which twenty representative scenarios from five Web service communities of interest were selected and analysed (aided by the RO models).

This shows the feasibility of EXPRESS as a SWS approach, as it provides a semantic description and at the same time fulfils the interaction requirements required by representative Web service scenarios.

However, as a result of the expert review, it is apparent that there are still some aspects that were underspecified: this included lack of explicit support for sequences, sorting, containers, sub-objects, and reverse properties. Future work (Section 8.4) discusses research activities to address these issues.

2. **Can it provide a similar level of semantic expressivity to existing approaches, and what are the trade-offs in terms of practicality?**

This question was answered by conducting a matchmaking experiment to compare the effect of the SWS description approach in EXPRESS and OWL-S. The results of the experiment show that EXPRESS descriptions offer very close semantic expressivity, in terms of discoverability, to the OWL-S ones. This is indicated by the adapted iSeM matchmaker performance which yielded very close precision-recall measures, with an improvement in speed ranging from 4% to 38%, depending on the matchmaker variant, while massively reducing the size of the service descriptions.

Moreover, the expert reviews provided feedback on the semantic quality of EXPRESS compared to the other approaches, RESTdesc and OWL-S. In general the experts found OWL-S overwhelmingly complex and this complexity outweighed the semantic richness it offered. Comparing RESTdesc and EXPRESS, most of the interviewed experts agreed that RESTdesc provided more semantic information

than EXPRESS; however, they considered the differences minimal. Experts who preferred EXPRESS stated the view that to provide any more semantic information was unnecessary.

Nevertheless, more work could be undertaken to evaluate EXPRESS's practicality. One of the directions for future work would to perform a case study were EXPRESS would be applied to develop a full application, and to study developers' feedback on both the practically and their opinions of the role of semantic descriptions; this is explained in further detail in Section 8.4.

3. Does it simplify the process of providing SWS services?

The expert review in Chapter 7 addresses this question. A scenario designed in EXPRESS was compared to the same scenario designed in two other SWS approaches, OWL-S and RESTdesc. The results show that in terms of simplicity experts' preferences were divided between EXPRESS and RESTdesc. The experts who preferred EXPRESS have expertise in publishing Linked Data, and they discussed similarities to Linked Data. This suggests that their familiarity with the concepts of Linked Data has influenced their preference. Moreover, experts who tended to develop APIs for Linked Data preferred EXPRESS, and those who used existing APIs preferred RESTdesc. However, all the experts agreed that EXPRESS is a more suitable solution for providing SWS from scratch, and for that purpose they considered it is easier than both RESTdesc and OWL-S.

8.3 Publications

The following publications were a result of this thesis.

1. Alowisheq, Areeb, Millard, David and Tiropanis, Thanassis (2011). Resource-Oriented Modelling: Describing Restful Web services Using Collaboration Diagrams. In, The 8th International Joint Conference on e-Business and Telecommunications, Seville, Spain, 18 - 21 Jul 2011.
2. Alowisheq, Areeb and Millard, David (2009) EXPRESS: EXPressing REstful Semantic Services. In, 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, Doctoral Workshop, Milan, Italy, 15 - 18 Sep 2009. , 453-456.
3. Alowisheq, Areeb, Millard, David and Tiropanis, Thanassis (2009) EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. In, 8th International Semantic Web Conference (ISWC 2009), Doctoral Consortium, Chantilly, VA, USA, 25 - 29 Oct 2009. Springer Berlin/Heidelberg, 941-948.

8.4 Future Work

The work done in this thesis explored EXPRESS's significance as a contribution to SWS research, as it successfully proposed and developed an approach for both describing and deploying RESTful SWS that reduced development effort and provided evidence of its practicality and semantic richness. This section describes the future work which is explained in the four subsections which address two main venues for exploring and improving EXPRESS: semantic richness and practicality, as shown in Figure 25.

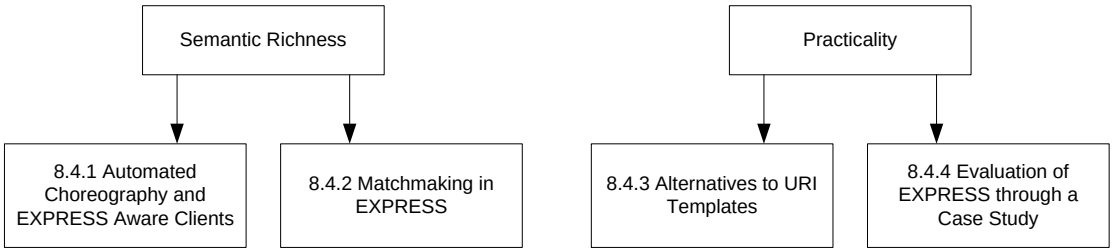


Figure 25 Future Work

8.4.1 EXPRESS Aware Clients and Automated Conversational Services

Chapter 5 of this thesis described the design and development of EXPRESS services. It explained the implementation of the server and how for conversational services the server provides the next states as URIs with allowed methods. The current implementation of EXPRESS provides deployment stubs and describes how a developer creates the code within those stubs. Chapter 5 also demonstrated the use of POSTer as a client, showing that any HTTP client can interact with those deployed stubs.

However that interaction is not automated. The ultimate aim is to design semantically intelligent clients that can interact with EXPRESSive services automatically in order to achieve goals, so the server provides the next states as URIs with allowed methods, and the client would then reason about them by converting them to rules and adding them to its knowledge base (KB).

A client's goal, specified as a rule, will be triggered, and the client will submit a request, if an appropriate URI rule is available and triggered. The next example sheds some light on the approach:

Client's goal as a rule

$$Client(?c) \wedge URI(?u) \wedge Book(?b) \wedge hasTitle(?b, "Sem\ Web") \wedge hasPrice\ (?b, ?p) \wedge Accepts(?u, "Sem\ Web") \wedge Returns(?u, ?p) \wedge canGET(?c, ?u) \wedge NoOfInputs(?u, 1) \longrightarrow GETS\ (?c, ?u)$$

The service URI and the method:

```
Link: </Book/hasPrice?hasTitle={}>; rel="GET"
```

This can be directly mapped to a rule and added to the KB:

$$Client(?c) \wedge URI(\_:URI\_1) \wedge Book(?b) \wedge hasTitle(?b, ?t) \wedge hasPrice\ (?b, ?p) \longrightarrow canGET(?c, \_:URI\_1)\ \wedge\ Accepts(\_:URI\_1, ?t)\ \wedge\ Returns(\_:URI\_1, ?p)$$

If the client's KB has sufficient assertions to trigger the URI rule, and the URI rule matches the goal rule, then the goal rule is triggered initiates action on the client. This would enable automated conversational services.

Other requirements on the client regardless of the type of interaction (conversational or atomic) would be automatically constructing messages from the client's KB, which means the client automatically constructing and responding to HTTP headers, and responding to HTTP codes.

In addition the clients would to be designed to interpret OWL restrictions as constraints, an issue discussed in Section 5.4. There are two potential ways around this issue, one is to use the syntax of OWL not its semantics (which has been the assumption in EXPRESS in Chapter 5) and programmatically deal with restrictions accordingly, rather than depending on a reasoner to trigger constraint violations. The other is to utilise approaches for local closed world reasoning (LCWR), which combines open world ontology languages like OWL with closed world assumptions. Tao *et al.* (2010) shows the use of LCWR for checking integrity constraints and several approaches have been proposed that add axioms to the ontology to enable closed world reasoning, such as the DBox approach (Seylan *et al.*, 2009) and NBox (Ren *et al.*, 2010). Moreover in the context of Semantic Web services Grimm and Hitlzer (Grimm and Hitzler, 2007) discuss the importance of LCWR for resource matching and approaches for achieving it. Further work is required to select and implement an appropriate approach for EXPRESS.

8.4.2 Matchmaking in EXPRESS

The plan for the short term is to participate in the S3 contest, by submitting both EXPRESS-TC (test collection) and iSeM EXPRESS (matchmaker) which where explained in Chapter 6. This has been encouraged by organisers of the S3 contests when contacted for enquiries about SME[2]. Participating in the experiment

would offer a platform where other researches could use the EXPRESS-TC and hence provide more feedback to the community about EXPRESS.

Moreover results of the matchmaking experiment in Chapter 6 showed that SVM variants of iSeM EXPRESS perform much worse than their OWL-S counterparts, this is due to the SVM variants being trained on an OWL-S sample of services, so further work is needed to train them on an EXPRESS sample of services.

The long term plan would be to develop a matchmaker designed for EXPRESS. In the matchmaking experiment in Chapter 6, the iSeM matchmaker, designed for OWL-S and SAWSDL was adapted to EXPRESS and used on an EXPRESSive test collection. However further research needs to be undertaken to see whether a matchmaker designed specifically for EXPRESSive services can outperform iSeM.

Section 6.2 explained the two elements in EXPRESSive descriptions that can be used for matchmaking. These are the URI of the endpoint (this maps to a resource or several resources in the domain ontology) and the HTTP method allowed on that URI. For the endpoint URI there are three main ways to utilise it for matchmaking. This thesis chose to explore one of them, which was extracting input and output concepts from filtering resources URI. The two other aspects provide different methods for matchmaking (explained in Section 6.2), these are:

1. graph matching (using graphs that correspond to the URI templates),
2. and monolithic DL matching, where URIs refer to classes.

An interesting venue to explore is which of these methods is more effective, and whether a hybrid approach that combines them would improve the performance. Moreover an adaptive method could be designed for the hybrid approach, using SVM that is trained to learn the appropriate weights for the combined methods.

Another method of matchmaking in EXPRESS, that could be explored, is to make use of ontology matching approaches such as (Doan *et al.*, 2004), where a client would specify the required concepts in an ontology and EXPRESSive ontologies would be retrieved and compared.

8.4.3 Alternatives to URI Templates

One of the potential criticisms of EXPRESS is dependency on URI templates. The argument against using URI templates is that introduces coupling between the server and the client, because from a purist viewpoint the URI should be opaque, and yet the client could infer information from the URI structure. It is interesting

that despite this argument the reviews from the interviewed experts seem to prefer URI templates, and view them as a practical solution.

Nevertheless, EXPRESS could be designed differently. There are two other possible alternatives that could be further investigated and developed: One is to have a mechanism to represent the resource type in the headers of exchanged messages, as an extra attribute in the Link elements. Another alternative that would provide more flexibility, would be to define a machine-readable media type specification. For a start, it would have the following features:

1. The type of a resource linked to an ontology or vocabulary
2. Effects of a certain method, or the value of rel attribute, expressed as a rule or SPARQL CONSTRUCT.

Media type specifications are written as human readable documents, developers read them then design the servers or clients accordingly. An interesting area of research would be studying the feasibility of machine-readable media type specifications. An excellent explanation of designing media-types is by Amundsen (2011a).

8.4.4 Evaluation of EXPRESS through a Case Study

The work described in this thesis uses exemplars, demonstrators and expert reviews to explore the practical issues around EXPRESS, as Section 3.5 pointed out this is in line with existing work (and in fact using expert reviews goes beyond the efforts made with most proposed approaches). However, in Section 3.5, Table 2 from Shaw (2002) referred to other validation techniques, such as experience, which can be achieved using longitudinal case studies, typically applied elsewhere in the Software Engineering world, where developers use and then reflect on a given approach as part of a longer term project.

For EXPRESS this would mean a study where developers would use EXPRESS in a real project. This would provide comprehensive qualitative feedback about the hands-on application of EXPRESS.

Moreover, applying EXPRESS to a real application would ground solutions to underspecified issues in EXPRESS, such as in providing sequences, containers, sorting sub-objects, reverse properties and error handling. It would also enable extra features suggested by the experts to be addressed, including aspects such as advanced filtering, content negotiation, trust, and versioning.

An area for future investigation is the association between the database design and the interface design, which was suggested by one of the experts, in which the ontology is derived from the database schema. This provides further opportunity for automation, as the SPARQL mappings (described in Section 5.2.2 and Appendix C) are not only specifications, but could become the actual implementation of the Web service.

8.5 Final Conclusions

This research aimed to study the potential of pragmatic solutions that can lower the entry barrier for Semantic Web services, and to develop an effective Semantic Web service approach that offers rich discovery by harnessing the strengths of semantic technologies while being accessible to every day Web developers.

The underlying assumption behind the approach is that an implicit, intuitive meta-model would be more likely to be adopted than an explicit, complex one. Therefore, in designing EXPRESS, interaction service descriptions and service ontologies or vocabularies were deliberately avoided, the aim was to investigate to what extent does using only the domain ontology and REST provide a viable substitute.

The demonstrator, evaluation and expert review that have been conducted show that compared to OWL-S, EXPRESS has succeeded in massively reducing the size of semantic descriptions, and improving the speed of semantic matchmaking, while providing similar accuracy. However, EXPRESS requires a different conceptualisation of the problem, leading the interviewed experts to voice mixed opinions about its practicality, although all of them appreciated its simplicity for building Web services from scratch. This is the area in which EXPRESS seems to hold the most promise, as a way of creating SWS from the ground up, driven by ontology design, and supported where possible by automated deployment. It is an approach that may have a lot of resonance with the Linked Data community who prioritise practical solutions, and are experienced with developing around existing Web standards.

It seems that, although much research has been done in SWS, the issue of their practicality in real world applications still remains in question. Above all the work undertaken with EXPRESS shows that we need to start from real problems and conduct a careful analysis of whether these are practical solutions, through engagement with practitioners, and not only Semantic Web enthusiasts.

References

Adida, B., Birbeck, M., McCarron, S. & Pemberton, S. (2008). RDFa in XHTML: Syntax and Processing. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2013/REC-rdfa-core-20130822/ [Accessed 14/10/2010].

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A. P. & Verma, K. (2005). Web Service Semantics - WSDL-S. *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/ [Accessed 14/9/2011].

Alarcon, R. & Wilde, E. (2010) Linking Data from RESTful Services. In*: Proceedings of the WWW2010 Workshop on Linked Data on the Web (LDOW 2010)*, April 27, Raleigh, North Carolina. volume 628.

Alexander, K., Cyganiak, R., Hausenblas, M. & Zhao, J. (2009) Describing Linked Datasets, On the Design and Usage of voiD, the "Vocabulary of Interlinked Datasets". In*: Proceedings of the WWW2009 Workshop Linked Data on the Web (LDOW2009)* April 20, Madrid, Spain. CEUR Workshop Proceedings, CEUR-WS.org, volume 583.

Allamaraju, S. (2010). *RESTful Web Services Cookbook*, O'Reilly.

Allemang, D. & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL 2nd ed.*, Morgan Kaufmann.

Alonso, G., Casati, F., Kuno, H. & Machiraju, V. (2004). *Web services,* Berlin Heidelberg, Springer.

Alowisheq, A. & Millard, D. E. (2009) EXPRESS: EXPressing REstful Semantic Services. In*: Proceedings of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology - Workshops | 2nd Doctoral Workshop (DOCW 2009)*, September 15-18, Milan, Italy. IEEE, pages 453-456.

Alowisheq, A., Millard, D. E. & Tiropanis, T. (2009) EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. In*: Proceedings of the 8th International Semantic Web Conference (ISWC 2009) | Doctoral Consortium* October 25-29, Chantilly, VA, USA. Lecture Notes in Computer Science, Springer, volume 5823, pages 941-948.

Alowisheq, A., Millard, D. E. & Tiropanis, T. (2011) Resource-Oriented Modelling: Describing Restful Web Services Using Collaboration Diagrams. In*: Proceedings of the 8th International Joint Conference on e-Business and Telecommunications (ICE-B 2011)*, July 18-21, Seville, Spain. SciTePress, pages 113-118.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C. K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., Rijn, D. v. d., Yendluri, P. & Yiu, A. (2007). Web Services Business Process Execution Language Version (WSBPEL) *OASIS Standard, OASIS Committee* [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel [Accessed 21/07/2013].

Amundsen, M. (2011a). *Building Hypermedia APIs With HTML 5 and Node*, O'Reilly Media, USA

Amundsen, M. (2011b). Hypermedia Types. *In:* WILDE, E. & PAUTASSO, C. (eds.) *REST: From Research to Practice.* New York: Springer.

Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D. L., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. & Sycara, K. (2001) DAML-S: Semantic Markup for Web Services. In*: Proceedings of the International Semantic*

Web Working Symposium (SWWS), July 30 - August 1, Stanford University, California, USA. pages 411-430.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Randy Katz, Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. & Zaharia, M. (2009). *Above the Clouds: A Berkeley View of Cloud Computing*. UC Berkeley Reliable Adaptive Distributed Systems Laboratory. UCB/EECS-2009-28.

Austin, J., Davis, R., Fletcher, M., Jackson, T., Jessop, M., Liang, B. & Pasley, A. (2005). DAME: Searching Large Data Sets within a Grid-enabled Engineering Application. *Proceedings of the IEEE,* 93(3):496-509.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P. F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, New York.

Bachlechner, D. & Fink, K. (2008). Semantic Web Service Research: Current Challenges and Proximate Achievements. *International Journal of Computer Science & Applications (IJCSA),* 5(3b):117-140.

Battle, R. & Benson, E. (2008). Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Journal of Web Semantics,* 6(1):61-69.

Battle, S., Bernstein, A., Boley, H., Gruninger, M., Hull, R., Kifer, M., Martin, D., McLlraith, S., McGuinness, D., Su, J. & Tabet, S. (2005). Semantic Web Services Framework (SWSF) Overview. *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2005/SUBM-SWSF-20050909/ [Accessed 9/10/2012].

Beckett, D. & McBride, B. (eds.) (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ [Accessed 17/7/2012].

Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y., Januszewski, K., Lee, S., McKee, B. & Munter, J. (2002). The Universal Description, Discovery and Integration (UDDI) Specification. *Technical Report, OASIS Committee* [Online]. Available: https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm [Accessed 22/06/2012].

Berners-Lee, T. (1991). *The Original HTTP as defined in 1991* [Online]. Available: http://www.w3.org/Protocols/HTTP/AsImplemented.html [Accessed 13/7/2013].

Berners-Lee, T. (1994). RFC 1630: Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web. *IETF* [Online]. Available: http://www.ietf.org/rfc/rfc1630.txt [Accessed 22/6/2012].

Berners-Lee, T. (1998). *Semantic Web Roadmap* [Online]. World Wide Web Consortium (W3C). Available: http://www.w3.org/DesignIssues/Semantic.html [Accessed 3/6/2013].

Berners-Lee, T. (2006). *Linked Data - Design Issues* [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html [Accessed 3/7/2013].

Berners-Lee, T., Cailliau, R., Groff, J.-F. & Pollermann, B. (1992). World-Wide Web: The Information Universe. *Internet Research,* 2(1):52-58.

Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A. & Sheets, D. (2006) Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In*: Proceedings of the 3rd International Semantic Web User Interaction Workshop*, November 6, Athens, Georgia, USA. SemanticWeb.org.

Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y. & Hendler, J. (2008). N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming,* 8(3):249-269.

Berners-Lee, T., Fielding, R. & Masinter, L. (2005). RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. *IETF* [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The Semantic Web. *Scientific American,* 284(5):28-37.

Bernstein, P. A. (1996). Middleware: a Model for Distributed System Services. *Communications of the ACM,* 39(2):86-98.

Biornstad, B. & Pautasso, C. (2009) Let It Flow: Building Mashups with Data Processing Pipelines. In*: Proceedings of Service-Oriented Computing - ICSOC 2007 Workshops*, September 17, Vienna, Austria. Lecture Notes in Computer Science, Springer, volume 4907, pages 15-28.

Birrell, A. D. & Nelson, B. J. (1984). Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems,* 2(1):39-59.

Bizer, C., Heath, T. & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems,* 5(3):1-22.

Blum, A. & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence,* 90(1):281-300.

Booch, G., Rumbaugh, J. & Jacobson, I. (1999). *Unified Modeling Language User Guide,* Reading, MA, Addison-Wesley

Booth, D., Haas, H., McCabe, F., Necomer, E., Champion, M., Ferris, C. & Orchard, D. (2004). Web Services Architecture. *W3C Working Group Note, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

Bournez, C. (2005). Team Comment on Web Service Modeling Ontology (WSMO) Submission. *World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2005/06/Comment.

Box, D. (2001). *A Brief History of SOAP* [Online]. Available: http://www.xml.com/lpt/a/759 [Accessed 13/7/2013].

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. & Winer, D. (2000). Simple Object Access Protocol (SOAP) 1.1. *W3C Note, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

Bray, T., Paoli, J. & Sperberg-McQueen, C. M. (1998). Extensible Markup Language (XML) 1.0. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/1998/REC-xml-19980210 [Accessed 14/10/2010].

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. & Yergeau, F. (2008). Extensible Markup Language (XML) 1.0. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2008/REC-xml-20081126/ [Accessed 13/12/2012].

Brickley, D. & Guha, R. V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ [Accessed 10/10/2012].

Brodie, M. L. (2000). The B2B e-commerce Revolution: Convergence, Chaos, and Holistic Computing. *Information Systems Engineering,* 15-36.

Bruijn, J. d., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Lausen, H., Oren, E., Polleres, A., Roman, D., Scicluna, J. & Stollberg, M. (2005a). Web Service Modeling Ontology (WSMO). *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/ [Accessed 12/12/2010].

Bruijn, J. d., Fensel, D., Keller, U., Kifer, M., Lausen, H., Krummenacher, R., Polleres, A. & Predoiu, L. (2005b). Web Service Modeling Language (WSML). *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2005/SUBM-WSML-20050603/ [Accessed 12/12/2010].

Bush, V. (1945). As we may think. *Atlantic Monthly,* 176(1):101-108.

Cabral, L., Domingue, J., Motta, E., Payne, T. R. & Hakimpour, F. (2004) Approaches to Semantic Web Services: An Overview and Comparison. In*: Proceedings of 1st European Semantic Web Symposium,The Semantic Web: Research*

and Applications (ESWS 2004), May 10-12, Heraklion, Crete, Greece. Lecture Notes in Computer Science, Springer, volume 3053, pages 225-239.

Cabral, L., Ning, L. & Kopecký, J. (2012) Building the WSMO-Lite Test Collection on the SEALS Platform. In*: Proceedings of the ESWC2012 2nd International Workshop on Evaluation of Semantic Technologies (IWEST 2012)*, May 28, Heraklion, Greece. CEUR Workshop Proceedings, CEUR-WS.org, volume 843.

Calladine, J. (2004). Giving Legs to the Legacy - Web Services Integration within the Enterprise. *BT Technology Journal,* 22(1):87-98.

Champin, P.-A. (2013) RDF-REST: A Unifying Framework for Web APIs and Linked Data. In*: Proceedings of the ESWC2013 1st Workshop on Services and Applications over Linked APIs and Data (SALAD 2013)*, May 26, Montpellier, France. CEUR Workshop Proceedings, CEUR-WS.org, volume 1056.

Cheng, G. & Qu, Y. Z. (2009). Searching Linked Objects with Falcons: Approach, Implementation and Evaluation. *International Journal on Semantic Web and Information Systems,* 5(3):49-70.

Chien, A., Calder, B., Elbert, S. & Bhatia, K. (2003). Entropia: Architecture and Performance of an Enterprise Desktop Grid System. *Journal of Parallel and Distributed Computing,* 63(5):597-610.

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. *W3C Note, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 [Accessed 10/10/2012].

City University. (2008). *Going For Gold, Introducing SOA at City University London.*

Connolly, D. (1997). *A draft of the editorial of the Mar/Apr 1997 issue of Web Apps Magazine* [Online]. Available: http://www.w3.org/People/Connolly/9703-web-apps-essay.html [Accessed 12/12/2013].

Connolly, D. & Hickson, I. (1999). An Entity Header for Linked Resources. *Draft, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Protocols/9707-link-header.html.

Decker, G., Lüders, A., Overdick, H., Schlichting, K. & Weske, M. (2008) RESTful Petri Net Execution. In*: Proceedings of the 5th International Workshop on Web Services and Formal Methods*, September 4-5, Milan, Italy. Lecture Notes in Computer Science, Springer, volume 5387.

Decker, G. & Weske, M. (2007) Behavioral Consistency for B2B Process Integration. In*: Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, June 11-15, Trondheim, Norway. Lecture Notes in Computer Science, Springer, volume 4495, pages 81-95.

DeLeon, A. & Dumontier, M. (2008) Publishing OWL Ontologies with Presto. In*: Proceedings of the OWLED 2008 OWL: Experiences and Directions*, April 1-2, Washington, DC. CEUR Workshop Proceedings, CEUR-WS.org, volume 496.

Doan, A., Madhavan, J., A Domingos, P. & Halevy, A. (2004). Ontology Matching: A Machine Learning Approach. *In:* STAAB, S. & STUDER, R. (eds.) *Handbook on Ontologies.* Springer Berlin Heidelberg.

Dong, H., Hussain, F. K. & Chang, E. (2012). Semantic Web Service matchmakers: state of the art and challenges. *Concurrency and Computation: Practice and Experience,* 25(7):961–988.

Donnelly, P. (2010). *Yahoo Finance Stock Quote Watch List Feed* [Online]. Yahoo. Available: http://pipes.yahoo.com/31337/watchlist [Accessed 26/02/2010].

Emmerich, W., Aoyama, M. & Sventek, J. (2007). The Impact of Research on Middleware Technology. *ACM SIGOPS Operating Systems Review,* 41(1):89-112.

