Title	Semantic Service Oriented Architecture - Component Model, Reference Architecture and Evaluated Prototype

Author(s)
Moran, Matthew John

Publication Date
2011-08-24

Item record
http://hdl.handle.net/10379/2162

Some rights reserved. For more information, please see the item record link above.
SEMANTIC SERVICE ORIENTED ARCHITECTURE -
COMPONENT MODEL, REFERENCE ARCHITECTURE AND
EVALUATED PROTOTYPE

Matthew John Moran

Submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy

Supervisors:
Prof. Manfred Hauswirth
Dr. Sami Bhiri

Internal Examiner: External Examiners:
Prof. Stefan Decker Prof. Samir Tata

Digital Enterprise Research Institute (DERI), National University of Ireland, Galway (NUIG)
April 2011
The studies presented in this thesis were performed at the Digital Enterprise Research Institute (DERI) at the National University of Ireland, Galway (NUIG). This material is based upon works supported by the Science Foundation Ireland under Grant No. SFI/02/CE1/I131 (Lión) and EU FP6 funding under the DIP project (FP6 - 507483).

©2011 Matthew Moran. All rights reserved.
Abstract

This dissertation presents a component model, reference architecture and peer-evaluated prototype for Semantic Service Oriented Architecture (SSOA). Semantic Web service technology, derived from the combination of the Semantic Web and Web services, is the enabling-technology for SSOA. Allowing the semantics of the public data and process model of Web services to be unambiguously communicated to potential service-consumers, provides the basis for Web services to be discovered, composed, and invoked as needed. The alternative is that these actions must be carried out manually in advance. The thesis is motivated by the fact that, although a body of research has developed into many aspects of Semantic Web services, a gap remains between conceptual models and languages for Semantic Web services on the one hand, and prototype implementations using those models and languages, primarily for service discovery, on the other. Filling this gap requires a combination of top-down and bottom-up approaches.

From a top-down perspective, the absence of a reference software architecture model, as distinct from a conceptual model, for Semantic Web services gives rise to a number of shortcomings in the body of knowledge. These include the identification of a minimal component model for an SSOA, defining corresponding behavioural and component communication models and, validating this architecture, both experimentally and qualitatively so that it can be regarded as a reference point for further research. Experimental validation is achieved by peer-review, through the application of a prototype implementation of the SSOA to a set of open and independently specified system-integration problems. Qualitative evaluation is carried out by first determining a matrix of requirements on which the SSOA is based, and using this as the basis for relating five contemporary Semantic Web service frameworks to the SSOA.

The thesis moves on to describe how research challenges arose in the course of building and validating the prototype SSOA implementation. Two specific challenges were in the areas of (1) Semantic Web service discovery where incomplete data, available in the service description, must be augmented by fetching additional data at the time the service is required, and (2) the enactment of process mediation between the public interface descriptions of a service requester and a matching Web service offer. Devising solutions to these problems, applying them to real-world problems and evaluating their outcomes both experimentally and qualitatively represents problem-driven contributions of this thesis from a bottom-up perspective.
Acknowledgments

Completing this thesis has been a long but rewarding road. There are many people to thank who have helped me along the way. I would like to thank Manfred Hauswirth for his sound advice, guidance and support over many years. I am, additionally, especially grateful to Sami Bhiri for his thorough and patient review of the content of this thesis, which has improved it significantly and, in no small way, has gotten me over the finishing line.

Thanks to Stefan Decker for all his help, the challenging conversations, and his constant enthusiasm. I am very grateful to Samir Tata for the time he invested in acting as my external examiner, and for his insightful and beneficial comments, which have added real value to this thesis. Thanks to Chris Bussler for introducing me to the field of Semantic Web service-based integration, guiding me through the first couple of years and remaining a good friend since. Thanks also to Dieter Fensel who pushed and cajoled research papers out of me, and the other DERI students in the early years, and gave us many opportunities to extend our research network.