Engelbart, D. C. (1963). Conceptual Framework for the Augmentation of Man's Intellect. *In:* HOWERTON, P. W. (ed.) *Vistas in Information Handling.* Washington, D.C.: Spartan Books.

Ennals, R., Brewer, E., Garofalakis, M., Shadle, M. & Gandhi, P. (2007). Intel Mash Maker: Join the Web. *SIGMOD Record,* 36(4):27-33.

Ennals, R. J. & Garofalakis, M. N. (2007) MashMaker: Mashups for the Masses. In*: Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 12-14, Beijing, China. 1247626: ACM, pages 1116-1118.

Erl, T. (2008). *SOA: Principles of Service Design,* Upper Saddle River, Prentice Hall

Falou, M. E., Bouzid, M., Mouaddib, A.-I. & Vidal, T. (2009) Automated Web Service Composition: A Decentralised Multi-agent Approach. In*: Proceedings of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology*, September 15-18, Milan, Italy. IEEE, pages 387-394.

Farnaghi, M. & Mansourian, A. (2013). Automatic Composition of WSMO based Geospatial Semantic Web Services Using Artificial Intelligence Planning. *Journal of Spatial Science,* 58(2):235-250.

Farrell, J. & Lausen, H. (2007). Semantic Annotations for WSDL and XML Schema. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2007/REC-sawsdl-20070828/ [Accessed 14/10/2010].

Fensel, D. (2004) Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In*: Proceedings of the IFIP International Conference Intelligence in Communication Systems (INTELLCOMM 2004)*, November 23-26, Bangkok, Thailand. Lecture Notes in Computer Science, Springer, volume 3283, pages 43-53.

Fensel, D., Krummenacher, R., Shafiq, O., Kühn, E., Riemer, J., Ding, Y. & Draxler, B. (2007). TSC – Triple Space Computing. *e & i Elektrotechnik und Informationstechnik,* 124(1):31–38.

Fensel, D. & van Harmelen, F. (2007). Unifying Reasoning and Search to Web Scale. *IEEE Internet Computing,* 11(2):94-96.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H. & Berners-Lee, T. (1997). RFC 2068: Hypertext Transfer Protocol–HTTP/1.1 *IETF* [Online]. Available: http://www.ietf.org/rfc/rfc2068.txt [Accessed 22/6/2012].

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999). RFC 2616: Hypertext Transfer Protocol–HTTP/1.1 *IETF* [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt [Accessed 22/6/2012].

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral Dissertation, University of California.

Fielding, R. T. (2007). A Little REST and Relaxation. Presented at: *The International Conference on Java Technology (JAZOON07)*, June 24-28 Zurich, Switzerland.

Fielding, R. T. (2008a). *On Software Architecture* [Online]. Untagled. Available: http://roy.gbiv.com/untangled/2008/on-software-architecture [Accessed 23/22/2010].

Fielding, R. T. (2008b). *REST APIs Must Be Hypertext-driven* [Online]. Untangled. Available: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven [Accessed 23/22/2010].

Fielding, R. T., Berners-Lee, T. & Frystyk, H. (1996). RFC 1945: Hypertext Transfer Protocol-HTTP/1.0. *IETF* [Online]. Available: https://www.ietf.org/rfc/rfc1945.txt [Accessed 22/6/2012].

Fikes, R. E. & Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence,* 2(3):189–208.

Filho, O. F. F. & Ferreira, M. A. G. V. (2009) Semantic Web Services: A RESTful Approach. In*: Proceedings of the IADIS International Conference on WWW/Internet 2009*, November 19 - 22, Rome, Italy. IADIS Press.

Foster, I., Kesselman, C., Nick, J. M. & Tuecke, S. (2002). Grid Services for Distributed System Integration. *Computer,* 35(6):37-46.

Foster, I., Parastatidis, S., Watson, P. & McKeown, M. (2008). How Do I Model State? Let Me Count the Ways. *Communications of the ACM,* 51(9):34-41.

Francisco, D. d., Nixon, L. & Valle, G. T. d. (2008) Towards a Multimedia Content Marketplace Implementation Based on Triplespaces. In*: Proceedings of the 7th International Semantic Web Conference (ISWC 2008)*, October 26-30,

Karlsruhe, Germany. Lecture Notes in Computer Science, Springer, volume 5318, pages 875–888.

Fremantle, P., Weerawarana, S. & Khalaf, R. (2002). Enterprise Services. *Communications of the ACM,* 45(10):77-82.

Frey, J., De Roure, D., Taylor, K., Essex, J., Mills, H. & Zaluska, E. (2006) CombeChem: A Case Study in Provenance and Annotation Using the Semantic Web. In*: Proceedings of the International Provenance and Annotation Workshop (IPAW 2006)*, May 3-5, Chicago, IL, USA. Lecture Notes in Computer Science, Springer, volume 4145, pages 270-277.

Frey, J. G., Bradley, M., Essex, J., Hursthouse, M. B., Lewis, S. M., Luck, M., Moreau, L., De Roure, D., M., S. & Welsh, A. (2003). Combinatorial Chemistry and the Grid. *In:* F.BERMAN, G.FOX & T.HEY (eds.) *Grid Computing: Making the Global Infrastructure a Reality.* John Wiley & Sons Ltd.

Gearon, P., Passant, A. & Polleres, A. (2013). SPARQL 1.1 Update. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-update-20130321/ [Accessed 10/12/2013].

Gessler, D. D., Schiltz, G. S., May, G. D., Avraham, S., Town, C. D., Grant, D. & Nelson, R. T. (2009). SSWAP: A Simple Semantic Web Architecture and Protocol for Semantic Web Services. *BMC Bioinformatics [Online],* 10(1).

Glaser, H., Jaffri, A. & Millard, I. (2009) Managing Co-reference on the Semantic Web. In*: Proceedings of the WWW2009 Workshop Linked Data on the Web (LDOW2009)* April 20, Madrid, Spain.: CEUR Workshop Proceedings, CEUR-WS.org, volume 583.

Gregorio, J. & de hOra, B. (eds.) (2007). RFC:5023 The Atom Publishing Protocol. *IETF* [Online]. Available: http://www.ietf.org/rfc/rfc5023.txt [Accessed 23/6/2012].

Grimm, S. (2007). Discovery - Identifying Relevant Services *In:* STUDER, R., GRIMM, S. & ABECKER, A. (eds.) *Semantic Web Services.* Springer.

Grimm, S. & Hitzler, P. (2007). Semantic Matchmaking of Web Resources with Local Closed-World Reasoning. *International Journal of Electronic Commerce,* 12(2):89-126.

Gudgin, M., Hadley, M. & Rogers, T. (2006). Web Services Addressing 1.0 - Core. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/ [Accessed 13/12/2012].

Gullapalli, S., Dyke, S., Hubbard, P., Marcusiu, D., Pearlman, L. & Severance, C. (2004) Showcasing the Features and Capabilities of NEESgrid: A Grid-based System for the Earthquake Engineering Domain. In*: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13 2004)*, June 4-6, Honolulu, Hawaii USA. IEEE, pages 268-269.

Hadley, M. (2009). Web Application Description Language (WADL). *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2009/SUBM-wadl-20090831/ [Accessed 31/8/2012].

Halpin, H. & Davis, I. (2007). Gleaning Resource Descriptions from Dialects of Languages (GRDDL). *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2007/REC-grddl-20070911/ [Accessed 11/9/2012].

Harris, S. & Seaborne, A. (2013). SPARQL 1.1 Query Language, Section 9: Property Paths. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/sparql11-query/#propertypaths [Accessed 15/9/2014].

Heath, T. & Bizer, C. (2011). Linked Data: Evolving the Web into a Global Data Space. *Synthesis Lectures on the Semantic web: Theory and Technology,* 1(1):1-136. Morgan & Claypool.

Hench, G., Simperl, E., Wahler, A. & Fensel, D. (2008) A Conceptual Roadmap for Scalable Semantic Computing. In*: Proceedings of the 2th IEEE International*

Conference on Semantic Computing (ICSC 2008), August 4-7, Santa Clara, CA, USA. IEEE, pages 562-568.

Henning, M. (2006). The Rise and Fall of CORBA. *Queue,* 4(5):28-34.

Hernandez, A. G. & Garcia, M. N. M. (2010) A Formal Definition of RESTful Semantic Web Services. In*: Proceedings of the First International Workshop on RESTful Design (WS-REST 2010)*, Raleigh, North Carolina, USA. ACM, pages 39-45.

Herrick, D. R. (2009) Google this!: Using Google Apps for Collaboration and Productivity. In*: Proceedings of the ACM SIGUCCS Fall Conference on User Services 2009*, October 11-14, St. Louis, Missouri, USA. 1629513: ACM, pages 55-64.

Hitzler, P., Krötzsch, M., Parisa, B., Patel-Schneider, P. F. & Rudolph, S. (2012). OWL 2 Web Ontology Language Primer *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/ [Accessed 2/9/2013].

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, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/ [Accessed 21/2/2011].

International Organization for Standardization (1986). ISO 8879:1986 Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML).

International Organization for Standardization (2004). ISO 18629:2004 Industrial automation systems and integration -- Process specification language (PSL).

Jackson, T., Austin, J., Fletcher, M. & Jessop, M. (2003) Delivering a Grid-enabled Distributed Aircraft Maintenance Environment (DAME). In*: Proceedings of the UK e-Science All Hands Meeting*, September 2-4, Nottingham, UK. pages 420-427.

Jackson, T., Austin, J., Fletcher, M., Jessop, M., Liang, B., Pasley, A., Ong, M., Ren, X., Allan, G., Kadirkamanathan, V., Thompson, H. & Fleming, P. (2005) Distributed Health Monitoring for Aero-engines on the GRID: DAME. In*: Proceedings of the IEEE Aerospace Conference*, March 5-12, Big Sky, Montana. IEEE pages 3738-3747.

Jackson, T., Jessop, M., Fletcher, M. & Austin, J. (2006) A Virtual Organisation Deployed on a Service Orientated Architecture for Distributed Data Mining Applications. In*: Proceedings of the IFIP TC2/ WG 2.5 Working Conference on Grid-Based Problem Solving Environments: Implications for Development and Deployment of Numerical Software*, July 17–21, Prescott, Arizona, USA. Lecture Notes in Computer Science, Springer, volume 239, pages 155-170.

Jhingran, A. (2006) Enterprise Information Mashups: Integrating Information, Simply. In*: Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006)*, September 12-15, Seoul, Korea. 1164128: ACM, pages 3-4.

Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y. & Barreto, C. (2005). Web Services Choreography Description Language Version 1.0. *W3C Candidate Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/ [Accessed 12/12/2013].

Kjernsmo, K. (2012). The Necessity of Hypermedia RDF and an Approach to Achieve it. Presented at: *The 1st Linked APIs Workshop (LAPIS) at ESWC2012*, May 27-31 Heraklon, Crete.

Klein, M., Konig-Ries, B. & Mussig, M. (2005). What is Needed for Semantic Service Descriptions? A Proposal for Suitable Language Constructs. *International Journal of Web and Grid Services,* 1(3/4):328–364.

Klems, M., Nimis, J. & Tai, S. (2008) Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing. In*: Proceedings of the (WEB2008) 7th Workshop on E-Business*, December 13, Paris, France.

Lecture Notes in Business Information Processing, Springer, volume 22, pages 110-123.

Klensin, J. (ed.) (2001). RFC 2821: Simple Mail Transfer Protocol. *IETF* [Online]. Available: https://www.ietf.org/rfc/rfc2821.txt [Accessed 21/7/2014].

Klusch, M. (2008a). Semantic Web Service Coordination. *In:* SCHUMACHER, M., HELIN, H. & SCHULDT, H. (eds.) *CASCOM: Intelligent Service Coordination in the Semantic Web.* Birkhäuser Verlag.

Klusch, M. (2008b). Semantic Web Service Description. *In:* SCHUMACHER, M., HELIN, H. & SCHULDT, H. (eds.) *CASCOM: Intelligent Service Coordination in the Semantic Web.* Birkhäuser Verlag.

Klusch, M., Dudev, M., Mišutka, J., Kapahnke, P. & Vasileski, M. (2010a). *Semantic Web Service Matchmaker Evaluation Environment User Manual*. DFKI.

Klusch, M. & Kapahnke, P. (2010a) iSem: Approximated Reasoning for Adaptive Hybrid Selection of Semantic Services. In*: Proceedings of the 4th IEEE International Conference on Semantic Computing (ICSC 2010)*, Pittsburgh, PA, USA. IEEE, pages 184-191.

Klusch, M. & Kapahnke, P. (2010b). *OWL-S Service Retrieval Test Collection version 4.0 (OWLS-TC)* [Online]. Available: http://projects.semwebcentral.org/projects/owls-tc/ [Accessed 22/2/2012].

Klusch, M. & Kapahnke, P. (2010c). *SAWSDL Service Retrieval Test Collection version 3.0 (SAWSDL-TC)* [Online]. Available: http://projects.semwebcentral.org/projects/sawsdl-tc/ [Accessed 22/2/2012].

Klusch, M., Khalid, M. A., Kapahnke, P., Fries, B. & Vasileski, M. (2010b). *OWL-S Service Retrieval Test Collection User Manual*. DFKI.

Kopecky, J. (2012). *Web Service Automation Supported by Lightweight Semantic Annotations.* Doctoral Dissertation, Semantic Technology Institute Innsbruck.

Kopecky, J., Gomadam, K. & Vitvar, T. (2008) hRESTS: An HTML Microformat for Describing RESTful Web Services. In*: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology*, December 9-12, Sydney, Australia. IEEE, pages 619-625.

Kopecky, J., Pedrinaci, C. & Duke, A. (2011) RESTful Write-oriented API for Hyperdata in Custom RDF Knowledge Bases. In*: Proceedings of the 7th IEEE International Conference on Next Generation Web Services Practices (NWeSP2011)*, October 19-21, Salamanca, Spain. IEEE, pages 199 - 204.

Kozierok, C. (2005). *The TCP/IP-Guide: A Comprehensive, Illustrated Internet Protocols Reference*, No Starch Press.

Kreger, H. (2003). Fulfilling the Web Services Promise. *Communications of the ACM,* 46(6):29-34.

Krummenacher, R., Norton, B. & Marte, A. (2010) Towards Linked Open Services and Processes. In*: Proceedings of the 3rd Future Internet Symposium (FIS2010)*, September 20-22, Berlin, Germany. Lecture Notes in Computer Science, Springer, volume 6369, pages 68-77.

Küster, U., König-Ries, B., Stern, M. & Klein, M. (2007) DIANE: an Integrated Approach to Automated Service Discovery, Matchmaking and Composition. In*: Proceedings of the 16th International World Wide Web Conference (WWW2007)*, 8-12 May, Banff, Alberta, Canada. ACM, pages 1033-1042.

Lassila, O. & Swick, R. R. (1997). Resource Description Framework (RDF) Model and Syntax. *W3C Working Draft, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/WD-rdf-syntax-971002/ [Accessed 10/10/2012].

Lathem, J., Gomadam, K. & Sheth, A. P. (2007) SA-REST and (S)mashups: Adding Semantics to RESTful Services. In*: Proceedings of the First IEEE International Conference on Semantic Computing (ICSC2007)*, September 17-19, Irvine, California. IEEE, pages 469-476.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McLlraith, S., Narayanan, S., Paulocci, M., Parsia, B., Payne, T. R., Sirin, E., Srinivasan, N. & Sycara, K. (2004). OWL-S: Semantic Markup for Web Services. *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ [Accessed 10/10/2012].

McGuinness, D. & Harmelen, F. v. (2004). OWL Web Ontology Language Overview. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/ [Accessed 12/2/2013].

McIlraith, S. A., Son, T. C. & Zeng, H. L. (2001). Semantic Web services. *IEEE Intelligent Systems & Their Applications,* 16(2):46-53.

Moreau, J.-J., Chinnici, R., Ryman, A. & Weerawarana, S. (2007). Web Services Description Language (WSDL) Version 2.0 Part 1: Core language. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2007/REC-wsdl20-20070626/ [Accessed 14/2/2012].

Nadalin, A., Kaler, C., Monzillo, R. & Hallam-Baker, P. (eds.) (2006). Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). *OASIS Standard* [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss [Accessed 10/12/2013].

Nally, M., Speicher, S., Arwe, J. & Hors, A. L. (2012a). Linked Data Basic Profile 1.0. *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2012/SUBM-ldbp-20120326/ [Accessed 12/2/2013].

Nally, M., Speicher, S., Arwe, J. & Hors, A. L. (2012b). Linked Data Basic Profile 1.0 - Use Cases and Requirements. *W3C Member Submission, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/Submission/2012/SUBM-ldbpucr-20120326/ [Accessed 12/2/2013].

Natis, Y. V. (2003). Service-Oriented Architecture Scenario. *AV-19-6751, Gartner Research* [Online]. Available: http://www.gartner.com/resources/114300/114358/114358.pdf [Accessed 9/10/2012].

Nelson, T. H. (1980) Replacing the Printed Word: A Complete Literary System. In*: Proceedings of IFIP Congress 80*, October 14-17, North Holland. 1013–1023.

Nottingham, M. (2010). RFC 5988: Web Linking. *IETF* [Online]. Available: http://tools.ietf.org/html/rfc5988 [Accessed 22/6/2012].

Nottingham, M. & Sayre, R. (2005). The Atom Syndication Format. *RFC 4287*.

O'reilly, T. (2005). *What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software* [Online]. Available: http://oreilly.com/web2/archive/what-is-web-20.html [Accessed 21/12/2011].

Object Management Group (1995). The Common Object Request Broker: Architecture and Specification.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A. & Li, P. (2004). Taverna: a Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics,* 20(17):3045-3054.

Paolucci, M., Kawamura, T., Payne, T. R. & Sycara, K. P. (2002) Semantic Matching of Web Services Capabilities. In*: Proceedings of the 1st International Semantic Web Conference (ISWC 2002)*, Chia, Sardinia, Italy. Lecture Notes in Computer Science, Springer, volume 2342, pages 333-347.

Pautasso, C. (2009). RESTful Web Service Composition with BPEL for REST. *Data & Knowledge Engineering,* 68(9):851-866.

Pautasso, C., Zimmermann, O. & Leymann, F. (2008) RESTful Web Services vs. "Big"' Web Services: Making the Right Architectural Decision. In*: Proceedings of*

WWW 2008 the 17th International Conference on World Wide Web, April 21-25, Beijing, China. ACM, pages 805–814.

Pearlman, L., Kesselman, C., Gullapalli, S., Spencer, B. F., Futrelle, J., Ricker, K., Foster, I., Hubbard, P. & Severance, C. (2004) Distributed Hybrid Earthquake Engineering Experiments: Experiences with a Ground-shaking Grid Application. In*: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, June 4-6, Honolulu, Hawaii, USA. IEEE, pages 14-23.

Pedrinaci, C., Domingue, J. & Krummenacher, R. (2010a) Services and the Web of Data: An Unexploited Symbiosis. In*: Proceedings of the AAAI Spring Symposium 2010 Workshop Linked Data Meets Artificial Intelligence*, March 22–24, Palo Alto, California, USA. pages 99-100.

Pedrinaci, C., Liu, D., Maleshkova,, M., L., D.,, Kopecky, J. & Domingue, J. (2010b) iServe: a Linked Services Publishing Platform. In*: Proceedings of the ESWC2010 Workshop on Ontology Repositories and Editors for the Semantic Web*, May 30th - June 2nd, Heraklion, Greece. CEUR Workshop Proceedings, CEUR-WS.org, volume 596.

Petrie, C., Margaria, T., Lausen, H. & Zaremba, M. (2009). *Semantic Web Services Challenge: Results from the First Year*, Springer.

Preist, C., Esplugas-Cuadrado, J., Battle, S. A., Grimm, S. & Williams, S. K. (2005) Automated Business-to-Business Integration of a Logistics Supply Chain Using Semantic Web Services Technology. In*: Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland. Lecture Notes in Computer Science, Springer, volume 3729, pages 987-1001.

Prescod, P. (2002) Roots of the REST/SOAP Debate. In*: Proceeding of the Extreme Markup Languages®*, August 4-9, Montréal, Quebec, Canada.

Prud'Hommeaux, E. & Seaborne, A. (2008). SPARQL Query Language for RDF. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ [Accessed 12/2/2012].

Raggett, D., Le Hors, A. & Jacobs, I. (1999). HTML 4.01 Specification. *W3C Recommendation, World Wide Web Consortium (W3C)* [Online]. [Accessed 10/1/2012].

Reisig, W. (1985). *Petri nets: An Introduction,* New York, Springer-Verlag.

Ren, Y., Pan, J. Z. & Zhao, Y. (2010). Closed World Reasoning for OWL2 with NBox. *Tsinghua Science & Technology,* 15(6):692-701.

Richardson, L. (2008). Act Three: The Maturity Heuristic. *Slides for QCon 2008 Talk* [Online]. Available: http://www.crummy.com/writing/speaking/2008-QCon/act3.html [Accessed 13/7/2013].

Richardson, L. & Ruby, S. (2007). *RESTful Web Services*, O'Reilly Media.

Riemer, J., Martin-Recuerda, F., Ding, Y., Murth, M., Sapkota, B., Krummenacher, R., Shafiq, O., Fensel, D. & Kühn, E. (2006) Triple Space Computing: Adding Semantics to Space-Based Computing. In*: Proceedings of the 1st Asian Semantic Web Conference (ASWC 2006)*, September 3-7, Beijing, China. Lecture Notes in Computer Science, Springer, volume 4185, pages 300–306.

Sauermann, L., Cyganiak, R. & Völkel, M. (2008). Cool URIs for the Semantic Web. *W3C Interest Group Note, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2008/NOTE-cooluris-20081203/ [Accessed 11/2/2012].

Sbodio, M. & Moulin, C. (2007) SPARQL as an Expression Language for OWL-S. In*: Proceedings of ESWC2007 Workshop on OWL-S: Experiences and Future Developments*, June 3-7th, Innsbruck, Austria.

Sbodio, M. L., Martin, D. & Moulin, C. (2010). Discovering Semantic Web Services Using SPARQL and Intelligent Agents. *Journal of Web Semantics,* 8(4):310-328.

Schenk, S. & Gearon, P. (2009). SPARQL 1.1 Update. *W3C Working Draft, World Wide Web Consortium (W3C)* [Online]. Available:

http://www.w3.org/TR/2009/WD-sparql11-update-20091022/ [Accessed 14/2/2012].

Seylan, I., Franconi, E. & Bruijn, J. D. (2009) Effective Query Rewriting with Ontologies over DBoxes. In*: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI2009)*, July 11-17, Pasadena, CA, USA. pages 923-925.

Shaw, M. (2002). What Makes Good Research in Software Engineering? *International Journal on Software Tools for Technology Transfer,* 4(1):1-7.

Speicher, S., Arwe, J. & Malhotra, A. (2014). Linked Data Platform 1.0. *W3C Last Call Working Draft, World Wide Web Consortium (W3C)* [Online]. Available: http://www.w3.org/TR/2014/WD-ldp-20140916/ [Accessed 17/9/2014].

Speiser, S. & Harth, A. (2011) Integrating Linked Data and Services with Linked Data Services. In*: Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011)*, May 29-June 2, Heraklion, Crete, Greece. Lecture Notes in Computer Science, Springer, volume 6643, pages 170-184.

Stadtmüller, S. & Norton, B. (2013). Scalable Discovery of Linked APIs. *International Journal of Metadata, Semantics and Ontologies,* 8(2):95-105.

Tabatabaei, S. G. H., Kadir, W. M. N. W. & Ibrahim, S. (2009). Automatic Discovery and Composition of Semantic Web Services Using AI Planning and WSMO. *International Journal of Web Services Practices* 4(1):1-10.

Tao, J., Sirin, E., Bao, J. & McGuinness, D. L. (2010) Integrity Constraints in OWL. In*: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, July 11-15, Atlanta, Georgia, USA. AAAI Press.

Taylor, K. R., Essex, J. W., Frey, J. G., Mills, H. R., Hughes, G. & Zaluska, E. J. (2006). The Semantic Grid and chemistry: Experiences with CombeChem. *Journal of Web Semantics,* 4(2):84-101.

Tummarello, G., Delbru, R. & Oren, E. (2007) Sindice.com: Weaving the Open Linked Data. In*: Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, November 11-15, Busan, Korea. Lecture Notes in Computer Science, Springer, volume 4825, pages 552-565.

Vandervalk, B., McCarthy, L. & Wilkinson, M. (2009) SHARE: A Semantic Web Query Engine for Bioinformatics. In*: Proceedings of the 4th Asian Semantic Web Conference (ASWC 2009)*, December 6-9, Shanghai, China. Lecture Notes in Computer Science, Springer, volume 5926, pages 367-369.

Verborgh, R., Steiner, T., Deursen, D. V., Roo, J. D., Walle, R. V. d. & Vallés, J. G. (2013). Capturing the Functionality of Web services with Functional Descriptions. *Multimedia Tools and Applications,* 64(2):365-387.

Verborgh, R., Steiner, T., Van Deursen, D., R., V. d. W. & Gabarró Vallés, J. (2011) Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In*: Proceedings of the 7th IEEE International Conference on Next Generation Web Services Practices (NWeSP2011)*, October 19-21, Salamanca, Spain. IEEE, pages 373 - 379.

Vinoski, S. (2008a). RESTful Web Services Development Checklist. *Internet Computing, IEEE,* 12(6):96-95.

Vinoski, S. (2008b). Serendipitous Reuse. *Internet Computing, IEEE,* 12(1):84-87.

Vitvar, T., Kopecky, J., Zaremba, M. & Fensel, D. (2007) WSMO-lite: Lightweight Semantic Descriptions for Services on the Web. In*: Proceeding of the 5th IEEE European Conference on Web Services (ECOWS 2007)*, November 26-28, Halle (Saale), Germany. IEEE, pages 77-86.

Webber, J., Parastatidis, S. & Robinson, I. (2010). *REST in Practice*, O'Reilly Media.

Wilkinson, M. D., Vandervalk, B. & McCarthy, L. (2009) SADI Semantic Web Services, Cause You Can't Always GET What You Want! In*: Proceedings of the Asia-Pacific Services Computing Conference (APSCC 2009)*, December 7-11, Singapore. IEEE, pages 113-18.

Zhao, H. & Doshi, P. (2009) Towards Automated RESTful Web Service Composition. In*: Proceedings of the 2009 IEEE International Conference on Web Services (ICWS 2009)*, July 6-10, Los Angeles, CA, USA. 1586928: IEEE, pages 189-196.

References

Zhou, B. & Yao, Y. (2010). Evaluating information retrieval system performance based on user preference. *Journal of Intelligent Information Systems (JIIS),* 34(3):227-248.

Zimmermann, O., Doubrovski, V., Grundler, J. & Hogg, K. (2005) Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned. In*: Proceedings of Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2005) Companion to the 20th Annual ACM SIGPLAN Conference*, October 16-20, San Diego, CA, USA. 1094965: ACM, pages 301-312.

Appendices

Appendix A: Research Strategies in SWS Approaches

Appendix A

Table 25 Analysis of research strategies in SWS

Publication	Description	Result	Validation
OWL-S (Martin *et al.*, 2004)	Presents the service ontology for marking up services semantically to provide automated Web service discovery, execution and composition.	[Specific Solution] • OWL-S ontology	[Examples] [Persuasion]
WSMO (Bruijn *et al.*, 2005a)	Presents the WSMO service ontology, part of the Web Service Modelling Framework (WSMF).	[Specific Solution] • WSMO ontology	[Examples] [Persuasion]
SAWSDL (Farrell and Lausen, 2007)	Defines SASWDL, a mechanism for extending WSDL documents for semantic annotation.	[Specific Solution] • SAWSDL Semantic Annotation Extension Mechanism	[Examples] [Persuasion]
SWS Coordination (Klusch, 2008a)	Presents a survey of matchmakers and composition planners for SWS description approaches (mainly OWL-S, WSMO and SAWSDL).	[Specific Solution: Result of an evaluation] • Matchmaking or composition Experiment	[Analysis: Experiment with statistically significant results] • The efficiency of a matchmaker or planner is typically measured on a test collection of Web service descriptions, and can be compared to the performance of similar matchmakers or planners. These indirectly provide evidence for the discoverability and composability of the approaches.
SWS Comparison (Cabral *et al.*, 2004)	Devised a conceptual model of SWS dimensions and used it to compare OWL-S, WSMO and other approaches.	[Specific Solution: Result of specific analysis] • Analysis of the approaches according to the model	[Persuasion] • Discussion of approaches according to the proposed model
WSDL-S (Akkiraju *et al.*, 2005)	Defines WSDL-S, a mechanism for extending WSDL documents for semantic annotation.	[Specific Solution] • WSDL-S Semantic Annotation Extension Mechanism	[Examples] [Persuasion]
SWSF (Battle *et al.*, 2005)	Defines the Semantic Web Service Framework (SWSF,) which includes Semantic Web Service Ontology (SWSO) and Semantic Web Service Language (SWSL).	[Specific Solution] • SWSO Ontology • SWSL • SWSF	[Examples] [Persuasion]
DSD (Klein *et al.*, 2005)	Introduces DIANE Elements (DE), a language for defining ontologies, and DIANE Service Description (DSD), and a process for creating service descriptions.	[Specific Solution] • DIANE Elements (DE) • DSD • Service Description Process	[Examples] [Persuasion]
SA-REST (Lathem *et al.*, 2007)	Defines SA-REST, a mechanism for semantic annotation of RESTful Web Services.	[Specific Solution] • SA-REST annotation mechanism	[Examples] [Persuasion]
hRESTS (Kopecky *et al.*, 2008)	Introduces hRESTS, a microformat for semantic annotation of RESTful Web services.	[Specific Solution] • hRESTS microformat	[Examples] [Persuasion]
MicroWSMO (Kopecky *et al.*, 2008)	Introduces MicroWSMO an extension of hRESTS the for semantic annotation of RESTful Web services.	[Specific Solution] • MicroWSMO microformat	[Examples] [Persuasion]

Publication	Description	Result	Validation
WSMO-Lite (Vitvar *et al.*, 2007)	Introduces WSMO-Lite, a lightweight service ontology.	[Specific Solution] • WSMO-Lite Ontology	[Examples] [Persuasion]
Kopecky (2012)	Introduces WSMO-Lite, MircoWSMO and hRESTS	[Specific Solution] • WSMO-Lite Ontology • hRESTS Mircroformat • MicroWSMO microformat • Matchmaking Experiment	[Examples] [Persuasion] [Analysis: Experiment with statistically significant results] • To demonstrate the viability of WSMO-Lite, in several SWS automation algorithms for discovery, ranking and composition have been adapted to WSMO-Lite and their performance compared to their original versions for SAWSDL and OWL-S.
RESTfulGrounding (Filho and Ferreira, 2009)	Introduces RESTfulGrounding ontology to map WADL to OWL-S.	[Specific Solution] • RESTfulGrounding Ontology.	[Examples] [Persuasion]
ReLL (Alarcon and Wilde, 2010)	Introduces ReLL, a vocabulary for describing Web pages and Web APIs.	[Specific Solution] • ReLL vocabulary	[Examples] [Persuasion] • Proof of concept implementation of a use case
SBWS (Battle and Benson, 2008)	Introduces SBWS, a method for integrating existing Web services by annotating WSDL and WADL documents so that these services can be used as if they were SPARQL endpoints.	[Specific Solution] • SBWS annotation method and implementation	[Examples] [Persuasion] • Implementation of the wrappers for Amazon RESTful Web services and using the descriptions for resolving SPARQL queries.
SPARQL descriptions (Sbodio *et al.*, 2010)	Introduces a method for representing preconditions and effects of Web services as graph patterns, and a method for their discovery using SPARQL queries	[Specific Solution] • A description method using graph patterns • A discovery method • Matchmaking experiment	[Examples] [Persuasion] [Analysis: Experiment with statistically significant results] • To demonstrate the efficiency of the discovery (matchmaking) method and descriptions a standard OWL-S test collection was transformed to SPAQL descriptions and the performance compared to the original version for OWL-S and associated matchmaker.
LIDS (Speiser and Harth, 2011)	Introduces LIDS, an approach for integrating data services with Linked Data	[Specific Solution] • LIDS approach • service description formalism • access mechanism for LIDS interfaces • LIDS wrapper	[Examples] [Persuasion] • Implementation of LIDS wrappers for GeoNames and Twitter, and used those to interlink with the Billion Triple Challenge dataset (BTC), and measured the time and added links as a result.

Appendix A

Publication	Description	Result	Validation
LOS (Krummenacher *et al.*, 2010)	Introduces Linked Open Services (LOS) as a method for describing both RESTful and non-RESTful Services as consumers and producers of RDF, and using SPARQL constructs for composing services exposing LOS descriptions.	[Specific Solution] • LOS method.	[Examples] [Persuasion] • A proof of concept implementation demonstrated with a use case.
Semantic REST (Battle and Benson, 2008)	Introduces Semantic REST, an implementation method to integrate REST-based websites into the Semantic Web.	[Specific Solution] • Semantic REST implementation method	[Examples] [Persuasion] • Implementation of a RESTful interface for a SPARQL endpoint for SemWebCentral.org mock semantic dataset.
Zhao and Doshi (2009)	Introduces a lightweight ontology for describing RESTful services as either sets of resources, instances or transitional services. It also introduces a conceptual model for representing the composition of services using situation calculus based state transition system.	[Specific Solution] • Lightweight ontology • Situation Calculus based STS	Examples] [Persuasion]
Hernandez and Garcia (2010)	Introduces a formal model for RESTful Web services using a combination of process calculus and triple space computing.	[Specific Solution] • A formal model for RESTful Semantic Web services	[Examples] [Persuasion] • Implementation of a RESTful interface for a SPARQL endpoint for SemWebCentral.org mock semantic dataset.
TSC (Riemer *et al.*, 2006)	Introduces Triple Space Computing (TSC) as a method for providing Semantic Web services.	[Specific Solution] • Architecture of TSC • TSC API design	[Persuasion] • Architecture and functionality specification
RESTdesc (Verborgh *et al.*, 2011)	Introduces RESTdesc an approach to describe Web APIs as N3 rules, in addition to a method for discovering and composing them.	[Specific Solution] • RESTdesc description approach • Method for discovery and composition	[Examples] [Persuasion] • Online demonstrator
iServe (Pedrinaci *et al.*, 2010b)	Introduces the iServe architecture and model that enables publishing service descriptions as Linked Data and supports annotating services with a Minimal Service Model (MSM).	[Specific Solution] • iServe Architecture • Publishing platform, • annotation tools • MSM Service description vocabulary	[Examples] [Persuasion] • Online demonstrator
SADI (Wilkinson *et al.*, 2009)	Introduces SADI, a framework to facilitate automatic integration of bioinformatics data and services.	[Specific Solution] • A method for conceptualising SWS for bioinformatics by imposing constraints on how I/O are defined in domain ontolgoy • A method for discovering services • A method for composing them using SPARQL queries	[Examples] [Persuasion] • Code available online • Online demonstrator • Implementation of two use cases, one in SHARE, and the other as a Taverna plug-in

Publication	Description	Result	Validation
HyperData (Kopecky *et al.*, 2011)	Description mechanism for RDF APIs and the integration of those descriptions as triples stored with the data, so that RDF data self-describes how it is updated.	[Specific Solution] • A method for the data to describe how it can be updated • A vocabulary to describe the resources as graphs and the relationships between them	[Examples] [Persuasion] • proof-of-concept triple-store wrapper demonstrated in a use case
Hypermedia RDF (Kjernsmo, 2012)	Vocabulary for making RDF a hypermedia type that not only describes data but what actions are applicable to it.	[Specific Solution] • A vocabulary for describing what actions are applicable to a certain RDF resource	[Persuasion] • Argument
RDF-REST (Champin, 2013)	Design of an RDF-REST approach to bridge the gap between RESTful Web services and Linked Data, by building conventional RESTful services on top of Linked Data.	[Specific Solution] • RDF-REST Architecture • Implementation of RDF-REST	[Examples] [Persuasion] • Is part of a real application, kernel for Trace-Based Systems, kTBS
SSWAP (Gessler *et al.*, 2009)	Introduces the design of SSWAP Protocol Architecture and implementation for creating describing publishing and discovery Web services to design RESTful Semantic Web services by describing a mapping between its inputs and outputs, using an RDF graph template.	[Specific Solution] • SSWAP Protocol • SSWAP Architecture • SSWAP Implementation	[Examples] [Persuasion] • Code online • Pipeline discovery platform • Online directory of services where more services can published
SWS Challenge (Petrie *et al.*, 2009)	The aim of the challenge is to explore the trade-offs among existing Semantic Web service approaches.	[Specific Solution] • Scenarios • Evaluation Framework	[Evaluation] • Qualitative evaluation of how well each approach achieves the scenarios
S3 Contest	S3 Contest on Semantic Service Selection, the reference contest for evaluating semantic service matchmakers.	[Specific Solution] • Test Collections • Evaluation Framework • Matchmaking Experiment	[Analysis: Experiment with statistically significant results]
(Bachlechner and Fink, 2008)	Study involved surveying and analysing opinions from both practitioners and researchers to evaluate the potential of Semantic Web services as integration architectures.	[Specific solution: answer or judgement] • Expert opinions on the potential of Semantic Web services as integration architectures.	[Evaluation] • Expert interviews and questionnaires

Appendix B: Web Service Scenarios and RO Models

Mashups

M1: Yahoo Pipes (Mashups)

The scenario is an example of creating a mashup using Yahoo Pipes. Yahoo Pipes is an interactive Web application which enables the creation and execution of mashups. It offers a workspace in which a user can add widgets such as data sources, filters, and functions to refine and merge the data.

A user has built a stock quote watch mashup using Yahoo Pipes (Donnelly, 2010), this displays the last quote and chart for the stocks. In this example, he uses the widgets provided to retrieve the original stock data from a .csv file stored at the Yahoo Finance downloads. He then uses a filter widget to filter the stock file for certain stock quotes. To loop through the obtained data he uses a loop widget that displays the results as a chart.

Infrastructural and Functional Requirements

1. Proprietary Workflows - The workflows description is not in an open format; it is specific to the platform executing it.

2. Workflows are controlled by and executed on one machine - There is no need for a participation of multiple machines or a their coordination.

3. Server/Service provider ownership of data - The data accessed by the client belongs to the service provider.

4. Open Accessibility to the Data - The data is accessible; there are no security restrictions.

5. Creation of the workflows is done by the end user with a GUI - Mashup Creator's level of expertise is minimal; the filtering and programming is through GUI, no coding is required from the end user, EU.

Non-functional Requirements

Tolerance of failure - In this scenario, and many other mashup scenarios, mashups are used by end users for providing specialised data for non-critical tasks, so the failure of mashups does not have a large impact on other tasks.

Scenario Breakdown

The generic scenario of building mashups using Yahoo Pipes (Donnelly, 2010) is broken down into the following steps:

(1.) The client creates a mashup;

(2.) It creates widgets that read inputs from other widgets or external resources;

(3.) The widget produces the results;

(4.) The client reads the results.

Resource-Oriented Model

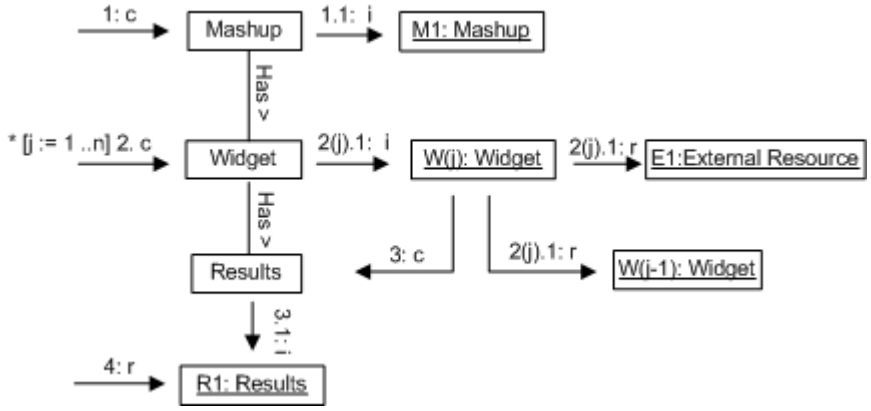


Figure 26 RO Model of M1

In this scenario, step 2, (creating widgets) is iterative. We used the `*[j:= 1..n]` UML convention to indicate this. The `Has` links show the structural relationships between the mashup, its widgets, and the results.

M2: The MashMaker Scenario, Desktop Mashups

(Ennals and Garofalakis, 2007) describe MashMaker, an interactive browser plug-in for creating mashups from Intel. The scenario provided explains how a user, who is planning to rent a house, uses MashMaker.

A user is interested in houses that have the best restaurants around. The user visits a housing website and adds it to MashMaker by clicking an icon in the browser. The houses are displayed in MashMaker as a tree where each house is a node, when a node is clicked, MashMaker suggests appropriate queries like "things nearby". The user searches for food nearby, then applies a filter widget to include only those within 0.5 a mile and having a rating of 3 or more. He adds a count widget to count how many restaurants match these criteria, and then copies this widget to the other houses, saves it, and publishes it.

The interaction occurs between the different Web servers where the data resides and MashMaker on the client. The actual processing and aggregation of the data happens on the client. However, in case of overlaying information on maps, Google Maps is utilised and some of the processing happens on the Google Maps Server. Then the results are transferred to the client.

Infrastructural and Functional Requirements

Similar to the requirements discussed in M1

Non-functional Requirements

Similar to the requirements discussed in M1

Technical Notes

Appendix B

1. **Client/Intermediary Data processing, filtering and aggregation**
   The processing of the data is performed on the client or partially, on an intermediary server like Google Maps.
2. **Data aggregating compositions**
   The compositions involved in creating mashups are based on joining data providing services, where the composition depends on matching elements or attributes of the data.
3. **Standards of data resources**
   The formats of the data sources vary, from HTML (Web pages), RSS, JSON to RDF.
4. **Scalability issue**
   Although the mashup is executed on the client there is a point that could affect the scalability of the architecture: this is the MashMaker server that hosts a database of extractors (Ennals *et al.*, 2007). Extractors describe how to extract structured information from HTML pages. The creation and maintenance of extractors is done in a wiki collaborative manner. The scalability issue is minor if the extraction is executed on the client, which seems to be the case, although is not explicitly stated.
5. **Architecture**
   1. **Multiple Servers for Data Sources**
   2. **An intermediary server for maintaining extractors and mashup reuse**
   3. **An application on the client to create mashups**

<u>Scenario Breakdown</u>

(1.) The user creates a mashup

(2.) The user creates two Web resources that link two websites to mashup

(3.) The user updates the mashup to mash the two Web resources

(4.) The user runs the mashup

(5.) The mashup returns the results

<u>Resource-Oriented Model</u>

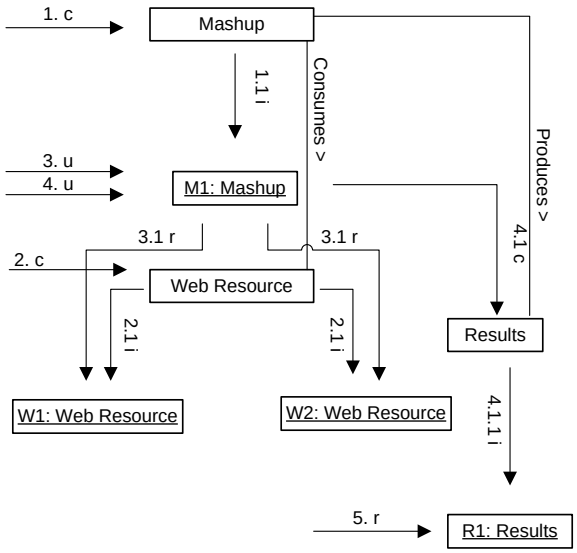


Figure 27 RO Model of M2

M3: Displaying the time and location of a Website's visitors using a layered mashup architecture

(Biornstad and Pautasso, 2009) Proposed a layered architecture for creating mashups from streaming data. Their approach is similar to Yahoo Pipes, where the mashup architecture executes the mashup and the results are sent to the client. They provide an example of a mashup that combines a Web server's log file with a geolocation service.

In this scenario a user wants to display the geographic locations of a website's visitors on a map. This map is constantly updated. He or she does that by using the system built on this architecture to access the Web server's log, through a secure shell socket (SSH). This provides real-time updates through a streaming push mechanism, in contrast to a request/response mechanism using HTTP, which increases the latency and network traffic. The user then uses the system to create components to extract the IPs from the log, resolving the DNS, looking up the coordinates and overlaying them on a map, which is the sent to the client.

The requirements are similar to the ones in scenario M2. However, there are some additional ones:

Infrastructural and Functional Requirements

1. **Accepts Streaming Data pushed by servers**

   Unlike other approaches it accepts data pushed to the mashup engine over open ports;

Non-functional Requirements

1. **Secure Access**

   In this scenario, access is enabled to access secure files on remote Web servers using SSH;

Appendix B

Technical Notes

1. Standards of data resources

   Web logs, RSS, JSON, accesses data from Web services using SOAP.

Scenario Breakdown

(1.) The application reads a Web log.

(2.) A local copy of the Web log is created.

(3.) The client reads the local copy. (The search is discussed in the modelling issues.)

(4.) IPs are extracted from the Web log.

(5.) The IPs are read to be sent to the DNS.

(6.) Resolve the IPs at the DNS.

(7.) Create a resource representing the DNS coordinates

(8.) Getting the Coordinates from the DNS.

(9.) Creates a Map.

(10.) Overlay the Map with the coordinates.

Resource-Oriented Model

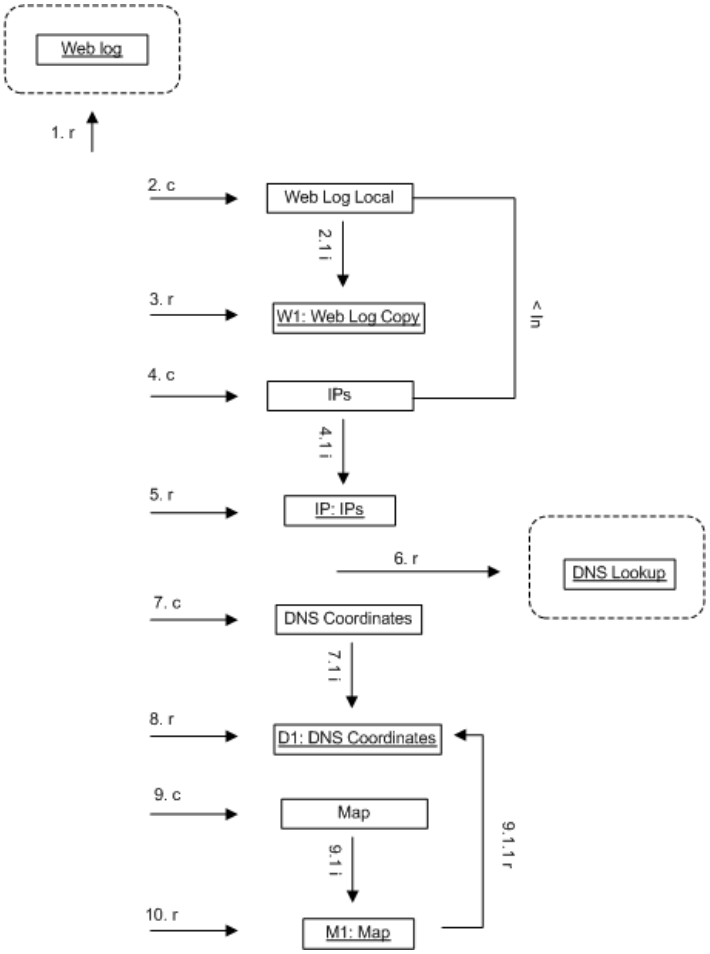


Figure 28 RO Model of M3

M4: Creating situational applications using the enterprise information mashup fabric

In (Jhingran, 2006), the author discusses the enterprise's need for *Situational Applications.* The author describes these as "applications that come together for solving some immediate business problems". The paper describes two scenarios to illustrate where it would be useful.

M4A: *In the first example a salesperson needs information on a client before making a call on prospect. The information needed is how much was sold to the customer during the last quarter, and did the customer have problems with sales.*

M4B: *A CFO that has a meeting with his CEO. The CFO wants to present a summary of the financial picture. This summary needs to be assembled from emails by finance personnel including presentations that contain embedded spreadsheets about the financial picture.*

Infrastructural and Functional Requirements

1. Information Assembly

In M4A there is no merging done on the data on information assembly.

Non-functional Requirements

1. Closed

The system is to be used inside an enterprise.

Technical Notes

1. Standards of data sources

The data depends on the applications that the enterprise uses; the more the mashup engine understands the formats of enterprise data, the more useful it would be.

Scenario Breakdown

(1.) Query the Customer info.

(2.) Reading the results of the query,

Resource-Oriented Model

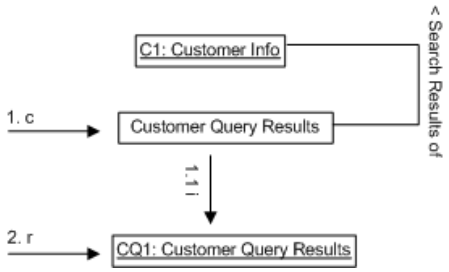


Figure 29 RO Model of M4

Appendix B

Enterprise Services

E1: SSPD (City University) (Enterprise Services)

The scenarios chosen were two integration projects from City University (City University, 2008). The first project was Single Sourcing of Programme Data (SSPD). The university uses information about the study programmes in different processes, like producing student handbooks, publishing programme information on the website, producing prospectus and quality and approval processes for development of new programmes. These processes are using the same information but they were operating independently. This led to inconsistencies in data and effort duplication.

SSPD is concerned with how programme information is created, updated and used enabling the processes mentioned above to be facilitated and any inconsistencies resolved. It enables academic and administrative staff to define and maintain module and programme specifications and submit them for approval.

Infrastructural and functional requirements

1. Complete control over the service providers and service consumers - The university systems are the service providers and the service consumers. There are no external entities involved.
2. Actions are triggered as a result of service invocation, so it is not a read only situation - The state of resources can be altered because of the service invocation.
3. The ability to deal with multiple systems and data formats – The services deal with legacy systems that use different technologies and formats to represent the data.