Thanks to Maciej Zaremba and Tomas Vitvar for our fruitful and enjoyable collaborations. I’m especially grateful to Maciej, as a generous and good friend, who carefully reviewed the final drafts of this thesis and kept me calm. I’d like to thank the original members of the SWS Research Cluster - Michal, Laurentiu, Ina, Adrian, Emilia, Armin, Brahmananda, Mick and Eyal for the many brainstorming sessions, the friendship, and the occasional pint. I’d also like to say a special thank you to Knud Moeller, for his good-natured friendship, and his help with the setting out of this thesis. Thanks too to Hilda and Claire who smoothed the way through NUIG over the years, and to Brian Wall for sound advice.

You can never really thank your parents adequately for all they go through and do for you, from your very first breath. With all my love, I send my Mam and Dad my thanks - for everything really. To my four beautiful sisters, Maeve, Claire, Joanne and Mags, thank you for all your unconditional support and love. Thanks to Aunties Anne and Eileen for all the love and prayers. Also a special word of thanks to the Gallagher family for welcoming me in so kindly and becoming my new home away from home.

Thanks to Drs. Sandra and Stephen for reassuring support and priceless advice.

I could not have completed this PhD research without my beautiful wife Veronica who I love with all my heart. She encouraged and supported me, and patiently allowed me to find my way through to the end of my research, at the cost of many missed weekends and nights together. Finally, I dedicate this thesis to our impossibly cute son, Tomás, who arrived into our lives just two weeks after I completed this PhD. He is a ray of light, a constant source of joy and amazement, and a great little fella altogether. He is perfect and if we loved him anymore, we’d simply burst.
Do more. Try harder. Start now.

from *In a Sunburned Country* by Bill Bryson

Imiónn an tuirse is fanann an tairbhe.
(The tiredness is forgotten while the result remains.)

Seanfhocal / Irish proverb

It’s all very well to write reviews
And carry umbrellas and keep dry shoes
And say what everyone’s saying there
And wear what everyone else must wear
But tonight I’m sick of the whole affair
I want free life and I want fresh air ...

from *Lasca* by Robert Service

for Bea, Arthur and Madge

I’m livin in Drumlister
An I’m getting very oul
I have to wear an Indian bag
To save me from the coul.
The deil a man in this townland
Wos claner raired nor me,
But I’m livin in Drumlister
In clabber to the knee.

from *Me and me Da* by W.F. Marshall

for Jack and Rose
Contents

I Prelude 1

1 Introduction 3
  1.1 Motivation and Problem Statement 3
  1.2 Research Questions 4
  1.3 Methodology 5
  1.4 Structure 7

II Background, Motivation & Requirements for a Semantic SOA 11

2 Background 13
  2.1 Historical Context 13
    2.1.1 Distributed Computing 13
    2.1.2 Agent Systems 19
    2.1.3 Semantic Web Services Architecture (SWSA) 22
    2.1.4 Summary 23
  2.2 Software Architecture 24
    2.2.1 Foundations 24
    2.2.2 Modelling Software Architecture 25
    2.2.3 Summary 27

3 Motivation & Requirements for Semantic Service Oriented Architecture 28
  3.1 Motivation: From SOA to Semantic SOA 28
    3.1.1 Limitations of Traditional SOA 29
    3.1.2 Scenario for a Semantic SOA 32
    3.1.3 From Distributed and Agent Systems to SSOA 34
  3.2 Requirements for a Semantic SOA 35
5 Algorithms for Three Key SSOA Features – Discovery, Composition and Choreography

5.1 Discovery ......................................................... 73
  5.1.1 WSMO Approach to Discovery .......................... 73
  5.1.2 Algorithm for Discovery with Data-Fetch ............. 74
  5.1.3 Sample WSML Description of a Data-Fetch Interface .. 77

5.2 Composition ................................................... 79
  5.2.1 Algorithm .................................................. 80

5.3 Process Mediation .............................................. 82
  5.3.1 Algorithm for the Choreography Engine ............... 83

5.4 Summary ....................................................... 87

IV Implementation, Evaluation and Related Work 88

6 SSOA Implementation & Evaluation against Independently-Set Integration Scenarios 89

6.1 The Semantic Web Services Challenge ....................... 90
  6.1.1 Usage of The RosettaNet B2B Standard ............... 92