Scenario Breakdown

The SSPD scenario from City University (City University, 2008) can be decomposed into the following steps:

(1.) Academic staff read the program info.
(2.) Creates a modification.
(3.) Can update it, when it is finished.
(4.) It is approved by the administrative staff.
(5.) The program info is updated.
(6.) It can be read by interested processes.

Resource-Oriented Model

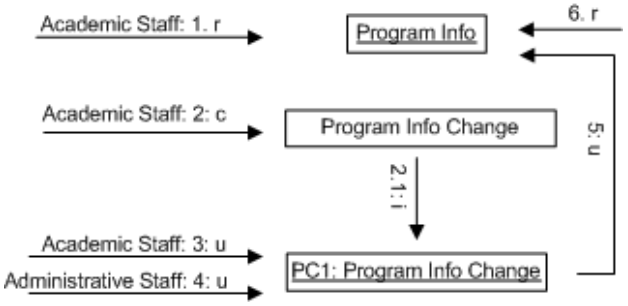


Figure 30 RO Model of E1

With step (3.), an update can also change the status of the modification to indicate it is ready to be submitted. Figure 30 shows how roles are modelled, with the name of the role associated with the action on the messages.

E2: MLE (City University) (Enterprise Services)

The other integration project from City University is called the Managed Learning Environment (MLE)

The University uses both the SITS:Vision student information management system, and a Virtual Learning Environment (WebCT Vista). The transfer of student information from SITS:Vision to WebCT Vista took place using a nightly scripting process, this was slow and had errors. MLE aims to have the SITS system trigger the updating process so that new information is added to WebCT directly.

Infrastructural and functional requirements

Similar to the requirements discussed in E1

Scenario Breakdown

The other integration project from City (MLE) is modelled below. The steps involved in MLE are

(1.) The SITS system creates updates,

(2.) The SITS system notifies WebCT,

(3.) WebCT reads the changes and gets updated.

Resource-Oriented Model

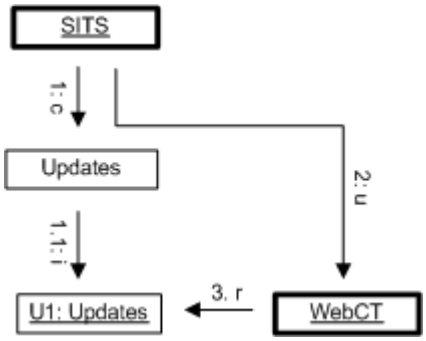


Figure 31 RO Model of E2

WebCT and SITS are active resources, indicated by the heavy lines (a UML convention). This means they initiate control activity.

E3: BT.com (Integrating BT's OSS)

BT used Web services to integrate core operational support systems (OSS) which are legacy subsystems to enhance existing services or provide new ones. The following scenario mentioned in (Calladine, 2004) illustrates this.

BT.com Online website

BT.com offers many customer services such as 'View my bill', 'Friends and Family', etc. BT would like its customers to use the website because it reduces the cost of operator-assisted services. BT.com needs access to core services from multiple internal heterogonous sub-systems.

Infrastructural and Functional Requirements

Complete control over the service providers and service consumers.

BT systems are the service providers and the service consumers; there are no external entities involved.

Actions are triggered as a result of service invocation, so it is not a read only situation.

The state of resources can be altered because of the service invocation.

The ability to deal with multiple systems and data formats.

The services deal with legacy systems that use different technologies and formats to represent the data.

Scenario Breakdown

The customer can read the bill, this will invoke reads to the subsystems.

The customer can update and read Family and Friends options, this will also invoke update and read requests to the system.

Resource-Oriented Model

Figure 32 RO Model of E3

E4: SCORe (Integrating BT's OSS)

Another BT project for the integration of operational support systems (OSS) is the SCORe scenario (Calladine, 2004).

Project SCORe (Service Consolidation and Operational Revitalisation)

A problem that was identified with the call-centres is the complexity of retrieving the data relative to a customer's contact. SCORe aims at reducing costs and increasing customer satisfaction. Because the data is held in multiple databases and controlled by several systems, this means that several calls to these systems were needed, using different technologies.

Infrastructural and Functional Requirements

Similar to the requirements discussed in E3.

Scenario Breakdown

The operator can retrieve customer information, which then retrieves it from the subsystems.

Resource-Oriented Model

Figure 33 RO Model of E4

Appendix B

B2B

B1: Reverse Auctioning (B2B)

The scenario modelled here is a reverse auctioning scenario mentioned in (Decker and Weske, 2007):

"*A buyer (e.g., car manufacturer) uses reverse auctioning for procuring specially designed components. In order to get help with selecting the right suppliers and organizing and managing the auction, the buyer outsources these activities to an auctioning service. The auctioning service advertises the auction, and beforehand, different suppliers can request permission to participate in it. The suppliers determine the shipper that would deliver the components to the buyer or provide a list of shippers with different transport costs and quality levels, which the buyer can choose from. Once the auction has started, the suppliers can bid for the lowest price. At the end, the buyer selects the supplier according to the lowest bid. After the auction is over, the auctioning service is paid.*"

Infrastructural and functional requirements

1. Registration - The auctioning service deals with many participants/clients that need to register before using the service. This implies the need for authentication and authorisation

2. Support for different client roles - There are two different roles for users of this service: buyers and suppliers.

3. The service provider and the service consumers are different entities

The service provider is the auctioning service, and the consumers are the buyer and the suppliers.

Non-functional requirements

1. Security

This involves authentication and authorisation for service consumers and encryption of payment transactions.

Scenario Breakdown

The reverse auctioning scenario mentioned in (Decker and Weske, 2007) can be broken down into these steps:

(1.) The buyer creates an auction.

(2.) The buyer starts the auction.

(3.) The suppliers place their bids.

(4.) The buyer selects a bid.

(5.) The buyer pays for the service.

(6.) The buyer deletes the auction.

Resource-Oriented Model

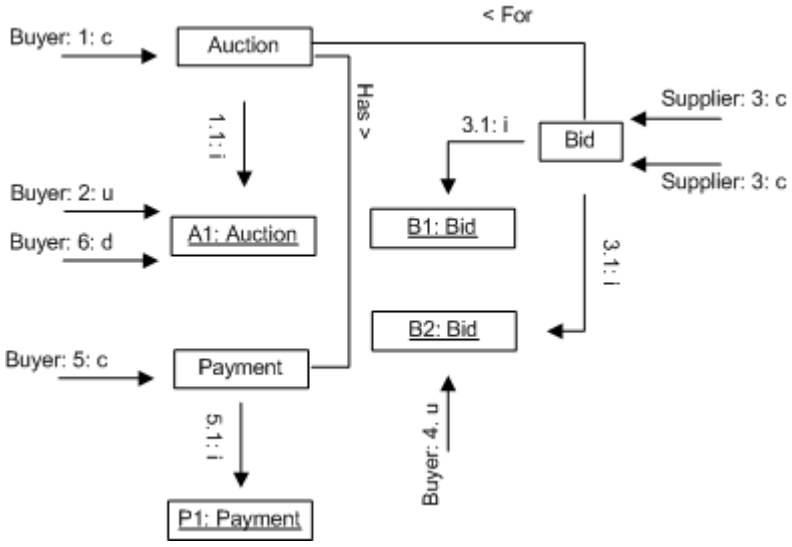


Figure 34 RO Model of B1

B2: Telecommunications Wholesaler

In (Zimmermann *et al.*, 2005), the authors discuss an IBM project that aims to enable a large telecommunications wholesaler to supply services to more than 150 customers. The wholesaler owns the physical network. The customers are either telecommunications companies extending their own network infrastructure, or companies that want to bundle telecommunication services with their products. These customers will use the order management services of the wholesaler to connect, configure, or disconnect telephone services for end users. The order management application should offer two main processes:

1. Provide a new Public Switched Telephone Network (PSTN) telephone service.
2. Move a PSTN telephone service to a new address.

A customer needs to follow the next steps, summarised from (Zimmermann et al., 2005), in order to perform the aforementioned processes:

1. *Identify the service to be moved and its current location or site address.*
2. *Identify the new address for the service. This has to be the address as recognized by the systems that record telecommunications plant and service information. Hence search aids are required.*
3. *When a recognized address is identified, the next step is to search for a transmission cable plant which exists at the target address and could be reused for provisioning this service.*

4. *Having identified a particular copper transmission path, this result has to be recorded.*

5. *Determine the features of the service at the new address, which depends on a complex set of factors. Some features may already exist from a previous service at this address, some transferred from the old address, and some may be requested.*
6. *Next, determine a phone number for the service at the new address and reserve it. The old number maybe kept, if the network at the new address permits, otherwise a list of numbers available must be supplied.*
7. *If a visit is required, then a time must be negotiated which suits both the customer and the field staff to be assigned to the task.*
8. *The request to move and the reservation is confirmed, allowing the commercial transaction to proceed.*

Infrastructural and Functional Requirements

1. Negotiation

The service infrastructure should support conventions that enable the service provider and service consumers to negotiate.

2. Workflow support

The processes needed involve the invocation of several services in a certain order.

3. Conversational services

The service infrastructure should enable execution of services where the all inputs cannot be known upfront.

Non-functional Requirements

4. Security

This involves authentication and authorisation for service consumers.

Scenario Breakdown

(1.) The client creates a service request.

(2.) Adds the new address of the service.

(3.) Determines the features of this service.

(4.) A number is created:

(4.A) A list of new numbers,

(4.B) The old number is kept.

(5.) Choose a number:

(5.A) The client chooses a number,

(5.B) The old number is read.

(6.) [Optional] A visit is arranged.

(7.) The client pays for the service.

Resource-Oriented Model

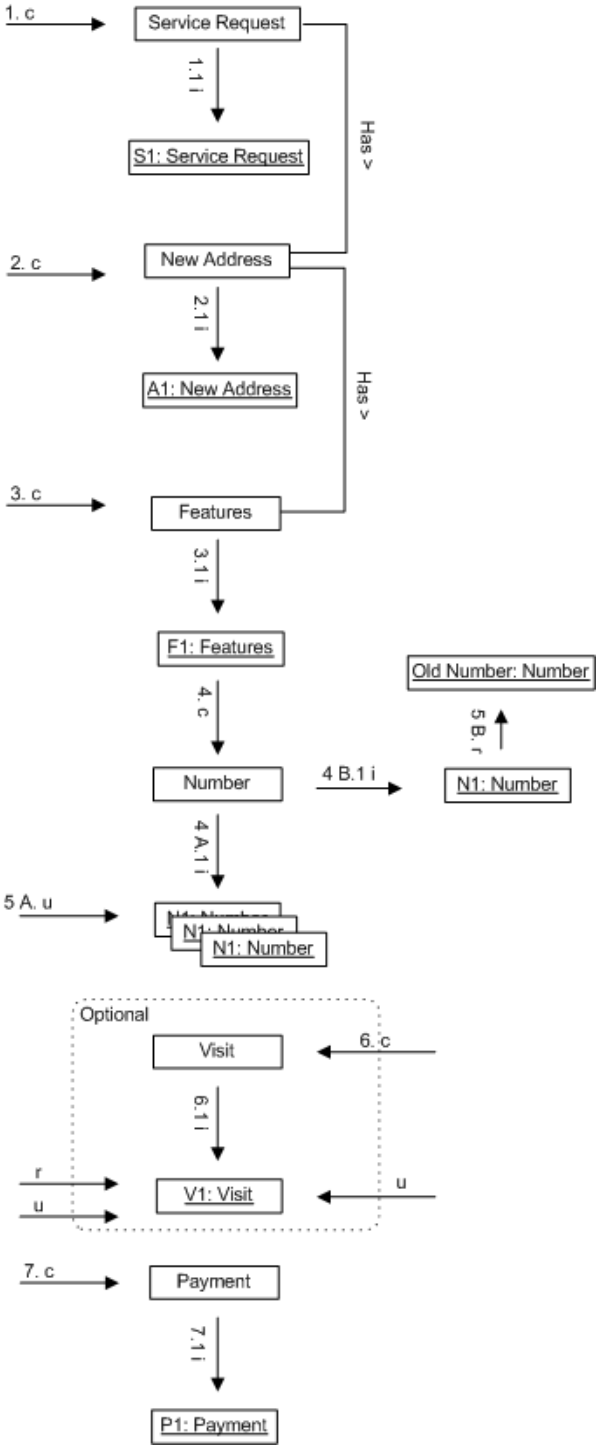


Figure 35 RO Model of B2

B3: E-Procurement

(Brodie, 2000) presents an e-procurement general scenario:

Appendix B

> ***"E-procurement has a buy side, a sell side, and the connection of the two. On the buy side, a customer such as a company purchasing agent needs to access information on all relevant products, including product specifications, comparisons with all competitive products, pricing including discounts, delivery arrangements, and promises. The seller must have all relevant information on the buyer, including company, finance, credit, contact, logistics, preferences, and legal. On the sell side, the vendor must provide all relevant, up-to-date catalogue information from hundreds or thousands of suppliers, together with real-time inventories and pricing. For a sale, transaction details must be irrefutably committed on both sides, and reflected in the inventory and financial systems."***

Infrastructural and Functional Requirements

The characteristics are identical to the ones in scenario B2.

Non-functional Requirements

Security

This involves authentication of buyers and sellers and the encryption of payment transactions.

Scenario Breakdown

(1.) The buyer reads the catalogue.

(2.) The buyer places the order.

(3.) The seller provides the pricing for that order.

(4.) The buyer reads the pricing.

(5.) The buyer provides the payment.

Resource-Oriented Model

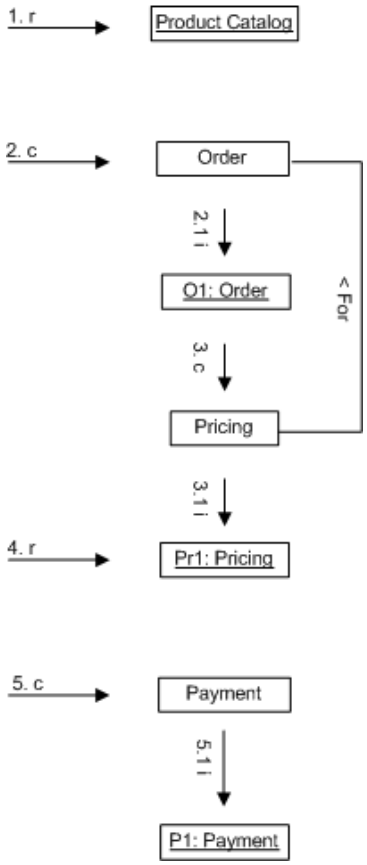


Figure 36 RO Model of B3

B4: Supply Chain Management

A scenario mentioned in (Preist *et al.*, 2005) illustrates an example of a supply chain and the different entities and interactions involved:

"We consider a manufacturing company in Bristol, UK, which needs to distribute its goods internationally. It does not maintain its own transportation capability, but instead outsources this to other companies, which we refer to as Freight Forwarders. These companies provide a service to the manufacturing company – they transport crates on its behalf. However, the manufacturing company still needs to manage relationships with these service providers. One role within this company, which we refer to as the Logistics Coordinator, is responsible for doing this. Specifically, it carries out the following tasks;

1. *Commissioning new service providers, and agreeing the nature of the service they will provide (e.g. locating a new freight forwarder in Poland, and agreeing that it will regularly transport crates from Gdansk to Warsaw).*
2. *Communicating with service providers to initiate, monitor and control shipments (e.g. informing the Polish freight forwarder that a crate is about to arrive at Gdansk; receiving a message from them that it has been*

delivered in Warsaw, and they want payment). This is done using one of the messaging standards, EDIFACT.

3. *Coordinating the activity of service providers to ensure that they link seamlessly to provide an end-to-end service (e.g. making sure the shipping company plans to deliver the crate to Gdansk when the Polish transport company is expecting it; informing the Polish company when the shipping company is about to drop it off).*
4. *Communicating with other roles in the company to coordinate logistics with other corporate functions (e.g. sales, to know what to dispatch; financial, to ensure payment of freight forwarders).*

In our scenario, we consider a specific logistics supply chain from Bristol, UK, to Warsaw, Poland. It consists of three freight forwarders. The first is a trucking company, responsible for transporting crates from the manufacturing plant in Bristol to the port of Portsmouth, UK. The second is a shipping company, responsible for shipping crates from Portsmouth to the Polish port of Gdansk. The third is another trucking company, which transports crates to the distribution warehouse in Warsaw. We assume that the Logistics Provider communicates with the Freight Forwarders using the EDIFACT standard, and is already successfully using this logistics chain."

Infrastructural and Functional Requirements

The requirements are identical to scenarios B2 and B3. However, there are others:

1. **Mediating between different standards**
   In this example, EDIFACT and RosettaNet, and this involves both the mediation of data and the mediation of protocols used.
2. **Discovery of services**
   In this example, EDIFACT and RosettaNet, and this involves both the mediation of data and the mediation of protocols used.

Non-functional Requirements

Similar to B1's requirements

Scenario Breakdown

(1.) Logistics coordinator creates a supply chain.

(2.) Read the offered services from the shipping company.

(3.) Logistics coordinator creates a service request.

(4.) The shipping company creates an offer.

(5.) Logistics coordinator agrees to that offer.

(6.) The shipping company starts a shipment.

(7.) Logistics coordinator updates the supply chain with info from the agreement

(8.) Logistics coordinator updates the shipping monitor with info from the shipment.

(9.) The shipping monitor monitors the shipment.

Resource-Oriented Model

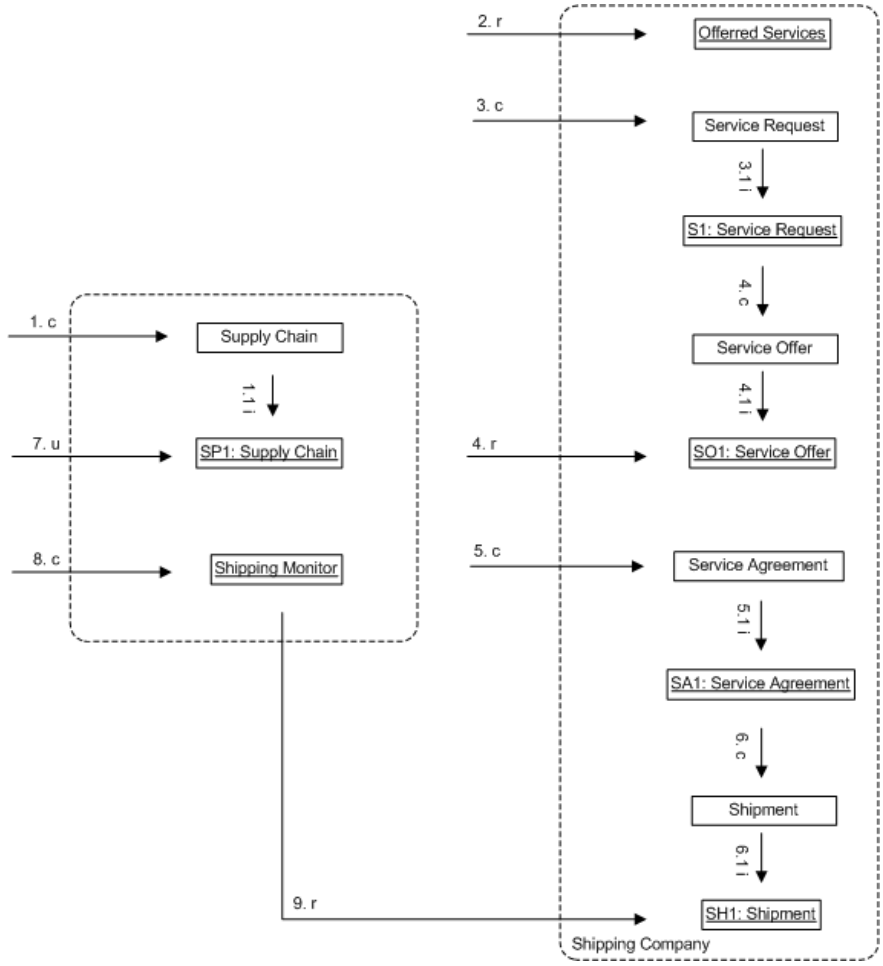


Figure 37 RO Model of B4

Cloud Computing

C1: NYT TimesMachine

The cloud computing scenario we chose is the New York Times project called TimesMachine, which is discussed in (Klems *et al.*, 2008). It aims to provide access to issues dating back to 1851, adding up to 11 million articles.

The technical team wanted to generate the PDF files from TIFF images. The generation was done based on request. However, this solution would not work for high traffic. The team decided to generate all the PDF files and serve them on request. The size of the TIFF files was 4 Terabytes. So they used Amazon's Elastic Compute Cloud (EC2) and Simple Storage Service (S3). The TIFF files were uploaded to S3 and they started a Hadoop cluster of 100 customized EC2 Amazon Machine Images. They transferred the conversion application. That resulted in the conversion to PDFs and storing the results to S3 taking only 36 hours.

Infrastructural and functional requirements

Appendix B

1. Configuration of Virtual Machines - In this scenario, the Amazon Machine Images (AMI) were configured to form a Hadoop cluster. This can be done through a Web-based control panel or through Web services. EC2 offers a SOAP interface and a query interface.

2. Transferring large amounts of data to and from the servers - This implies the need for reliable, efficient and secure data transfer. This is explained discussed in the following 3 points.

3. The data is owned and manipulated by the client - In contrast to mashups where the client requests the data, here clients request resources to manipulate their data.

4. The client transfers the job/application to the servers - In this scenario the client uploads to the cloud the application that manipulates the data.

5. Multitenancy - This means that the services and resources are used by multiple clients other than the New York Times and this implies a stronger need for security and for resource virtualisation.

6. Batch processing - Interaction with the server does not need to happen during the processing.

<u>Non-functional requirements</u>

1. Service Level Agreements - There is no formal specification for the agreement, as the SLA is a webpage. Therefore, the negotiation of SLA is not automated.

2. Reliability - This should be based on the SLA and include:

The availability of services;

The recoverability of data and applications.

Since it is built on a business model, what are the penalties in the case the reliability criteria are not met?

3. Security - The security involves:

The authentication and authorisation of the service consumer, in this case the technical team at The New York Times:

The encryption of the communication to guarantee confidentiality

The encryption of the data and applications on the client which are owned by the clients, to ensure that no one else can access them.

4. Monitoring - Amazon offers a Web console, command line tools, and a Web API (Web service) to monitor the instances.

<u>Scenario Breakdown</u>

The New York Times project scenario TimesMachine (Klems *et al.*, 2008) is decomposed into the following steps:

(1.) Create the data items, upload the images;

(2.) Create a Hadoop Cluster;

(3.) Create an application and upload the converter;

(4.) The application returns the results;

(5.) The client reads the results.

Resource-Oriented Model

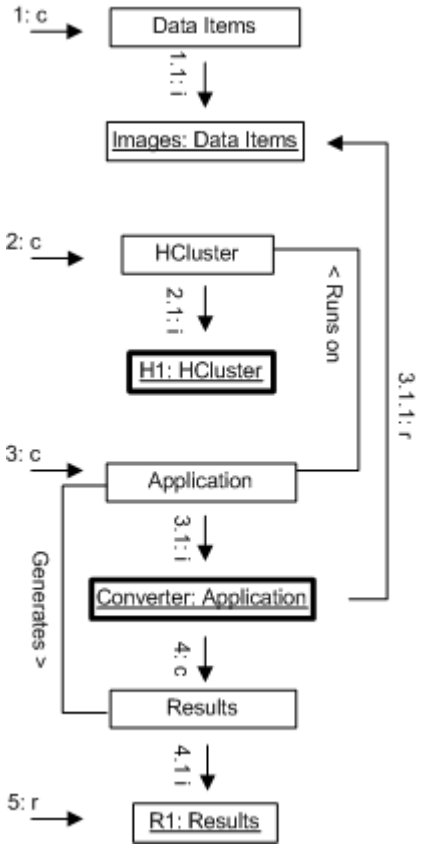


Figure 38 RO Model of C1

The client sends the representation of the resource when creating or updating it, the client receives a resource representation when it reads a resource.

C2: Major League Baseball MLB Website's Chat System

Another scenario mentioned in (Klems *et al.*, 2008), the MLB Advanced Media a company that develops and maintains the MLB websites wanted to add a chat service.

The technical team faced the problem that this chat service has to be up and running at a very short notice, there was no time to buy and set up new equipment. So they decided to use machines from Joynet, a cloud computing provider. The machines acquired were used to test and launch the new product. At the development stage they needed 10 virtual machines and 20 for the chat clusters. When they launched the chat system they needed extra RAM for the machines; when the playoff and World Series started they needed extra machines with extra RAM and processing power. When the season ended they could scale down on the resources required.

Infrastructural and Functional Requirements

Appendix B

The requirements are similar to C1. However, they differ in some technical issues.

1. Flexible Scalability

   The resources are utilised efficiently, acquired when needed or released otherwise.

2. Standards used

   There is no Web API (Web service) interface to Joynet services.

3. Used as hosting server

   The scenario described here is more like a hosting server than cloud computing.

Non-functional Requirements

Identical to C1s requirements

Scenario Breakdown

(1.) Create Machine instances.

(2.) Increase the number of machines and increase their RAM.

(3.) Install "Create" the chat system.

(4.) Run the Chat system.

(5.) Increase the RAM in the machines.

(6.) Scale down the machines.

Resource-Oriented Model

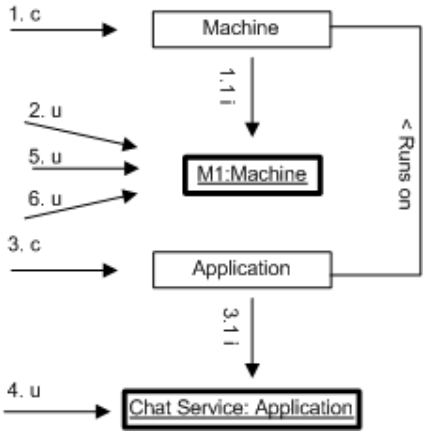


Figure 39 RO Model of C2

C3: Colorado State University using Google Apps

In (Herrick, 2009), the author discusses Colorado State University's use of Google Apps, including Google Mail, Google Calendar, and Google Talk, Google Docs, Google Sites and Google Video.

In 2009, Colorado State University (CSU) used Google Apps as an e-mail hosting solution for its undergraduate students. Google Apps Education Edition, is free for colleges and universities. CSU wanted to replace their old system with an outsourced e-mail and collaboration solution. The important issues were cost, reliability and the scope of services. Google Apps was selected mainly because it

offered e-mail, calendar and personal website services for students. Moreover. the interoperability between these applications was a also plus. This increased students' collaboration and communication. The faculty and move their accounts to use the suite because of its potential.

Infrastructural and Functional Requirements

The requirements are similar to C2. However it differs in the following

1. It is a Software as a service
2. Instead of acquiring software solutions, the university used Google Apps.
3. The client does not transfer applications to the server.
4. The client uses the services as applications existing on their systems.

Non-functional Requirements

Identical to C1s requirements.

Scenario Breakdown

(1.) Create a user account.

(2.) Pay for the service.

(3.) The Apps are created for this account.

Resource-Oriented Model

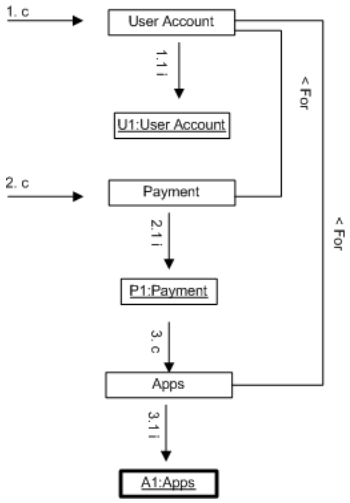


Figure 40 RO Model of C3

C4: LingoSpot, a business built using Google App Engine

LingoSpot is one of the case studies mentioned in Google App Engine's documentation[21]. Lingospot provides services for online publishers to help readers discover more of their content, including virally-distributed widgets for related videos and articles, as well as smart discovery links within context.

> *"We use the App Engine to scale our services to Web audiences limitlessly, ranging from a million+ users in 30 minutes at large sites, to supporting*

[21] Google App Engine, App Engine Developer Profiles, http://code.google.com/appengine/casestudies.html

hundreds of smaller sites that have installed our viral widgets, without worrying an iota about provisioning capacity for the traffic and growth. Google App Engine enables users to run programs written in Python or Java, it also offers APIs to access datastore, Google Accounts, URL fetch, Google Maps, and email services. It offers a Web-based Administration Console to manage applications."

Infrastructural and Functional Requirements

1. **Platform as a service**
2. **LingoSpot used Google Apps Engine as a development and hosting platform**
3. **Dynamic Scalability**
4. **The system autonomously responds to the peaks on demand.**

Non-functional Requirements

Identical to C1s requirements.

Scenario Breakdown

(1.) Create a user account.

(2.) Read the SDK.

(3.) Upload the application.

(4.) Run the application.

Resource-Oriented Model

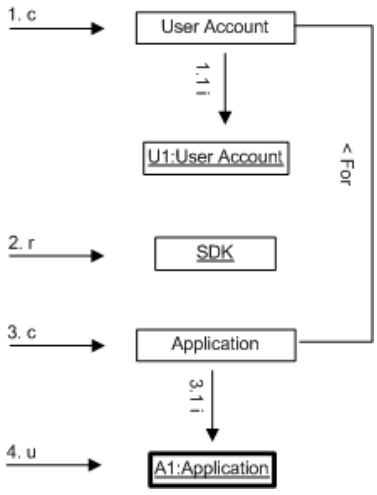


Figure 41 RO Model of C4

Grid Computing

G1: NEESgrid (Grid Computing)

NEES is an NSF funded project to build a virtual laboratory for earthquake engineers. Using grid technologies it enables remote access and control to observational sensors, experimental data, computational resources, and earthquake engineering control systems such as shake tables, reaction walls, and robots. NEESgrid also enables access to collaboration tools (Gullapalli *et al.*, 2004).

Earthquake engineers wanted to study the effect of an earthquake on different types of substances and structures. These different structures and their shake tables are distributed across a number of labs. The aim was to coordinate these experiments with computer simulations. So the Multi-site Online Simulation Test, MOST, was devised to test and illustrate this capability using the NEESgrid system. MOST combined physical experiments testing the effect of an earthquake on the interior of a multi-story building at three different sites, each testing a part of the structure. MOST linked the physical experiments at the University of Illinois at Urbana-Champaign (UIUC) and at the University of Colorado, Boulder (CU) with a numerical simulation at National Centre for Supercomputing Applications (NCSA). A simulation coordinator coordinates the overall experiment (Pearlman et al., 2004).

<u>Infrastructural and functional requirements</u>

1. Remote access to instruments - Services can be interfaces to instruments, in this case lab instruments such as shake tables.

2. Notifications - Running services send notifications to the clients or to the service/job scheduler.

3. Batch Processing - When a service or job is run, there is no need for the client to interact and results are delivered when it stops.

4. Coordination between running services - The services communicate to ensure correct synchronisation.

5. Negotiation - It involves interactions between the client and the server to ensure compliance between the client's requirement and the server's policies.

6. Support of sending and receiving large volumes of data - Large volumes of data are being transferred between different services, requiring reliable, efficient, and secure transfer.

7. Service Scheduling - Services are invoked and controlled by schedulers, in this case the Experiment Coordinator is controlling several experiment executions.

<u>Non-functional requirements</u>

1. Security - The security involves:

- **The authentication and authorisation of the researchers and scientists to protect sensitive data and applications;**
- **The encryption of the messages and transferred data to guarantee confidentiality.**

2. Monitoring - This is needed to ensure that the different components are functioning.

3. Reliability - Reliable data transfer and service execution, no delays, interruptions or outages.

Scenario Breakdown

The NEESgrid scenario consists of the following steps:

(1.) Create experiments and the simulation.

(2.) Create an experiment coordinator.

(3.) The coordinator starts the experiments.

(4.) The coordinator retrieves experiment results.

(5.) The coordinator reads the results.

(6.) The coordinator aggregates the results

(7.) The results are read.

Resource-Oriented Model

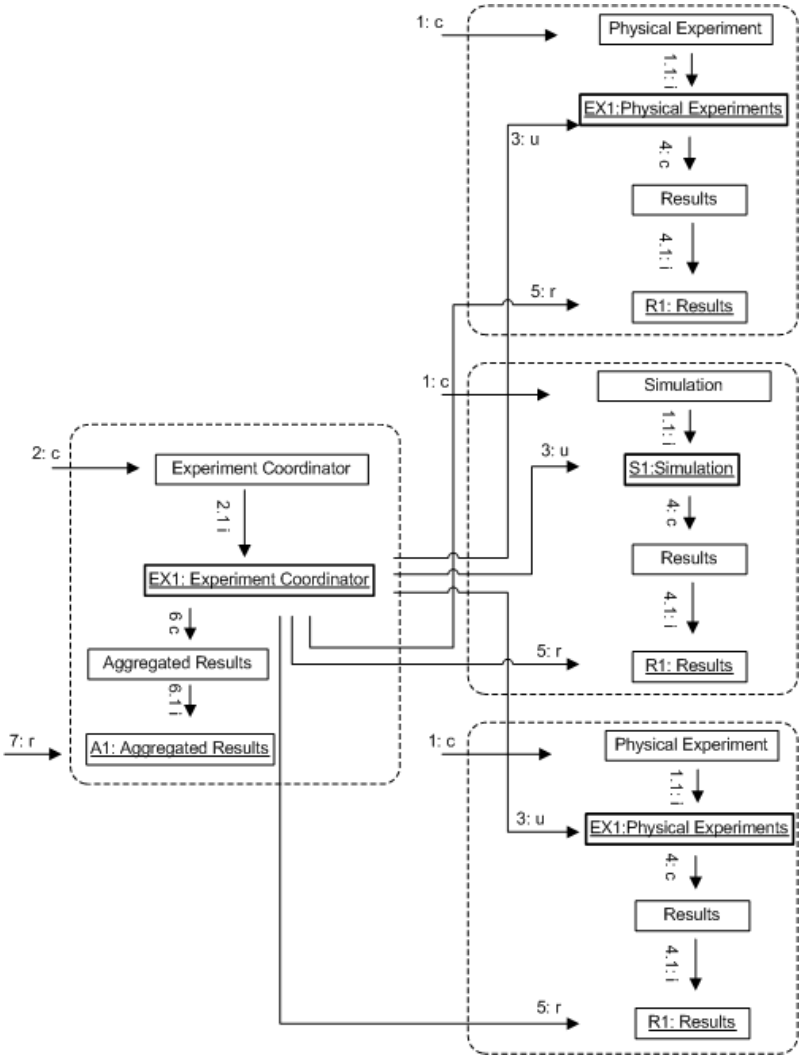


Figure 42 RO Model of G1

Due to the complex nature of the NEESgrid scenario and the limited space, structural links between resources were not modelled.

G2: Distributed Aircraft Maintenance Environment

The DAME (Distributed Aircraft Maintenance Environment) project (Jackson *et al.*, 2003), is a Grid enabled system for aeroengine fault diagnosis and prognosis. The aim of the project is to use Grid technology to manage and analyse the vast amounts of data to diagnose existing anomalies and predict potential problems in aircraft engines. (Jackson *et al.*, 2005) state the challenges for DAME, which are: the huge amount of data captured by the monitoring tool; the need for advanced pattern matching and data mining of the captured and historical data; the requirement of collaboration from diverse actors, and the heterogeneity and distributiveness of the data assets and tools.

Work on DAME was further researched in BROADEN (Business Resource Optimisation for Aftermarket and Design Engineering on Networks) (Jackson *et*

al., 2006) which investigated the use of SOA techniques to achieve their goals. The main usage scenarios for DAME are:

There is a QUICK monitoring service installed on the aircraft. This service captures the engine's monitoring data. QUICK can produce up to 1 Gigabyte of data for each engine. An aircraft can have two or more engines; this can scale to many Terabytes each year, for a fleet. Downloading and storing this amount of data efficiently requires a huge number of distributed repositories at different airports and these repositories must be available for the health monitoring of the engines. DAME's Engine Data Service is responsible for the downloading and storage of that data. The scenario mentioned in (Austin et al., 2005) illustrates the challenge.

> *"Heathrow, with its two runways, is authorized to handle a maximum of 36 landings per hour. Let us assume that on average half of the aircraft landing at Heathrow have four engines and the remaining half have two engines. In future, if each engine downloads around 1 GB of data per flight, the system at Heathrow must be capable of dealing with a typical throughput of around 100 GB of raw engine data per hour, all of which must be processed and stored. The data storage requirement alone for an operational day is, therefore, around 1 TB, with subsequent processing generating yet more data."*

Due to the vast amounts of data, the choice for DAME was to be highly distributed, having the airports as the units of distribution. The monitoring data from an aeroplane arriving at an airport is stored at that airport. Therefore, the search queries are distributed across airport nodes, where each node deals with the data it stores. This means that data relating to one engine is found in the different airports it landed in. To make DAME work, each airport node has a data repository, pattern matching service, and a data catalogue.

An engine specialist wants to analyse a particular engine's data. The specialist provides the engine's identifiers; the system submits it to a global catalogue, which returns a handle to the data in the repository and also provides access to a pattern matching control (PMC) service, which can distribute the search process across different nodes. The specialist searches for a feature in the engine data; the PMC becomes the master node and distributes the query to the other nodes. The search is performed in parallel; the PMC collects the results and returns them to the specialist.

The requirements are similar to G1. However, it differs in the following issues:

<u>Infrastructural and Functional Requirements</u>

1. Clients and Servers are controlled and managed by the same entity.
   Although DAME is implemented on a grid infrastructure, all the different components belong to the same entity.

2. Service Brokering

   There is a service broker which forwards services to different machines (servers/nodes), in this scenario, the PMC.

Non-functional Requirements

1. Support of sending and receiving large volumes of data

   Large volumes of data are being transferred between different services requiring reliable, efficient, and secure transfer.

Scenario Breakdown

(1.) Downloading the engine monitoring data.

(2.) Copying it to the nodes.

(3.) A client reads the Global Catalogue.

(4.) Sends a query to the node, the node distributes the query.

   (4.1) The query is run on the data.

   (4.2) Reads the results.

   (4.3) Aggregates the results.

(5.) The client reads the results.

Resource-Oriented Model

Appendix B

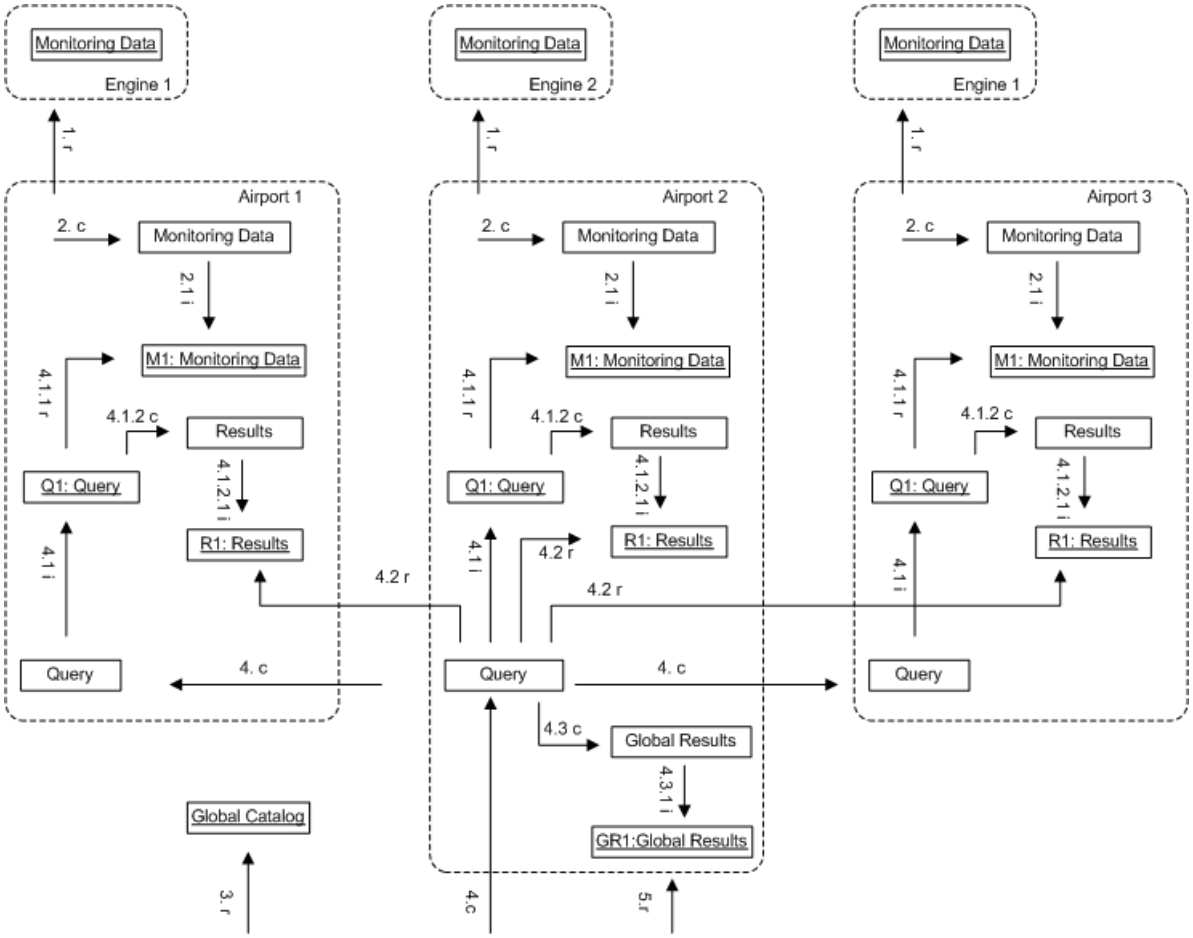


Figure 43 RO Model of G2

G3: Virtual Screening with Desktop Grids

Entropia, (Chien *et al.*, 2003) is an architecture for desktop grids. Desktop grids utilise the idle commodity computing resources (desktops) to perform highly distributed and computing intensive tasks. A binary virtual machine is installed on each desktop. These communicate with a job manager and resource scheduler to receive jobs, execute them, and return results. Desktop grids are effective when there is high need for parallel processing power and there is no need for communication between nodes during processing or the communication is minimal. (Chien *et al.*, 2003) describe "Virtual Screening" as one of the scenarios that make use of desktop grids:

In virtual screening, for drug discovery, a vast number of potential drug molecules are tested, ranging from hundreds of thousands to millions. The aim is to discover if these drugs affect the activity of a studied protein. Testing involves a process called docking that assesses the binding affinity of the test molecule to a specific place on a protein. Each potential molecule can be evaluated independently making the process suitable for desktop grids. The results are binding scores.

So a scenario based on that would be: an end-user submits a computation to the Job Manager, for example evaluating 50000 potential molecules. The Job Manager divides the computation into independent subjobs: in this scenario evaluating every five molecules together results in 10000 subjobs. The subjobs are submitted to the Subjob Scheduler. Any available resources are periodically reported to the Node Manager that informs the Subjob Scheduler. Results of the subjobs are sent to the Job Manager then handed back to the end-user.

Infrastructural and Functional Requirements

1. Virtual Machines installed on clients/participants
   For the desktop grid to work, virtual machines need to be installed on the nodes or desktops forming the grid computational resources.
2. Job Management
   Managing breaking down the jobs into independent sub-jobs that are assigned to nodes, then assembling the results and returning them to the client.
3. Job Scheduling
   The scheduling involves having knowledge of the numbers and sizes of tasks/jobs and the availability of resources. The VMs on the nodes inform the scheduler of the availability.
4. There are three entities in this scenario
   Desktop grid service provider: in this scenario Entropia;
   Nodes/participants: the desktops, which become grid resources after installing the VMs;
   Client: who has a computationally intensive task to run.

Non-functional Requirements

1. Security

In addition to the security issues mentioned in G1, another security measure is unobtrusiveness, meaning that the virtual machines and any jobs running on them do not harm or access unauthorised data or applications on the nodes they are executed on.

2. Tolerance of failure
3. In this scenario, tasks are being submitted to desktops, which are volatile, and it is likely that they could be switched off or cut off the network.

Scenario Breakdown

(1.) Create VMs on nodes.
(2.) Create the job.
(3.) Submit the job to the Job Manager.
(4.) The Job Manager splits the job into subjobs.
(5.) The Job Scheduler reads the subjobs.
(6.) The Job Scheduler sends them to the nodes.
(7.) The subjobs have results.

Appendix B

(8.) The Job Manager reads the results.

(9.) Aggregates the results.

(10.) The client reads the results.

Resource-Oriented Model

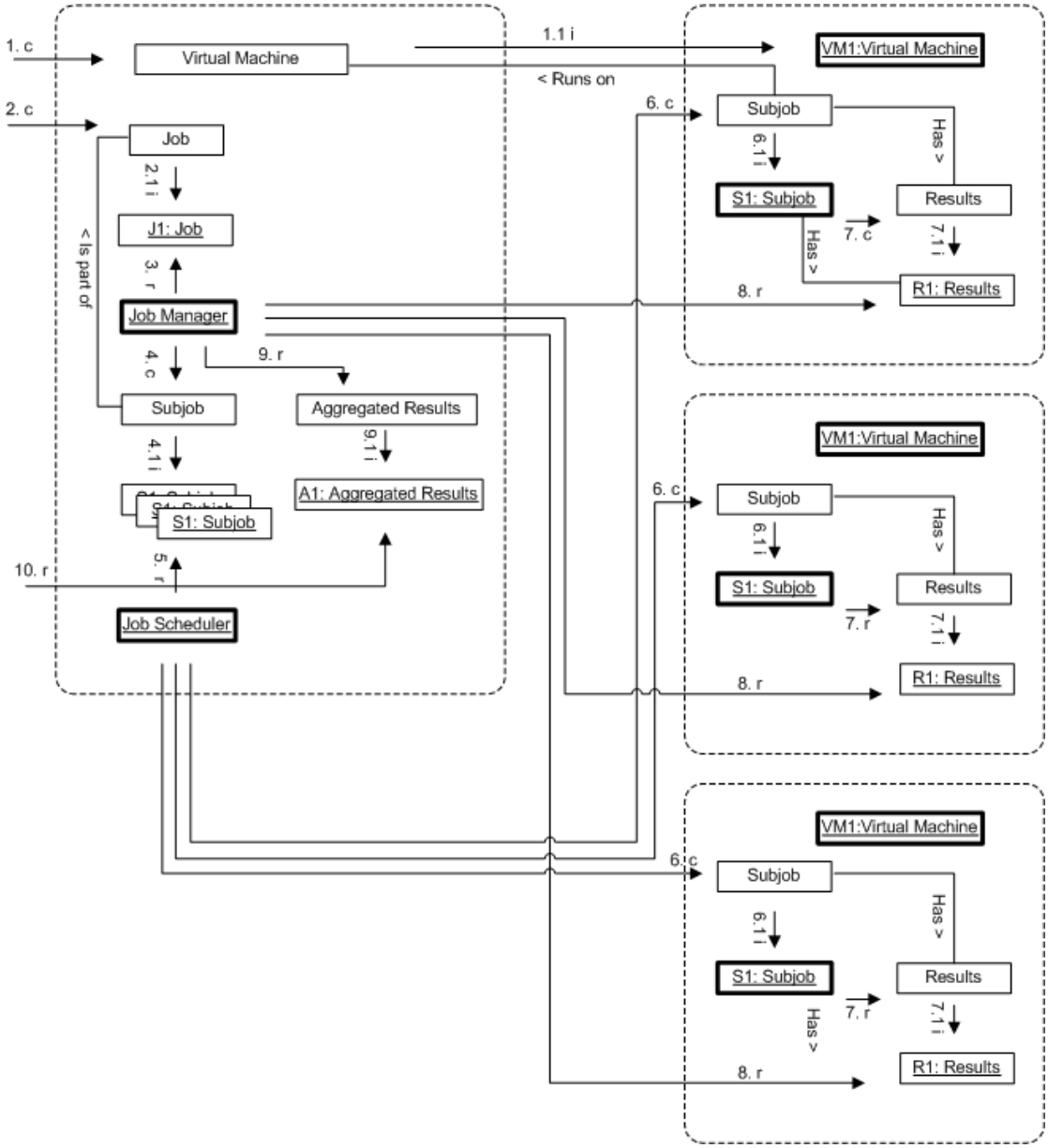


Figure 44 RO Model of G3

G4: CombeChem testbed on the Grid

The CombeChem project (Frey *et al.*, 2003) developed a testbed to combine structure data sources and property data sources, using the grid technologies to create a knowledge-sharing environment. The grid infrastructure enriches laboratory devices and supports provenance and automation techniques.

As part of the CombeChem project, Smart Lab was developed. It is intended to aid chemists during the different stages of an experiment, i.e. planning the experiment, performing the experiment, and analysing the results. The following scenario of using Smart Lab is built upon the description of the Smart Lab in (Taylor *et al.*, 2006).

A chemist uses the tablet PC to plan an experiment, gets it authorised by his/her supervisor. After the plan is authorised, the chemist follows it through to perform the experiment; during the experiment the chemist can observe and make notes that will be stored with the experimental process. Moreover, sensors and devices in the lab will store observations related to the experiment while it is being executed. After the experiment is performed, results are recorded.

The requirements are identical to scenario G1 and G2. However, there are others:

Infrastructural and Functional Requirements

1. Workflow support

The different processes that are executed can be coordinated and saved as workflows, so new workflows can be generated from them by changing processes or parameters.

2. Provenance Maintenance

The workflows provide means to link results to the steps they were generated from, thus providing a trial record and a method to reproduce the results.

Scenario Breakdown

(1.) The chemist creates the plan.

(2.) The chemist creates the experiment process that is based on the plan.

(3.) The process is updated by sensors and the chemist's observations.

(4.) Chemist can retrieve the process containing all the information about the process.

Resource-Oriented Model

Appendix B

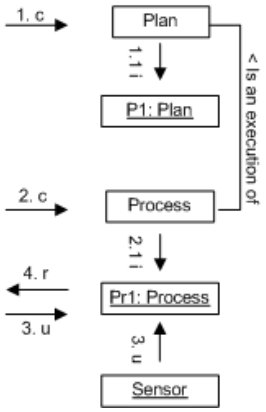


Figure 45 RO Model of G4

Table 26 Interaction requirements across scenarios

	Mutability		Atomicity	Synchronisation		Plurality		Roles
	Info Retrieval	Updating	Conversa-tional	Polling	Notifica-tion	Collection	Filtered Collection	
Mashups								
M1: Yahoo Pipes (Donnelly, 2010)	3	3	Y	0	0	1	1	0
M2: MashMaker (Ennals and Garofalakis, 2007)	1	6	Y	1	0	1	0	0
M3: Layered Mashup Architecture (Biornstad and Pautasso, 2009)	7	4	Y	1	1	2	1	0
M4: Mashup Fabric Customer Info (Jhingran, 2006)	1	0	N	1	0	0	0	0
Total	12	13	3	3	1	4	2	0
Enterprise Services								
E1: SSPD (City University) (City University, 2008)	1	4	Y	0	0	0	0	1
E2:MLE (City University) (City University, 2008)	1	1	Y	0	1	0	0	0
E3: BT.com (Integrating BT's OSS) (Calladine, 2004)	2	1	N	0	0	0	0	1
E4: SCORe (Integrating BT's OSS) (Calladine, 2004)	2	0	N	0	0	1	0	0
Total	6	6	2	0	1	1	0	2
B2B								
B1: Reverse Auctioning (Decker and Weske, 2007)	2	6	Y	0	0	1	0	1
B2: Telecommunications Wholesaler (Zimmermann *et al.*, 2005)	2	9	Y	1	0	1	0	0
B3: E-Procurement (Brodie, 2000)	2	3	Y	1	0	1	1	0
B4: Supply Chain Management (Preist *et al.*, 2005)	2	7	Y	0	0	2	0	0
Total	8	25	4	2	0	5	1	1
Cloud Computing								
C1: NYT Times Machine (Klems *et al.*, 2008)	1	5	Y	3	0	2	0	0
C2: MLB Website Chat System (Klems *et al.*, 2008)	0	6	Y	3	0	0	0	0
C3: Colorado State University (Herrick, 2009)	0	3	Y	1	0	1	0	0
C4: LingoSpot [22]	1	4	Y	2	0	0	0	0
Total	2	18	4	9	0	3	0	0
Grid Computing								

[22] **Google App Engine, App Engine Developer Profiles, http://code.google.com/appengine/casestudies.html**

G1: NEESGrid (Pearlman *et al.*, 2004)	2	7	Y	0	1	2	0	1
G2: Dist. Aircraft Maintenance Env. (Jackson *et al.*, 2005)	5	6	Y	4	0	2	1	0
G3: Virtual Screening on Desktop Grids(Chien *et al.*, 2003)	4	9	Y	4	2	2	1	1
G4: CombeChem testbed on the Grid (Frey *et al.*, 2006)	0	5	Y	4	1	0	0	1
Total	11	27	4	12	4	6	2	3
Total	**39**	**89**	**17**	**26**	**6**	**19**	**5**	**6**

Appendix C: Mappings to SPARQL Queries

Appendix C

Class

URI Pattern

```
/AClass
```

Graph Pattern

```
?x   a    AClass;
?x   ?y   ?z.
```

And in the case of a specific class, Book this would be:

URI

```
http://bookstore.com/Book
```

RDF Graph

```
<http://bookstore.com/Book/DBSys>  a :Book;
       :isbn    "0123735564"^^xsd:string;
       :title   "Database Systems"^^xsd:string;
       :author  <http://bookstore.com/Person/JSmith>.
<http://bookstore.com/Book/SemWeb>  a     :Book;
       :isbn    "2266776375"^^xsd:string;
       :title   "Semantic Web"^^xsd:string;
       :author  <http://bookstore.com/Person/TBL>.
```

Table 27 HTTP methods as SPARQL queries for the class book

GET	
Description	Retrieves information about all books

Corresponding SPARQL Query	`CONSTRUCT { ?s ?p ?o}`
	`WHERE {  ?s a  <http://bookstore.com/Book>;`
	`  ?p ?o }`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "0123735564"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`
	`<http://bookstore.com/Book/SemWeb>  a  :Book;`
	`  :isbn  "2266776375"^^xsd:string;`
	`  :title  "Semantic Web"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/TBL>.`

Explanation	The triples returned by the CONSTRUCT query are formatted according to the graph pattern associated with the class resource type. Every individual of class book is returned, with triples where this individual is the subject.
PUT	
Description	Creates a named individual

Payload	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "0123735564"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`

Corresponding SPARQL Query	`INSERT`
	`{GRAPH <Server> { ?s ?p ?o}}`
	`WHERE {GRAPH <Payload>  { ?s ?p ?o}}`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "1334340005"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`

Explanation	The INSERT operation adds triples to the server, these triples will match triples in the payload, in this case it is a book individual together with associated triples. Of course there needs to be checks on the payload to ensure it adheres to the structure accepted by this resource type, which is denoted by the associated graph pattern, and that the subject of these triples is a named individual.

POST	
Description	Creates an individual

Payload	`_:a232324 a :Book;`
	`:isbn "0123735564"^^xsd:string;`
	`:title "Database Systems"^^xsd:string;`
	`:author <http://bookstore.com/Person/JSmith>.`

Corresponding SPARQL Query	`INSERT {GRAPH <Server>`
	`{ ?s ?p ?o}}`
	`WHERE {GRAPH <Payload>`
	`{ ?s ?p ?o}}`

Result	`<http://bookstore.com/Book/DBSys> a :Book;`
	`:isbn "1334340005"^^xsd:string;`
	`:title "Database Systems"^^xsd:string;`
	`:author <http://bookstore.com/Person/JSmith>.`

Explanation	Similar to the PUT method above, the only difference is that the subject of these triples in the payload is a blank node which also needs to be checked, and replaced by the server with a named individual of class book in this case.
DELETE	
Description	Delete all individuals of this class

Corresponding SPARQL Query	`DELETE {`
	`GRAPH <Server> {`
	`?s a <http://bookstore.com/Book>;`
	`?p ?o. }}`
	`WHERE`
	`{ ?s a <http://bookstore.com/Book>;`
	`?p ?o.}`

Explanation	The DELETE operation would delete all the individuals of this class, and their associated properties: i.e. triples which have these individuals as their subjects.

Appendix C

Individual

URI Pattern
```
/AClass/Individual
```
Graph Pattern
```
Individual a     AClass;
Individual ?x    ?y.
```

And in the case of a specific book, DBSys, this would be:

URI
```
http://bookstore.com/Book/DBSys
```
RDF Graph
```
<http://bookstore.com/Book/DBSys>  a :Book;
      :isbn      "0123735564"^^xsd:string;
      :title     "Database Systems"^^xsd:string;
      :author    <http://bookstore.com/Person/JSmith>.
```

Table 28 HTTP methods as SPARQL queries for a book individual

GET	
Description	Retrieves information about DBSys at this URI http://bookstore.com/Book/DBSys

Corresponding SPARQL Query	`CONSTRUCT {`
	`  <http://bookstore.com/Book/DBSys> ?p ?o }`
	`WHERE {`
	`  <http://bookstore.com/Book/DBSys> ?p ?o }`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "0123735564"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`

Explanation	The CONSTRCUT query returns triples in the format specified by the graph pattern associated with the individual resource type (see Table 10), which returns the values of the associated triples.
PUT	
Description	Updating the ISBN of the book at this URI http://bookstore.com/Book/DBSys
Payload	`<http://bookstore.com/Book/DBSys>` `  :isbn  "1334340005"^^xsd:string.`

Corresponding SPARQL Query	`WITH <Server>`
	`DELETE`
	`  { <http://bookstore.com/Book/DBSys> ?p ?oOld}`
	`INSERT`
	`  { <http://bookstore.com/Book/DBSys> ?p ?oNew}`
	`WHERE`
	`  { GRAPH  <Payload> {`
	`  <http://bookstore.com/Book/DBSys>  ?p  ?oNew }`
	`  GRAPH  <Server>  {`
	`  <http://bookstore.com/Book/DBSys>  ?p  ?oOld }}`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "1334340005"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`

Explanation	To update an individual, this is mapped to a DELETE/INSERT operation, the payload contains the triples that specify the properties that will be updated and their new values. The DELETE/INSERT operation deletes from the server the triples that match the pattern : Individual ?p ?old But since there is a WHERE clause, this pattern has to also match the triples provided in the payload. Therefore only triples containing properties provided in the payload will be affected in the server, and replaced by the triples provided in the payload which is the effect of the INSERT clause.

DELETE	
Description	Deletes the individual and associated properties.

Corresponding SPARQL Query	DELETE {
	GRAPH <Server> {
	<http://bookstore.com/Book/DBSys> ?p ?o. }}
	WHERE
	{ <http://bookstore.com/Book/DBSys> ?p ?o. }

Explanation	The triple <http://bookstore.com/Book/DBSys> ?p ?o. Matches the individual and its properties at the server, and the DELETE operation removes those triples.

Appendix C

Property

The Individual's property URI Pattern

```
/AClass/Individual/Property
```

Corresponding Graph Pattern for an Object Property

```
Individual      Property   ?x
?x              ?p         ?o
```

This applies to GET. However, for PUT and DELETE the corresponding Graph Pattern is either the one above or

```
Individual      Property   ?x
```

meaning, when the author of a book is deleted, only the link between the author and this specific article is deleted, the author's information is not, the decision whether the value is deleted is left to the implementation. Moreover the latter pattern is also the corresponding graph pattern for Data Properties.

Table 29 HTTP methods as SPARQL queries for a book's author

GET	
Description	Retrieving information about the author of the book DBSys at this URI `http://bookstore.com/Book/DBSys/author`
Corresponding SPARQL Query	`CONSTRUCT { <http://bookstore.com/Book/DBSys> :author ?x. ?x ?p ?o. }` `WHERE {<http://bookstore.com/Book/DBSys> :author ?x. ?x ?p ?o. }`
Result	`<http://bookstore.com/Book/DBSys> :author <http://bookstore.com/Person/JSmith>.` `<http://bookstore.com/Person/JSmith> a :Person; :name "John Smith"^^xsd:string.`
Explanation	The CONSTRUCT query returns a triple containing the author as the subject, replacing the variable ?x, moreover it returns properties and their values where the author is the subject. If this was a data property instead of an object property the ?x ?p ?o triple pattern will be omitted.
PUT	
Description	Changing the author of the book at this URI `http://bookstore.com/Book/DBSys/author`
Payload	`<http://bookstore.com/Book/DBSys> a :Book; :author <http://bookstore.com/Person/TBL>.`

Corresponding SPARQL Query	`WITH <Server>`
	`DELETE`
	`{ <http://bookstore.com/Book/DBSys> :author ?oOld}`
	`INSERT`
	`{ <http://bookstore.com/Book/DBSys> :author ?oNew}`
	`WHERE`
	`{ GRAPH <Payload> { <http://bookstore.com/Book/DBSys> :author ?oNew }`
	`GRAPH <Server> { <http://bookstore.com/Book/DBSys> :author ?oOld }}`

Result	`<http://bookstore.com/Book/DBSys> a :Book; :isbn "1334340005"^^xsd:string; :title "Database Systems"^^xsd:string; :author <http://bookstore.com/Person/TBL>.` If the server had multiple values for author, they would all be deleted and replaced with author(s) in the payload.
Explanation	The PUT method updates the value of the author. The DELETE/INSERT operation replaces triples, so to update the author in this case, the triples in the payload replace the ones in the server. This is also the case for data properties. However assuming that the property is a dependent one, for it to be completely replaced, not only for the triple that connects it to the book, the following triple pattern would be added to the DELETE clause, and the server clause: ?oOld ?p ?o, and this triple pattern ?oNew ?p ?o to the INSERT clause and the payload clause, hence replacing triples associated with the replaced property value.

DELETE	
Description	Deletes the author of the book at this URI

Corresponding SPARQL Query	`DELETE {`
	`  GRAPH  <Server> {`
	`  <http://bookstore.com/Book/DBSys>  :author  ?x. }}`
	`WHERE`
	`  { <http://bookstore.com/Book/DBSys>  :author  ?x. }`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "1334340005"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string.`

Explanation	As explained above the table, this depends on the implementation, either the association between the book an author is deleted, as shown in the operation above, or the triples who have the author as the subject are deleted too, and in that case the triple pattern ?x ?p ?o, would exist in both clauses.

Appendix C

Filtered Individuals

URI Pattern

```
/AClass?DataProperty={value1}&ObjectProperty={value2}
```

Corresponding Graph Pattern

```
?x        a          AClass
?x        DataProperty    value1
?x        ObjectProperty  value2
?x        ?y         ?z
```

Table 30 HTTP methods as SPARQL queries a book with specified properties

GET	

Description	Retrieves information about all books who have "Database Systems" as their title and JSmith as their author.
	`/Book?title="Database Systems"`
	`&author="http://bookstore.com/Person/JSmith"`

Corresponding SPARQL Query	`CONSTRUCT {`
	`  ?s ?p ?o.`
	`}`
	`WHERE {`
	`  ?s  :title "Database Systems".`
	`  ?s  :author <http://bookstore.com/Person/JSmith>.`
	`  ?s  ?p ?o.}`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn  "0123735564"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`

Explanation	This is similar to the GET method in Table 27, however the CONSTRUCT query differs in the WHERE clause as it specifies values for given properties.
PUT	

Description	Updates individuals who have "Database Systems" as their title and JSmith as their authors, by changing their ISBN.
	`/Book?title="Database Systems"`
	`&author="http://bookstore.com/Person/JSmith"`

Payload	`_:b  a :Book;` `  :isbn "1234567890"^^xsd:string.`

Corresponding SPARQL Query	`WITH <Server>`
	`DELETE`
	`  { ?s  ?p  ?oOld}`
	`INSERT`
	`  { ?s  ?p  ?oNew}`
	`WHERE`
	`  { GRAPH  <Payload> { ?x  ?p ?oNew }`
	`  GRAPH  <Server>  {`
	`  ?s  ?p ?oOld.`
	`  ?s  :title "Database Systems".`
	`  ?s  :author  <http://bookstore.com/Person/JSmith>. }}`

Result	`<http://bookstore.com/Book/DBSys>  a :Book;`
	`  :isbn "1234567890"^^xsd:string;`
	`  :title  "Database Systems"^^xsd:string;`
	`  :author  <http://bookstore.com/Person/JSmith>.`

Explanation	The DELETE/INSERT operation above means: for book individuals, (?s) which match the two triple patterns i.e. have the title "Database Systems" and the author JSmith, delete the old triples at the server, replace them with new ones, where the subject would remain the same as it was (i.e. the same book (?s)) but the replaced properties matches the ones provided in the payload (?p).
POST	

Description	Creates an individual which has "Database Systems" as its title and JSmith as its author.
	`/Book?title="Database Systems"`
	`&author=<http://bookstore.com/Person/JSmith>`

Corresponding SPARQL Query	`INSERT DATA`
	`{GRAPH <Server>`
	`{ <http://bookstore.com/Book/NewBook> a :Book;`
	`:title "Database Systems";`
	`:author <http://bookstore.com/Person/JSmith>.}}`
	The book URI is provided by the server for the newly created book.

Result	`<http://bookstore.com/Book/DBSys> a :Book;`
	`:title "Database Systems"^^xsd:string;`
	`:author <http://bookstore.com/Person/JSmith>.`

Explanation	Similar to the POST operation in Table 27, however the property values are specified in the query string rather than the payload.
DELETE	
Description	Delete all individuals of this class who have “Database Systems” as their title and JSmith as their author.

Corresponding SPARQL Query	`DELETE {`
	`GRAPH <Server> {`
	`?s ?p ?o. }}`
	`WHERE`
	`{ ?s ?p ?o;`
	`:title "Database Systems";`
	`:author <http://bookstore.com/Person/JSmith>.}`

Explanation	Also similar to the DELETE operation in Table 27, however the graph pattern specifies individuals who have these values for the specified properties.

Appendix C

Properties of Filtered Individuals

URI Pattern

```
/AClass/TheProperty?Property1={valueA}&Property2={valueB}
```

Corresponding Graph pattern

```
?x        a              AClass
?x        TheProperty    ?y
?x        Property1      valueA
?x        Property2      valueB
?y        ?p             ?o
```

Table 31 HTTP methods as SPARQL queries for properties of filtered individuals

GET	
Description	Retrieves information about authors of books with the title "Database Systems". `Book/author?title="Database Systems"`
Corresponding SPARQL Query	`CONSTRUCT { ?b :author ?x. ?x ?p ?o } WHERE { ?b :title "Database Systems". ?b :author ?x. ?x ?p ?o }`
Result	`<http://bookstore.com/Book/DBSys> :author : <http://bookstore.com/Person/JSmith>. <http://bookstore.com/Person/JSmith> :name "John Smith"^^xsd:string.`
Explanation	The CONSTRUCT query returns triples about the author (?x) of the book, which has been specified to have the title "Database Systems" in the WHERE clause.
PUT	
Description	Updates the authors of books who have "Database Systems" as their title `/Book/author?title="Database Systems"`
Payload	`?x a :Book; :author <http://bookstore.com/Person/TBL>.`
Corresponding SPARQL Query	`WITH <Server> DELETE { ?book :author ?oldAuthor. ?oldAuthor ?oldp ?oldo. } INSERT { ?book :author ?newAuthor. ?newAuthor ?newp ?newo. } WHERE { GRAPH <Payload> { ?somebook :author ?newAuthor. ?newAuthor ?newp ?newo. } GRAPH <Server> { ?book :author ?oldAuthor. ?oldAuthor ?oldp ?oldo. ?book :title "Database Systems".}}`
Result	`<http://bookstore.com/Book/DBSys> a :Book; :title "Database Systems"^^xsd:string; :author <http://bookstore.com/Person/TBL>.`
Explanation	This is similar to the PUT operation in Filtered individuals in Table 30, the difference however, is in that operation, the replaced properties can be many, in this case it is specified :author.

DELETE	
Description	Delete the author property of books which have “Database Systems” as their title.

Corresponding SPARQL Query	`DELETE {`
	`  GRAPH  <Server> {`
	`  ?book  :author  ?author.`
	`  ?author ?p ?o. }}`
	`WHERE {  ?book  :author ?author.`
	`  ?author ?p ?o.`
	`  ?book  :title "Database Systems".}`

Explanation	The association between the author and the book, would be deleted, and also details of the author, for books who have “Database Systems” as their title.

Appendix D: DVD/MP3 Player OWL-S Service

Appendix D

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF  xmlns:owl       = "http://www.w3.org/2002/07/owl#"
xmlns:rdfs      = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf       = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:service   = "http://www.daml.org/services/owl-s/1.1/Service.owl#"
xmlns:process   = "http://www.daml.org/services/owl-s/1.1/Process.owl#"
xmlns:profile    = "http://www.daml.org/services/owl-s/1.1/Profile.owl#"
xmlns:grounding = "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"

xml:base  =
"http://127.0.0.1/services/1.1/dvdplayermp3player_price_service.owls">

<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://127.0.0.1/ontology/Service.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Process.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Profile.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Grounding.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/my_ontology.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/concept.owl" />
</owl:Ontology>

<service:Service rdf:ID="DVDPLAYERMP3PLAYER_PRICE_SERVICE">
<service:presents rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_PROFILE"/>
<service:describedBy rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_PROCESS"/>
<service:supports rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_GROUNDING"/>
</service:Service>

<profile:Profile rdf:ID="DVDPLAYERMP3PLAYER_PRICE_PROFILE">
<service:isPresentedBy rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_SERVICE"/>
<profile:serviceName xml:lang="en">
2For 1 Price service
</profile:serviceName>
<profile:textDescription xml:lang="en">
This service returns prices of a given pair MP3 Player brand and
DVD Player brand.
</profile:textDescription>
<profile:hasInput  rdf:resource="#_MP3PLAYER"/>
<profile:hasOutput rdf:resource="#_PRICE"/>
<profile:hasInput  rdf:resource="#_DVDPLAYER"/>

<profile:has_process rdf:resource="DVDPLAYERMP3PLAYER_PRICE_PROCESS" />
</profile:Profile>

<!--<process:ProcessModel rdf:ID="DVDPLAYERMP3PLAYER_PRICE_PROCESS_MODEL">
<service:describes rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_SERVICE"/>
<process:hasProcess rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_PROCESS"/>
</process:ProcessModel>-->

<process:AtomicProcess rdf:ID="DVDPLAYERMP3PLAYER_PRICE_PROCESS">
<service:describes rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_SERVICE"/>
<process:hasInput  rdf:resource="#_MP3PLAYER"/>
<process:hasOutput rdf:resource="#_PRICE"/>
<process:hasInput  rdf:resource="#_DVDPLAYER"/>
</process:AtomicProcess>

<process:Input rdf:ID="_MP3PLAYER">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://127.0.0.1/ontolo
gy/my_ontology.owl#MP3Player</process:parameterType>
<rdfs:label></rdfs:label>
</process:Input>

<process:Output  rdf:ID="_PRICE">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/ontology/concept.owl#Price</process:parameterType>
<rdfs:label></rdfs:label>
</process:Output>
```

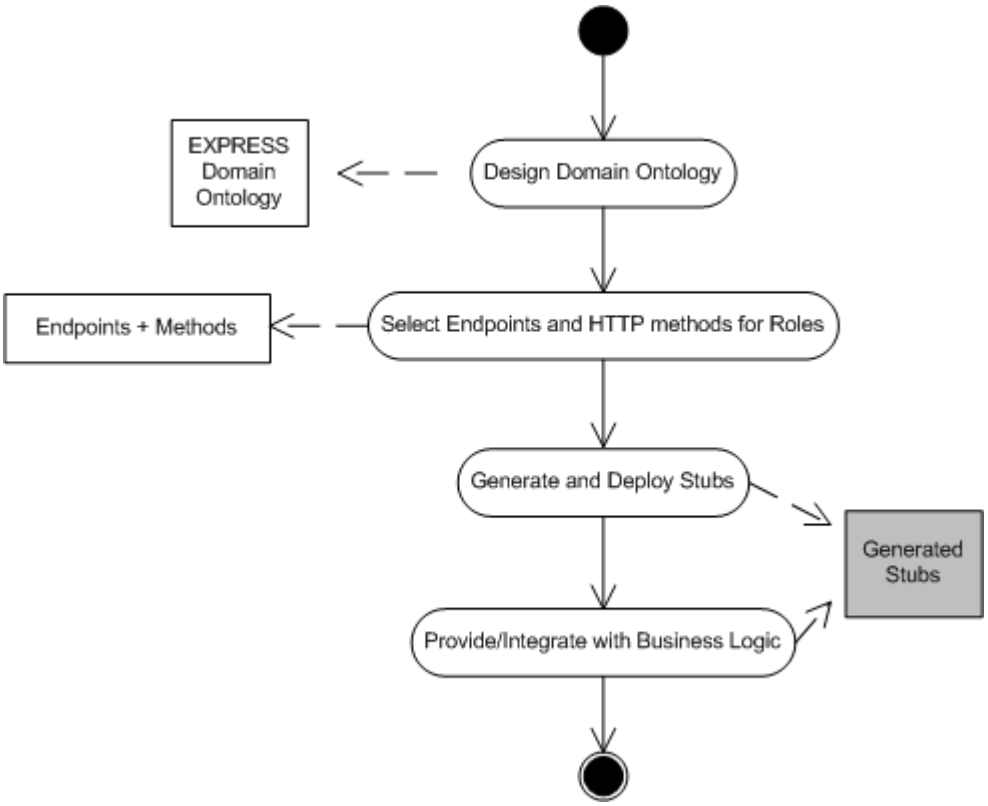
```
<process:Input rdf:ID="_DVDPLAYER">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://127.0.0.1/ontolo
gy/my_ontology.owl#DVDPlayer</process:parameterType>
<rdfs:label></rdfs:label>
</process:Input>

<grounding:WsdlGrounding rdf:ID="DVDPLAYERMP3PLAYER_PRICE_GROUNDING">
<service:supportedBy rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_SERVICE"/>
<grounding:hasAtomicProcessGrounding>
      <grounding:WsdlAtomicProcessGrounding
rdf:ID="DVDPLAYERMP3PLAYER_PRICE_AtomicProcessGrounding"/>
    </grounding:hasAtomicProcessGrounding>
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding
rdf:about="#DVDPLAYERMP3PLAYER_PRICE_AtomicProcessGrounding">
    <grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1.wsdl</grounding:wsdlDocument>
    <grounding:owlsProcess rdf:resource="#DVDPLAYERMP3PLAYER_PRICE_PROCESS"/>
    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
          <grounding:operation rdf:datatype=
               "http://www.w3.org/2001/XMLSchema#anyURI">
               http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/get_PRICE
          </grounding:operation>
        <grounding:portType
          rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
          http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/Dvdplayermp3playerPri
          ceSoap
        </grounding:portType>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
    <grounding:wsdlInputMessage
     rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
     http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/get_PRICERequest
     </grounding:wsdlInputMessage>
    <grounding:wsdlOutputMessage
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
    http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/get_PRICEResponse
    </grounding:wsdlOutputMessage>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource="#_MP3PLAYER"/>
        <grounding:wsdlMessagePart rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI">
        http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/_MP3PLAYER
       </grounding:wsdlMessagePart>
     <grounding:xsltTransformationString>None(XSL)
     </grounding:xsltTransformationString>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource="#_DVDPLAYER"/>
        <grounding:wsdlMessagePart
           rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
           http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/_DVDPLAYER
        </grounding:wsdlMessagePart>
        <grounding:xsltTransformationString>None(XSL)
        </grounding:xsltTransformationString>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
        <grounding:owlsParameter rdf:resource="#_PRICE"/>
        <grounding:wsdlMessagePart
```

```
          rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
          http://127.0.0.1/wsdl/Dvdplayermp3playerPrice1/_PRICE
          </grounding:wsdlMessagePart>
        <grounding:xsltTransformationString>None (XSL)
       </grounding:xsltTransformationString>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Appendix E: Expert Review Materials

PART ONE

EXPRESS



EXPRESS – Service Design and Deployment

1. A developer provides an ontology representing entities in the Web service interface they would like to expose.
2. The EXPRESS deployment engine extracts resources from the OWL file and assigns URIs.
3. The developer chooses URIs for endpoints and permitted HTTP methods and optionally roles and access control.
4. Stubs are connected to existing business logic, coded, or the code is generated.

The semantics of interacting with the endpoints is implicitly expressed by two things:
1. The resource type the endpoint represents: this is indicated by way EXPRESS provides endpoints.
2. The HTTP method.

Resource Type	URI Template	Graph Pattern		
Class	/Class	?x	a	Class
Individual	/Class/Individual	Individual	?x	?y
Property	/Class/Individual/Property	Individual ?x	Property ?p	?x ?o
Filter Individuals	/Class?Property={a}	?x ?x a1	a Property ?p	Class a1 ?o
Filtered Individuals' Properties	/Class/Property1?Property2={b}	?x ?x ?x ?y	a Property1 Property2 ?p	Class ?y b ?o

Bookstore EXPRESS Ontology

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix dc: <http://purl.org/dc/terms/>.
@prefix : <http://bookstore.com/EXPRESS/>.
<http://bookstore.com/EXPRESS> a owl:Ontology.
:Book   a    owl:Class;
        owl:equivalentClass  dbpedia:Book.
:title  a    owl:DatatypeProperty;
        owl:equivalentProperty    dc:title;
        rdfs:domain     :Book;
        rdfs:range      xsd:string.
:author a    owl:DatatypeProperty;
        owl:equivalentProperty dc:author;
        rdfs:domain      :Book;
        rdfs:range  xsd:string.
:isbn   a    owl:DatatypeProperty;
        owl:equivalentProperty    dbpedia:isbn;
        rdfs:domain     :Book;
        rdfs:range  xsd:string.
:Order   a   owl:Class.
:containsItem a owl:ObjectProperty;
        rdfs:domain :Order;
        rdfs:range :Book.
```

Some possible endpoints

```
GET    http://bookstore.com/EXPRESS/Book/DBSys
PUT    http://bookstore.com/EXPRESS/Book/DBsys/title
DELETE http://bookstore.com/EXPRESS/Order/1231324
```

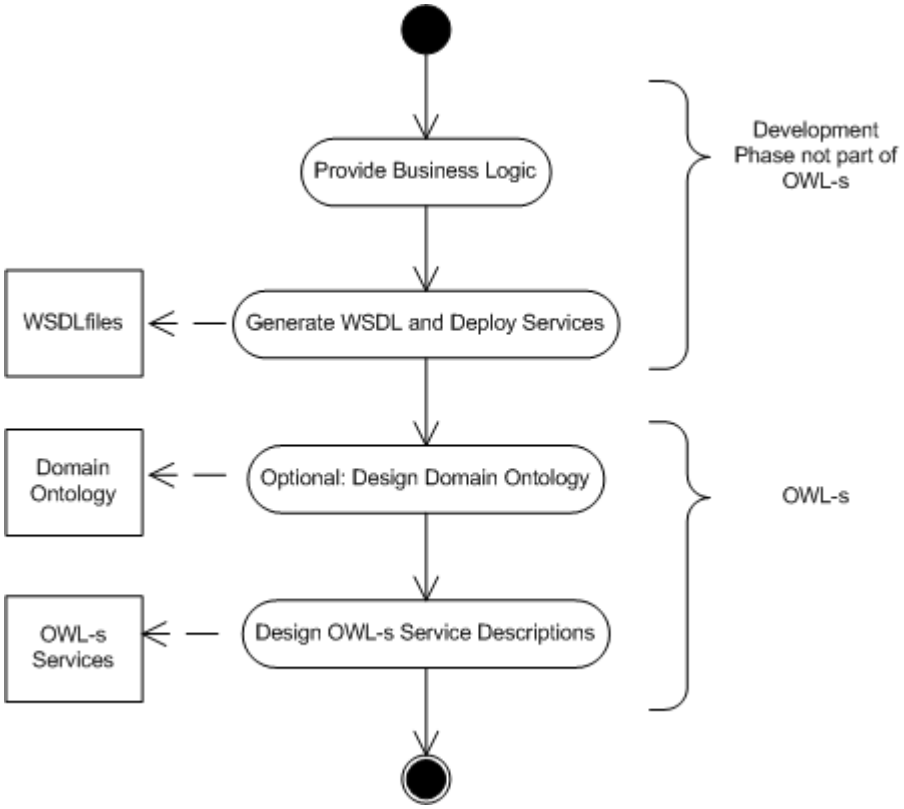
The Endpoints for this example

```
GET    http://bookstore.com/EXPRESS/Book?isbn={}
POST   http://bookstore.com/EXPRESS/Order?containsItem={}
```

EXPRESS – Interaction

Client	Server
GET /EXPRESS HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK Link: </EXPRESS/Book?isbn={}>; rel="GET" 
GET /EXPRESS/Books?isbn={0123735564} HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK Link: </EXPRESS/Order?containsItem={}>; rel="POST" @prefix xsd: <http://www.w3.org/2001/XMLSchema#>. @prefix : <http://bookstore.com/EXPRESS/>. <http://bookstore.com/EXPRESS/Book/Sem> a :Book; :isbn "0123735564"^^xsd:string; :title "Semantic Web for the Working Ontologist"^^xsd:string; :author "Allemang and Hendler"^^xsd:string.
POST /EXPRESS/Order?containsItem= http://bookstore.com/EXPRESS/Book/Sem HTTP/1.1 Host: bookstore.com	
	HTTP 201 Created Location: http://bookstore.com/EXPRESS/Order_1 @prefix : <http://bookstore.com/EXPRESS/>. :Order_1 a :Order; :containsItem <http://bookstore.com/EXPRESS/Book/Sem>

OWL-S


Provide Business Logic
Development Phase not part of OWL-s
WSDLfiles
Generate WSDL and Deploy Services
Domain Ontology
Optional: Design Domain Ontology
OWL-s
OWL-s Services
Design OWL-s Service Descriptions

Appendix E

OWL-S Service Design and Deployment

OWL-S does not involve the steps in deploying the Web service, as it comes after the development phase. However, because we are comparing it with EXPRESS and it is involved in the development phase, it is necessary to discuss the tasks OWL-S assumes are done.

Deploying a service can be done by generating a WSDL file from the business logic, which can be written either in Java, PHP, .NET. The underlying framework used also takes care of translating the exchange messages into SOAP.

The Bookstore WSDL files

1. Get a Book by its ISBN

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
xmlns="http://bookstore.com/wsdl/bookbyisbn.wsdl"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://bookstore.com/wsdl/bookbyisbn.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://bookstore.com/wsdl/bookbyisbn.wsdl"  name="book">
<wsdl:types>
       <xsd:schema targetNamespace="http://bookstore.com/wsdl/bookbyisbn.wsdl"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
           <xsd:complexType name="book">
                <xsd:sequence>
                     <xsd:element name="title" type="xsd:string"/>
                     <xsd:element name="author"type="xsd:string"/>
                          <xsd:element name="isbn"  type="xsd:string"/>
                </xsd:sequence>
           </xsd:complexType>
     </xsd:schema>
</wsdl:types>
<wsdl:message name="getBookByISBNRequest">
     <wsdl:part name="isbn" type="xsd:string">
     </wsdl:part>
</wsdl:message>

<wsdl:message name="getBookByISBNResponse">
     <wsdl:part name="book" type="tns:book">
     </wsdl:part>
</wsdl:message>
<wsdl:portType name="BookByISBNSoap">
     <wsdl:operation name="getBookByISBN">
          <wsdl:input message="tns:getBookByISBNRequest">
          </wsdl:input>
          <wsdl:output message="tns:getBookByISBNResponse">
          </wsdl:output>
     </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="BookByISBNSoapBinding" type="BookByISBNSoap">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="getBookByISBN">
          <wsdlsoap:operation soapAction="getBookByISBN"/>
          <wsdl:input>
          <wsdlsoap:body use="encoded" encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://bookstore.com/wsdl/bookbyisbn"/>
          </wsdl:input>
          <wsdl:output>
               <wsdlsoap:body use="encoded" encodingStyle=
                   "http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://bookstore.com/wsdl/bookbyisbn"/>
          </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="getBookByISBNService">
<wsdl:port name="BookByISBNSoap" binding="BookByISBNSoapBinding">
```

```
<wsdlsoap:address location="http://bookstore.com/BookService"/>
</wsdl:port>
</wsdl:service></wsdl:definitions>
```

2. Order a Book

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
xmlns="http://bookstore.com/wsdl/bookorder.wsdl"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://bookstore.com/wsdl/bookorder.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://bookstore.com/wsdl/bookorder.wsdl" name="bookorder">

<wsdl:types>
     <xsd:schema targetNamespace="http://bookstore.com/wsdl/bookorder.wsdl"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <xsd:complexType name="book">
               <xsd:sequence>
                    <xsd:element name="title" type="xsd:string"/>
                    <xsd:element name="author"type="xsd:string"/>
                    <xsd:element name="ISBN"  type="xsd:string"/>
               </xsd:sequence>
          </xsd:complexType>
          <xsd:complexType name="order">
               <xsd:element name="book" maxOccurs="unbounded"
type="bookType"/>
          </xsd:complexType>
     </xsd:schema>
</wsdl:types>
<wsdl:message name="BookOrderServiceRequest">
     <wsdl:part name="book" type="tns:book">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="BookOrderServiceResponse">
     <wsdl:part name="order" type="tns:order">
     </wsdl:part>
</wsdl:message>

<wsdl:portType name="BookOrderServiceSoap">
     <wsdl:operation name="BookOrderService">
          <wsdl:input message="tns:BookOrderServiceRequest">
          </wsdl:input>
          <wsdl:output message="tns:BookOrderServiceResponse">
          </wsdl:output>
     </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="BookOrderServiceSoapBinding" type="BookOrderServiceSoap">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="BookOrderService">
          <wsdlsoap:operation soapAction="BookOrderService"/>
          <wsdl:input>
                    <wsdlsoap:body use="encoded" encodingStyle=
                    "http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://bookstore.com/wsdl/bookorder"/>
          </wsdl:input>
          <wsdl:output>
               <wsdlsoap:body use="encoded" encodingStyle=
                    "http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://bookstore.com/wsdl/bookorder"/>
          </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="BookOrderService">
<wsdl:port name="BookOrderServiceSoap" binding="BookOrderServiceSoapBinding">
<wsdlsoap:address location="http://bookstore.com/BookService"/>
</wsdl:port>
</wsdl:service></wsdl:definitions>
```

Appendix E

OWL-S Semantic Description

The Bookstore OWL-S files

1. Get a book by its ISBN

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix service: <http://www.daml.org/services/OWL-S/1.1/Service.owl#>.
@prefix profile: <http://www.daml.org/services/OWL-S/1.1/Profile.owl#>.
@prefix process: <http://www.daml.org/services/OWL-S/1.1/Process.owl#>.
@prefix grounding: <http://www.daml.org/services/OWL-S/1.1/Grounding.owl#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix expr: <http://www.daml.org/services/OWL-
S/1.1/generic/Expression.owl#>.
@prefix swrlb: <http://www.w3.org/2003/11/swrlb#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix domOnt: <http://bookstore.com/DomainOntology#>.
@prefix groundingWSDL: <http://bookstore.com/wsdl/getBookByISBN.wsdl#>.
@prefix : <http://bookstore.com/owls/getBookByISBN.owls#>.

<http://bookstore.com/owls/getBookByISBN.owls> a owl:Ontology;
     owl:imports <http://bookstore.com/DomainOntology.owl>,
          <http://dbpedia.org/ontology/>,
          <http://www.daml.org/services/OWL-S/1.1/Grounding.owl>,
          <http://www.daml.org/services/OWL-S/1.1/Process.owl>,
          <http://www.daml.org/services/OWL-S/1.1/Profile.owl>,
          <http://www.daml.org/services/OWL-S/1.1/Service.owl>,
             <http://www.w3.org/2003/11/swrl>,
             <http://www.daml.org/services/OWL-S/1.1/generic/Expression.owl>.

:getBookByISBNService a service:Service;
     service:presents :getBookByISBNProfile;
     service:describedBy :getBookByISBNProcess;
     service:supports :getBookByISBNGrounding.
:getBookByISBNProfile a profile:Profile;
     profile:hasInput :ISBN;
     profile:hasOutput :Book;
     service:presentedBy :getBookByISBNService;
     profile:serviceName "getBookByISBN"^^xsd:string.
:getBookByISBNProcess a process:AtomicProcess;
     process:hasInput :ISBN;
     process:hasOutput :Book;
       process:hasPrecondition :ValidISBN;
     process:hasResult :BookhasISBN;
     service:describes :getBookByISBNService;
     rdfs:label "getBookByISBNProcess"^^xsd:string.
:ISBN a process:Input;
     process:parameterType
     "http://bookstore.com/DomainOntology#ISBN"^^xsd:anyURI;
     rdfs:label "ISBN"^^xsd:string.
:Book a process:Output;
     process:parameterType "http://dbpedia.org/ontology/Book"^^xsd:anyURI;
     rdfs:label "Book"^^xsd:string.
:ValidISBN a expr:SWRL-Condition;
  expr:expressionLanguage expr:SWRL;
  expr:expressionBody "<swrl:AtomList
  xmlns:swrl=\"http://www.w3.org/2003/11/swrl#\">
     <rdf:first xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\">
     <swrl:ClassAtom>
     <swrl:classPredicate
     rdf:resource=\"http://bookstore.com/DomainOntology#Valid\" />
          <swrl:argument1 rdf:resource=\"#ISBN\" />
</swrl:ClassAtom>
</rdf:first>
<rdf:rest rdf:resource=\"http://www.w3.org/1999/02/22-rdf-syntax-
                    ns#nil\" xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-
                    syntax-ns#\">
</swrl:AtomList>"^^rdf:XMLLiteral.
```

```
:BookhasISBN a process:Result;
     process:hasEffect [ a expr:SWRL-Expression;
     expr:expressionLanguage expr:SWRL;
     expr:expressionBody "<swrl:AtomList
     xmlns:swrl=\"http://www.w3.org/2003/11/swrl#\">
     <rdf:first xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\">
     <swrl:ClassAtom>
     <swrl:classPredicate
     rdf:resource=\"http://bookstore.com/DomainOntology#hasISBN\" />
                    <swrl:argument1 rdf:resource=\"#Book\" />
                    <swrl:argument2 rdf:resource=\"#ISBN\" />
     </swrl:ClassAtom>
     </rdf:first>
     <rdf:rest rdf:resource=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#nil\"
             xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\" />
             </swrl:AtomList>"^^rdf:XMLLiteral.].
:getBookByISBNGrounding a grounding:WsdlGrounding;
     grounding:hasAtomicProcessGrounding :getBookByISBNAtomicProcessGrounding;
     service:supportedBy :getBookByISBNService.

:getBookByISBNAtomicProcessGrounding a grounding:WsdlAtomicProcessGrounding;
     grounding:owlsProcess :getBookByISBNProcess;
     grounding:wsdlDocument
     "http://bookstore.com/wsdl/bookbyisbn.wsdl"^^xsd:anyURI;
     grounding:wsdlOperation [  a grounding:WsdlOperationRef;
     grounding:operation "groundingWSDL:getBookByISBN"^^xsd:anyURI;
     grounding:portType "groundingWSDL:BookByISBNSoap"^^xsd:anyURI.];

grounding:wsdlInputMessage "groundingWSDL:getBookByISBNRequest"^^<xsd:anyURI>;
grounding:wsdlOutputMessage
     "groundingWSDL:getBookByISBNResponse"^^<xsd:anyURI>;
      grounding:wsdlInput [a grounding:WsdlInputMessageMap;
       grounding:owlsParameter :ISBN;
       grounding:wsdlMessagePart "groundingWSDL:ISBN"^^xsd:anyURI;
       grounding:xsltTransformationString "<?xml version=\"1.0\"?>
        <xsl:stylesheet version=\"1.0\"
           xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\"
           xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"
           xmlns:domOnt=\"http://bookstore.com/DomainOntology#\">
           <xsl:template match=\"/ \">
           <xsl:value-of select=\"rdf:RDF/domOnt:ISBN/domOnt:hasISBNValue\"/>
           </xsl:template></xsl:stylesheet>"^^xsd:string.];
     grounding:wsdlOutput [  a grounding:WsdlOutputMessageMap;
       grounding:owlsParameter :Book;
       grounding:wsdlMessagePart "groundingWSDL:book"^^xsd:anyURI;
       grounding:xsltTransformationString "<xsl:stylesheet version=\"1.0\"
               xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\">
               <xsl:template match=\"/\">
               <rdf:RDF xmlns:rdfs=\"http://www.w3.org/2000/01/rdf-schema#\"
               xmlns:dbpedia=\"http://dbpedia.org/ontology/\"
               xmlns:xsd=\"http://www.w3.org/2001/XMLSchema#\"
               xmlns:owl=\"http://www.w3.org/2002/07/owl#\"
               xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"
               xmlns:dc=\"http://purl.org/dc/terms/\">
               <dbpedia:Book>
               <dc:title rdf:datatype=\"xsd:string\">
               <xsl:value-of select=\"book/title\"/>
               </dc:title>
               <dc:author rdf:datatype=\"xsd:string\">
               <xsl:value-of select=\"book/author\"/>
               </dc:author>
               </dbpedia:Book>
               </rdf:RDF>
               </xsl:template>
               </xsl:stylesheet>"^^xsd:string.].
```

Appendix E

2. Order a Book

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix service: <http://www.daml.org/services/OWL-S/1.1/Service.owl#>.
@prefix profile: <http://www.daml.org/services/OWL-S/1.1/Profile.owl#>.
@prefix process: <http://www.daml.org/services/OWL-S/1.1/Process.owl#>.
@prefix grounding: <http://www.daml.org/services/OWL-S/1.1/Grounding.owl#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix expr: <http://www.daml.org/services/OWL-
S/1.1/generic/Expression.owl#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix domOnt: <http://bookstore.com/DomainOntology#>.
@prefix groundingWSDL: <http://bookstore.com/wsdl/bookorder.wsdl#>.
@prefix : <http://bookstore.com/owls/BookOrderService.owls#>.
<http://bookstore.com/owls/BookOrderService.owls> a owl:Ontology;
     owl:imports     <http://bookstore.com/DomainOntology.owl>,
          <http://dbpedia.org/ontology/>,
          <http://www.daml.org/services/OWL-S/1.1/Grounding.owl>,
          <http://www.daml.org/services/OWL-S/1.1/Process.owl>,
          <http://www.daml.org/services/OWL-S/1.1/Profile.owl>,
          <http://www.daml.org/services/OWL-S/1.1/Service.owl>,
          <http://www.w3.org/2003/11/swrl>,
          <http://www.daml.org/services/OWL-S/1.1/generic/Expression.owl>.
:BookOrderServiceService a service:Service;
     service:describedBy :BookOrderServiceProcess;
     service:presents :BookOrderServiceProfile;
     service:supports :BookOrderServiceGrounding.

:BookOrderServiceProfile a profile:Profile;
     profile:hasInput :Book;
     profile:hasOutput :Order;
      profile:hasResult :OrderedBook;
     profile:serviceName "BookOrderService";
     service:presentedBy :BookOrderServiceService.

:BookOrderServiceProcess a process:AtomicProcess;
     process:hasInput :Book;
     process:hasOutput :Order;
     process:hasResult :OrderedBook;
     service:describes :BookOrderServiceService;
     rdfs:label "BookOrderServiceProcess".

:Book a process:Input;
     process:parameterType "http://dbpedia.org/ontology/Book"^^xsd:anyURI;
     rdfs:label "Book".

:Order a process:Output;
 process:parameterType "http://bookstore.com/DomainOntology#Order"^^xsd:anyURI;
 rdfs:label "Order".

:OrderedBook a process:Result;
   process:hasEffect [     a expr:SWRL-Expression;
     expr:expressionLanguage expr:SWRL;
     expr:expressionBody
     "<swrl:AtomList xmlns:swrl=\"http://www.w3.org/2003/11/swrl#\">
     <rdf:first xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\">
       <swrl:ClassAtom>
        <swrl:classPredicate
        rdf:resource=\"http://bookstore.com/DomainOntology#containsItem\" />
          <swrl:argument1 rdf:resource=\"#Order\" />
          <swrl:argument2 rdf:resource=\"#Book\" />
       </swrl:ClassAtom>
       </rdf:first>
     <rdf:rest rdf:resource=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#nil\"
          xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\" />
          </swrl:AtomList>"^^rdf:XMLLiteral.].
```

```
:BookOrderServiceGrounding a grounding:WsdlGrounding;
     service:supportedBy :BookOrderServiceService;
     grounding:hasAtomicProcessGrounding
:BookOrderServiceAtomicProcessGrounding.

:BookOrderServiceAtomicProcessGrounding a
grounding:WsdlAtomicProcessGrounding;
 grounding:owlsProcess :BookOrderServiceProcess;
 grounding:wsdlDocument "http://bookstore.com/wsdl/bookorder.wsdl"^^xsd:anyURI;
 grounding:wsdlOperation [a grounding:WsdlOperationRef;
    grounding:operation "groundingWSDL:BookOrderService"^^xsd:anyURI;
    grounding:portType "groundingWSDL:BookOrderServiceSoap"^^xsd:anyURI.];
grounding:wsdlInputMessage groundingWSDL:BookOrderServiceRequest"^^xsd:anyURI;
grounding:wsdlOutputMessage "groundingWSDL:BookOrderServiceResponse"^^xsd:anyURI;
grounding:wsdlInput [a grounding:WsdlInputMessageMap;
    grounding:owlsParameter :Book;
    grounding:wsdlMessagePart "groundingWSDL:book"^^xsd:anyURI;
    grounding:xsltTransformationString
          "<xsl:stylesheet version=\"1.0\"
          xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\"
          xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"
          xmlns:domOnt=\"http://bookstore.com/DomainOntology#\"
          xmlns:tns=\"http://bookstore.com/wsdl/bookorder.wsdl\">
          <xsl:template match=\"/\">
          <xsl:for-each select=\"rdf:RDF/domOnt:Book/\">
          <book>
          <ISBN><xsl:value-of select=\"dbpedia:isbn\"/></ISBN>
          <title><xsl:value-of select=\"dc:title\"/></title>
          <author><xsl:value-of select=\"dc:author\"/></author>
          </book>
          </xsl:template>
          </xsl:stylesheet>"^^xsd:string.
          ];
     grounding:wsdlOutput [a grounding:WsdlOutputMessageMap;
          grounding:owlsParameter :Order;
          grounding:wsdlMessagePart "groundingWSDL:order"^^xsd:anyURI;
          grounding:xsltTransformationString "<xsl:stylesheet version=\"1.0\"
               xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\">
               <xsl:template match=\"/\">
               <rdf:RDF xmlns:rdfs=\"http://www.w3.org/2000/01/rdf-schema#\"
                    xmlns:dbpedia=\"http://dbpedia.org/ontology/\"
                    xmlns:xsd=\"http://www.w3.org/2001/XMLSchema#\"
                    xmlns:owl=\"http://www.w3.org/2002/07/owl#\"
                    xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"
                    xmlns:dc=\"http://purl.org/dc/terms/\"
                    xmlns:domOnt=\"http://bookstore.com/DomainOntology#\">
                    <domOnt:Order>
                    <domOnt:containsItem>
                    <dbpedia:Book>
                    <dbpedia:isbn>
                    <xsl:value-of select=\"Order/Book/ISBN\"/>
                    </dbpedia:isbn>
                    </dbpedia:Book>
                    </domOnt:containsItem>
                    <domOnt:OrderID>
                    <xsl:value-of select=\"Order/@orderId\"/>
                         </domOnt:OrderID>
                         </domOnt:Order>
                    </rdf:RDF>
                    </xsl:template>
                    </xsl:stylesheet>"^^xsd:string. ].
```

Appendix E

Composite Service

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix list: <http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#>.
@prefix grounding: <http://www.daml.org/services/owl-s/1.1/Grounding.owl#>.
@prefix profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>.
@prefix process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
@prefix service: <http://www.daml.org/services/owl-s/1.1/Service.owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
<http://bookstore.com/owls/OrderBookByISBNService.owls> a owl:Ontology;
    owl:imports <http://bookstore.com/DomainOntology.owl>,
          <http://dbpedia.org/ontology/>,
          <http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl>,
          <http://www.daml.org/services/owl-s/1.1/Grounding.owl>,
          <http://www.daml.org/services/owl-s/1.1/Process.owl>,
          <http://www.daml.org/services/owl-s/1.1/Profile.owl>,
          <http://www.daml.org/services/owl-s/1.1/Service.owl>,
          <http://www.w3.org/2003/11/swrl>.
:OrderBookByISBNService  a service:Service;
     service:describedBy :OrderBookByISBNProcess;
     service:presents :OrderBookByISBNProfile;
     service:supports :OrderBookByISBNGrounding.
:OrderBookByISBNProfile a profile:Profile;
     profile:hasInput :Book,:ISBN;
     profile:hasOutput :Order;
     profile:serviceName "Order Book By ISBN";
     service:presentedBy :OrderBookByISBNService.
:Book a process:Input;
 process:parameterType "http://dbpedia.org/ontology/Book"^^xsd:anyURI.
:ISBN a process:Input;
 process:parameterType "http://bookstore.com/DomainOntology#ISBN"^^xsd:anyURI.
:Order a process:Output;
 process:parameterType "http://bookstore.com/DomainOntology#Order"^^xsd:anyURI.
:OrderBookByISBNProcess a process:CompositeProcess;
     process:composedOf [ a process:Sequence;
     process:components [ a process:ControlConstructList;
                          list:first :Perform1;
                          list:rest [ list:first :Perform2;
                           list:rest list:nil.].].];
     process:hasInput  :Book,:ISBN;
     process:hasOutput :Order;
     process:hasResult [ a process:Result;
                         process:withOutput [ a process:OutputBinding;
                         process:toParam :Order;
                         process:valueSource [ a process:ValueOf;
                         process:fromProcess :Perform2;
                         process:theVar :Order.];    ]; ];
     service:describes :OrderBookByISBNService.
:Perform1 a process:Perform;
     process:hasDataFrom [ a process:InputBinding;
process:toParam <http://bookstore.com/owls/getBookByISBN.owls#ISBN>;
process:valueSource [ a process:ValueOf;
process:fromProcess process:TheParentPerform;
                      process:theVar :ISBN. ];   ];
process:process <http://bookstore.com/owls/getBookByISBN.owls#getBookByISBNProcess>.
:Perform2 process:hasDataFrom [a process:InputBinding;
 process:toParam <http://bookstore.com/owls/BookOrderService.owls#Book>;
 process:valueSource [ a process:ValueOf;
          process:fromProcess :Perform1;
          process:theVar <http://bookstore.com/owls/getBookByISBN.owls#Book>;].];
 process:process <http://bookstore.com/owls/BookOrderService.owls#BookOrderServiceProcess>;
     a process:Perform.
:OrderBookByISBNGrounding a grounding:WsdlGrounding;
      grounding:hasAtomicProcessGrounding
<http://bookstore.com/owls/BookOrderService.owls#BookOrderServiceGrounding>,
<http://bookstore.com/owls/getBookByISBN.owls#getBookByISBNGrounding>;
     service:supportedBy :OrderBookByISBNService.
```

The bookstore domain ontology

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc: <http://purl.org/dc/terms/>.
@prefix : <http://bookstore.com/DomainOntology#>.

<http://bookstore.com/DomainOntology> a owl:Ontology.

:Valid          a owl:Class.

:ISBN      a owl:Class.

:hasISBN   a owl:ObjectProperty;
           rdfs:domain     dbpedia:Book;
           rdfs:range      :ISBN.

:hasISBNValue    a owl:DatatypeProperty;
            rdfs:domain :ISBN;
            rdfs:range     xsd:string.


:Order a owl:Class.

:containsItem a owl:ObjectProperty;
                rdfs:domain     :Order;
                rdfs:range :Book.

:orderID a           owl:DatatypeProperty;
                rdfs:domain     :Order;
                rdfs:range  xsd:string.
```

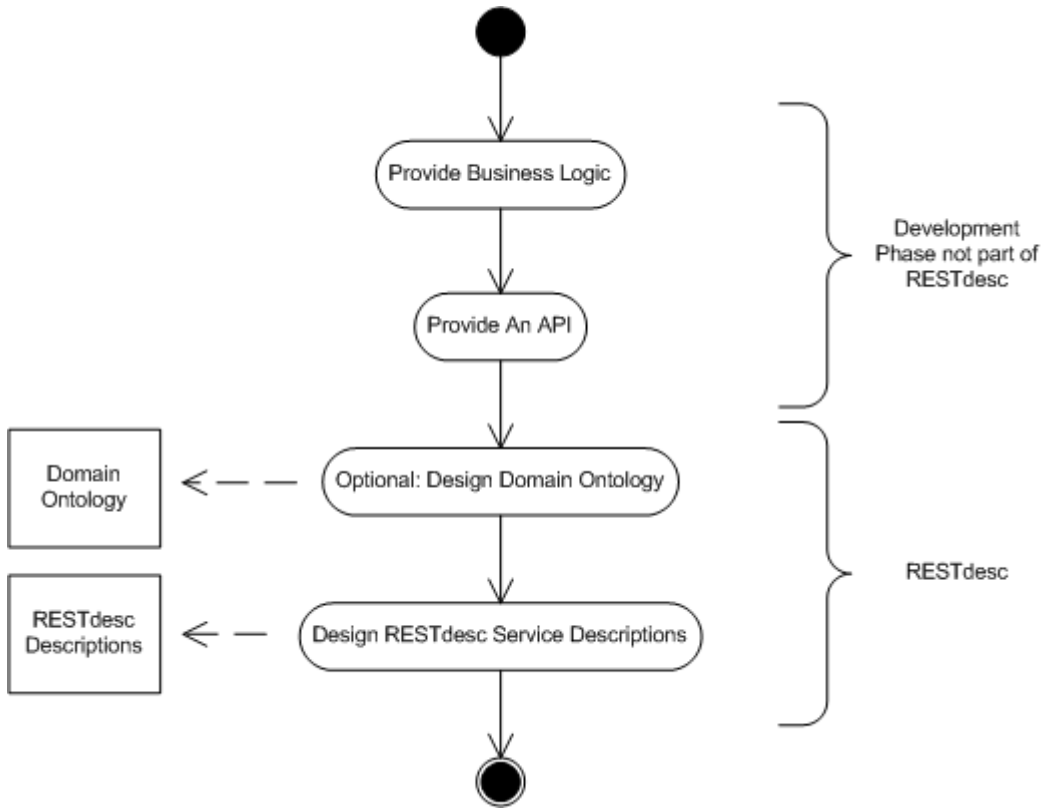
OWL-S Interaction

Client	Server
GET /owls/getBookByISBN.owls HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK 
GET /DomainOntology HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK 
GET /wsdl/getBookByISBN.wsdl HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK 
POST /BookService HTTP/1.1 Host: bookstore.com <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"> <soapenv:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wns="http://bookstore.com/wsdl/bookbyisbn.wsdl"> <wns: getBookByISBN> <wns:getBookByISBNRequest> 0123735564 <wns:getBookByISBNRequest> </wns: getBookByISBN> </soapenv:Body> </soapenv:Envelope>	
	HTTP/1.1 200 OK Content-Type: application/soap+xml <?xml version="1.0"?> <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope" soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"> <soap:Body xmlns:wsn=" http://bookstore.com/wsdl/bookbyisbn.wsdl"> <wns:getBookByISBN> <wns:getBookByISBNResponse> <wns:book> <wns:title>Semantic Web for the Working </wns:title> <wns:author>Allemang and Hendler </wns:author> <wns:isbn>0123735564</wns:isbn> </wns:book> </wns: getBookByISBNResponse> <wns:getBookByISBN>

	</soap:Body> </soap:Envelope>
GET /owls/BookOrderService.owls HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK **Order Book OWL-S File**
GET /wsdl/getBookByISBN.wsdl HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK **Order Book WSDL File**
POST /BookService HTTP/1.1 Host: bookstore.com <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"> <soapenv:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wns="http://bookstore.com/wsdl/bookorder.wsdl"> <wns: BookOrder> <wns:BookOrderRequest> <wns:book> <wns:title>Semantic Web for the Working Ontologist </wns:title> <wns:author>Allemang and Hendler</wns:author> <wns:isbn>0123735564</wns:isbn> </wns:Book> </wns:BookOrderRequest> </wns: BookOrder> </soapenv:Body> </soapenv:Envelope>	
	<?xml version="1.0"?> <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope" soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"> <soap:Body xmlns:wsn=" http://bookstore.com/wsdl/bookorder.wsdl"> <wns:BookOrder> <wns:BookOrderResponse> <wns:order ordered="12345"> <wns:book> <wns:title>Semantic Web for the Working </wns:title> <wns:author>Allemang and Hendler </wns:author> <wns:isbn>0123735564</wns:isbn> </wns:book> </wns:order> </wns:BookOrderResponse> <wns:BookOrder> </soap:Body> </soap:Envelope>

Appendix E

RESTdesc



RESTdesc – Service Design and Deployment

RESTdesc does not involve the steps regarding deploying the API, because it assumes the APIs have already been developed and deployed. However because we are comparing it with EXPRESS, and this is involved in the development phase, it is necessary to discuss the tasks RESTdesc assumes are there.

In developing the API, several factors come into place, like the programming language and framework, their support of HTTP (RESTful concepts), the existence of legacy software, involved or a Web Application.

However, process usually is the same across RESTful Web service frameworks:

1. Map the data into resources.
2. Specify their URIs.
3. Decide which HTTP methods can be performed on the resource.
4. Design the representations sent and accepted to and from the client, decide on the media types, and link the resources.

The Existing Web API

Resource	Description
1. GET /Books?isbn={ISBN}	Returns a book with the given ISBN
2. POST /Order	Given a book, this creates an order for a book, returns order ID

1. GET /Books?isbn={ISBN}

Sample Request

```
GET  /Books?isbn={1585425524} HTTP/1.1
Host: www.bookstore.com
Accept: application/json;
```

Sample Response

```
HTTP/1.1 200 OK
{"book": {
   "isbn": "0123735564",
   "title": "Semantic Web for the Working Ontologist",
   "author": "Allemang and Hendler"
  }}
```

2. POST /Order

Sample Request

```
POST /Order HTTP/1.1
Content-type: application/json;
Accept: application/json;

{"book": {
   "ISBN": "0123735564",
   "title": "Semantic Web for the Working Ontologist",
   "author": "Allemang and Hendler"
}}
```

Sample Response

```
HTTP/1.1 201 Created
Location: http://www.bookstore.com/Order/123242

 {"order": {
   "id": "123242",
   "contains":
    [{"book": {"isbn": "0123735564",
                "title": "Semantic Web for the Working Ontologist",
                "author": "Allemang and Hendler"
                }}
    ]
  }
}
```

Appendix E

RESTdesc – Semantic Description

RESTdesc v.1

1 GET /Books?isbn={ISBN}

```
@prefix http: <http://www.w3.org/2011/http#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix tmpl: <http://purl.org/restdesc/http-template#>.
@prefix dc: <http://purl.org/dc/terms/>.
{
?book    a       dbpedia:Book;
                 dbpedia:isbn             ?isbn.
}
=>
{
_:request         http:methodName          "GET";
                  http:requestURI         ("/books?isbn=" ?isbn);
                  http:resp               [ http:body ?book ].

?book             dc:title                _:title;
                  dc:author               _:author.}.
```

2 POST /Order V.1

```
@prefix ord: <http://bookstore.com/bookorder#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix tmpl: <http://purl.org/restdesc/http-template#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
{
?book    a       dbpedia:Book;
}
=>
{
_:request        http:methodName     "POST";
                 http:requestURI     "/order";
                 http:body           [ tmpl:formData ("book=" ?book ) ];
                 http:resp           [ tmpl:location ("order/" ?orderID);
                                       http:body    ?order ].
?order   a       ord:Order;
                 ord:orderID         ?orderId;
                 ord:containsItem    ?book.}.
```

RESTdesc v2.0

1 GET (BookURI)

```
@prefix http: <http://www.w3.org/2011/http#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix tmpl: <http://purl.org/restdesc/http-template#>.
@prefix dc: <http://purl.org/dc/terms/>.
{
  ?book          a       dbpedia:Book;
}
=>
{
  _:request      http:methodName           "GET";
                 http:requestURI           ?book ;
                 http:resp                [ http:body ?book ].
                 ?book dc:title            _:title.
                 ?book dc:author           _:author.}.
```

2 POST (BookURI)

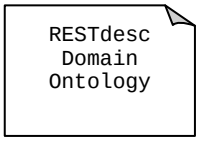
```
@prefix ord: <http://bookstore.com/bookorder#>.
@prefix http: <http://www.w3.org/2011/http#>.
{
  ?book   a      dbpedia:Book;
}
```

```
=>
{
 _:request       http:methodName            "POST";
                 http:requestURI            ?book;
                 http:resp                  [ http:body ?order ].

?order   a       ord:Order;
                 ord:orderID               _:id;
                 ord:containsItem          ?book.}.
```

The bookstore domain ontology for the RESTdesc Service

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix : <http://bookstore.com/bookorder#>.

<http://bookstore.com/bookorder> a owl:Ontology.

:Order    a          owl:Class.

:containsItem a      owl:ObjectProperty;
                rdfs:domain     :Order;
                rdfs:range      dbpedia:Book.

:orderID a           owl:DatatypeProperty;
                rdfs:domain :Order;
                rdfs:range xsd:string.
```

RESTdesc – Interaction

Client	Server
OPTIONS /API HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK Allow: GET,HEAD,OPTIONS Content-Type: text/n3 
GET /bookorder HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK Content-Type: text/n3 
GET /book?isbn={0123735564} HTTP/1.1 Host: bookstore.com	
	HTTP/1.1 200 OK {"book": { "isbn": "0123735564", "title": "Semantic Web for the Working Ontologist", "author": "Allemang and Hendler", "uri": "http://bookstore.com/books/0123735564" }}
POST /order HTTP/1.1 Host: bookstore.com {"book": { "isbn": "0123735564", "title": "Semantic Web for the Working Ontologist", "author": "Allemang and Hendler" }}	
	HTTP 201 Created Location: http://bookstore.com/Order/12345

PART TWO

EXPRESS Example 1:

Endpoint:

```
GET http://bookstore.com/EXPRESS/Book?isbn={}
```

Ontology:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix dc: <http://purl.org/dc/terms/>.
@prefix : <http://bookstore.com/EXPRESS/>.
<http://bookstore.com/EXPRESS> a owl:Ontology.

:Book a owl:Class;
     owl:equivalentClass dbpedia:Book.

:title a owl:DatatypeProperty;
     owl:equivalentProperty dc:title;
     rdfs:domain :Book;
     rdfs:range xsd:string.

:author a owl:DatatypeProperty;
     owl:equivalentProperty dc:author;
     rdfs:domain :Book;
     rdfs:range xsd:string.

:isbn a owl:DatatypeProperty;
     owl:equivalentProperty dbpedia:isbn;
     rdfs:domain :Book;
     rdfs:range xsd:string.
```

EXPRESS Example 2:

Endpoint:

```
GET http://bookstore.com/getBookByISBN/Book?_isbn={}
```

Ontology:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix domOnt: <http://bookstore.com/DomainOntology#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix : <http://bookstore.com/getBookByISBN/>.

<http://bookstore.com/getBookByISBNEXPRESS.owl> a owl:Ontology;
     owl:imports <http://bookstore.com/DomainOntology>,
                    <http://dbpedia.org/ontology>.

:Book a owl:Class;
     owl:equivalentClass  dbpedia:Book.

:ISBN a owl:Class;
     owl:equivalentClass  domOnt:ISBN.

:_isbn a owl:ObjectProperty;
     rdfs:domain      dbpedia:Book;
     rdfs:range       domOnt:ISBN.
```

domOnt : http://bookstore.com/DomainOntology

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix : <http://bookstore.com/DomainOntology#>.

<http://bookstore.com/DomainOntology> a owl:Ontology.

:ISBN a owl:Class.

:hasISBN a owl:ObjectProperty;
     rdfs:domain dbpedia:Book;
     rdfs:range :ISBN.

:hasISBNValue a owl:DatatypeProperty;
     rdfs:domain :ISBN;
     rdfs:range xsd:string.
```

Participant Information Sheet

Study Title: Towards RESTful and Resource-Oriented Semantic Web Services

Researcher: Areeb Alowisheq

Ethics number: 6324

Please read this information carefully before deciding to take part in this research. If you are happy to participate you will be asked to sign a consent form.

What is the research about?

This research investigates an approach for simplifying the development of Semantic Web services (SWS) by reducing their semantic descriptions. This study aims to compare three Semantic Web service approaches in terms of their required development effort. These are OWL-S, RESTdesc and EXPRESS.

Why have I been chosen to participate?

You invited to participate in this study because you are an expert in Semantic Web Technologies. Your opinion and expertise will help in assessing and comparing aspects of different Semantic Web service approaches.

What will happen to me if I take part?

I will ask you to sign a consent form, and then the study will begin. I will conduct an interview with you, with open-ended questions, and I will record your voice during the interview.

Are there any benefits in my taking part?

This research is not designed to help you personally, but your feedback will help me gather expert opinions on the development efforts

Will my participation be confidential?

Yes. Any data will be stored will not be linked to your name. Your data and that of other participants will be stored and used on secure systems.

Are there any risks involved?

No.

What happen if I change my mind?

You have the right to terminate your participation in the research, at any stage, you do not need to give any reasons, and without your legal rights being affected. Any data collected form you will be immediately destroyed.

Where I can get more information?

For further details, please contact either myself or my study supervisor, Dr.David E. Millard.

Areeb Alowisheq: **aaa08r@ecs.soton.ac.uk**

Dr.David E. Millard: **dem@ecs.soton.ac.uk**

CONSENT FORM (Version 1)

Study title: Towards Resource-oriented RESTful Semantic Web services

Researcher name: Areeb Alowisheq

Study reference: Towards Resource-oriented RESTful Semantic Web services

Ethics reference: 6324

Please initial the box(es) if you agree with the statement(s):

I have read and understood the information sheet and have had the opportunity to ask questions about the study	☐
I agree to take part in this research project and agree for my data to be used for the purpose of this study	☐
I understand my participation is voluntary and I may withdraw at any time without consequence and my data will be deleted if I withdraw at any time	☐
I agree to record my voice during my participation in this study	☐

Data Protection

I understand that information collected about me during my participation in this study will be stored on a password protected computer and that this information will only be used for the purpose of this study. All files containing any personal data will be made anonymous.

Name of participant (print name)……

Signature of participant……..……..…….

Name of Researcher (print name): Areeb Alowisheq

Signature of Researcher……..

Date……

Appendix F: Sample Expert Review Transcript

The transcript below is from the interview with second expert, answering the question about development speed.

Time	Speaker	Discussion
29:18	Researcher	If you were required to provide a semantic API for a bookstore, if you had to use one of these approaches how long do you think it will take you?
29:36	Participant	For each approach?
29:37	Researcher	Yes.
29:40	Participant	I think so in order of time, sort of ascending I imagine, it will go like, I guess RESTdesc, EXPRESS and then OWL-S. But it does depend I think on, if you are building it with that in mind I suppose, than if you are building it, a service, from scratch, from the start, then I guess OWL-S is rigorous way of building it, but I think for development time so I guess RESTdesc would be the quickest way, just because of simplicity, then I guess EXPRESS after that coz it's a, I guess, you have to bear it more in mind while you are developing it, I guess as soon as you get into WSDL Web services that will get more complex so it needs more debugging time and testing and um.. yeah it is a lot harder. I think it will bind you a bit more to which languages you developing it as well, so some languages have better support than other languages, so I guess for SOAP services java obviously is really good, if you go to python or PHP it gets more sort of unreliable, I can imagine RESTdesc and EXPRESS are a bit more lighter weight, and it is less reliant on what language you use, so you can probably be a bit more agnostic for language, yeah

Appendix G: Expert Review Analysis Screenshots

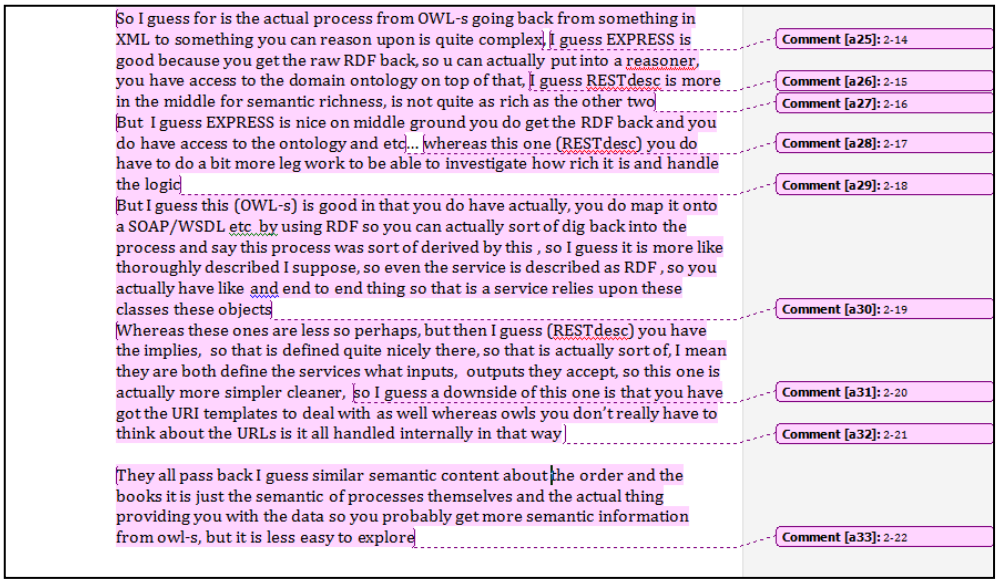


Figure 46 Interview analysis document, text is annotated with identifiers

Appendix G

	A	B	C	D	E	F
1	Interview	Quote ID	Quote	Sub-theme	Theme	Interview Question
2	1	1-2	"and the owl-s approach is heavy duty. Not quite structured as RESTdesc,"	OWL-s Cons	Development Speed	Q2. Providing Semantic API, How Long?
3	1	1-5	"RESTdecs should be very straightforward to build on top of the existing HTTP APIs, just follows the n3 format which is widely accepted. I think it even providing the necessary ontologies and the full pattern of the HTTP API it can be even done by automatic process it should be really fast"	RESTdesc Pros	Development Speed	Q3. Providing Clients, describe in terms of : 1- ease of use, dev speed 2- semantic quality
4	1	1-6	"It can be really fast even for EXPRESS, I cannot see the true difference between the EXPRESS approach and HTTP API approach, certainly it is more structured but in terms of semantic quality, it doesn't provide much semantics"	EXPRESS Pros	Development Speed	Q3. Providing Clients, describe in terms of : 1- ease of use, dev speed 2- semantic quality
5	2	2-13	obviously OWL-s you are doing SOAP so it may actually be quicker so there are a few lines you could say grab the WSDL file and form my request and it is good that clients don't need to know actually that it is linked data behind the scenes, not having to worry about what the underlying technology is	OWL-s Pros	Development Speed	Q3. Providing Clients, describe in terms of : 1- ease of use, dev speed 2- semantic quality
6	2	2-6	"It is amazing how much verbose it is It amazing the different in size compared to the other two"	OWL-s Cons	Development Speed	Before during explanation
7	2	2-9	so in order of time, sort of ascending I imagine, it will go like, RESTdesc, EXPRESS and then OWL-s, But I guess it depends on if you are building it with that in mind, than if you are building it from scratch, from the start, then I guess OWL-s is rigorous way of building it, so I guess RESTdesc would be the quickest, just because of simplicity, then I guess EXPRESS after that coz it's a you have to have it in mind while you are developing it, I guess as soon as you get into WSDL web services that will get more complex so it needs more debugging time and testing and.. yeah it is a lot harder.	RESTdesc Pros	Development Speed	Q2. Providing Semantic API, How Long?
8	2	2-9	so in order of time, sort of ascending I imagine, it will go like, RESTdesc, EXPRESS and then OWL-s, But I guess it depends on if you are building it with that in mind, than if you are building it from scratch, from the start, then I guess OWL-s is rigorous way of building it, so I guess RESTdesc would be the quickest, just because of simplicity, then I guess EXPRESS after that coz it's a you have to have it in mind while you are developing it, I guess as soon as you get into WSDL web services that will get more complex so it needs more debugging time and testing and.. yeah it is a lot harder.	OWL-s Cons	Development Speed	Q2. Providing Semantic API, How Long?
9	2	2-9	so in order of time, sort of ascending I imagine, it will go like, RESTdesc, EXPRESS and then OWL-s, But I guess it depends on if you are building it with that in mind, than if you are building it from scratch, from the start, then I guess OWL-s is rigorous way of building it, so I guess RESTdesc would be the quickest, just because of simplicity, then I guess EXPRESS after that coz it's a you have to have it in mind while you are developing it, I guess as soon as you get into WSDL web services that will get more complex so it needs more debugging time and testing and.. yeah it is a lot harder.	EXPRESS Pros	Development Speed	Q2. Providing Semantic API, How Long?
10	2	2-11	I think EXPRESS and RESTdesc would be fairly similar for development speed, there are fairly standard POSTing GETting to a URL so there is nothing those languages can't do already so it is fairly straightforward	RESTdesc Pros	Development Speed	Q3. Providing Clients, describe in terms of : 1- ease of use, dev speed 2- semantic quality
			I think EXPRESS and RESTdesc would be fairly similar for development speed, there are fairly standard POSTing GETting to a URL so there is nothing those			Q3. Providing Clients, describe in terms of : 1- ease of use, dev speed 2-

Figure 47 Interview analysis spreadsheet, Quote ID are the identifiers in Figure 46