6.2 Mediation Scenario ............................................ 93
  6.2.1 Initial Problem .......................................... 93
  6.2.2 Solution to the Initial Problem ......................... 94
  6.2.3 Runtime Phase .......................................... 96
  6.2.4 Modified Problem ....................................... 102

6.3 Discovery Scenario ............................................ 103
  6.3.1 Initial Problem .......................................... 103
  6.3.2 Solution to Discovery using WSMX .................... 104

6.4 Composition Scenario .......................................... 110
  6.4.1 Solution to Composition ................................ 111

6.5 Experimental Results of the WSMX SSOA in Integration Scenarios ............... 114
  6.5.1 Mediation ................................................ 114
  6.5.2 Discovery Problems ..................................... 115

6.6 Summary ....................................................... 122
7 Comparative Evaluation of WSMX as an SSOA to Peer Systems - DIANE and SWE-ET

7.1 DIANE ......................................................... 124
  7.1.1 DIANE: Data Model .................................. 124
  7.1.2 DIANE: Architecture .................................. 125
  7.1.3 DIANE: Mediation .................................... 126
  7.1.4 DIANE: Discovery .................................... 127
  7.1.5 DIANE: Composition .................................. 128

7.2 SWE-ET ....................................................... 129
  7.2.1 SWE-ET: Data Model .................................. 129
  7.2.2 SWE-ET: Architecture ................................ 130
  7.2.3 SWE-ET: Mediation .................................. 132
  7.2.4 SWE-ET: Discovery ................................... 133
  7.2.5 SWE-ET: Composition ................................ 134

7.3 Comparison of DIANE, SWE-ET and WSMX .................. 134
  7.3.1 Comparison: Architecture ............................ 134
  7.3.2 Comparison: Experimental Evaluation ............... 138
  7.3.3 Comparison: Discovery ................................ 138
  7.3.4 Comparison: Mediation ................................ 141

7.4 Summary ..................................................... 142

8 Related Work .................................................. 145

8.1 The Internet Reasoning Service (IRS) ....................... 145
  8.1.1 Global View ........................................... 146
  8.1.2 Data Model ............................................ 146
  8.1.3 Component Model ...................................... 147
  8.1.4 Behavioural Model .................................... 150
  8.1.5 Communication Model ................................ 151
  8.1.6 Summary ............................................... 151

8.2 DAML-S Virtual Machine & OWL-S Broker .................. 152
  8.2.1 Global View ........................................... 152
  8.2.2 Data Model ............................................ 155
  8.2.3 Component Model ...................................... 155
  8.2.4 Behaviour Model ...................................... 157
List of Figures 203

List of Tables 205

List of Listings 205
Part I

Prelude
1. Introduction

1.1 Motivation and Problem Statement

The Web's ubiquity make it a widely available common infrastructure on which software applications may be invoked remotely by the exchange of messages over the HTTP protocol [Berners-Lee et al., 1996]. [McIlraith et al., 2001] describe how the Web, once solely a repository for text and images, is evolving into a provider of services - information providing services and world altering services. There is an abundance of definitions for Web services but the essence is captured by Benjamins in [Staab et al., 2003] who states that Web services are pieces of software available on the Web that people can access through a standard protocol and execute remotely. Furthermore, when used together, Web services can deliver a complex functionality. [Petrie and Bussler, 2003] expand on this theme in describing virtual enterprises formed by a sea of services that can be discovered, composed and invoked, as needed at runtime.

To make this vision a reality, the meaning (semantics) of things that are associated with Web services must be handled. [McIlraith et al., 2001] proposed that to enable Web service automation a fundamental component will be the markup of Web services to make them computer-interpretable, use-apparent, and agent-ready. Semantic Web technology supplies the basis for modeling and describing the meaning of things through the use of ontologies, as defined by [Gruber, 1993], which allows shared notions of concepts to be defined with a machine-understandable semantics. The combination of Semantic Web and Web services technology is referred to as Semantic Web services (SWS).

A body of research has developed that charts out the landscape for Semantic Web services. Several fixed points have been established on this landscape that promote common understanding of concepts and help coordinate the ongoing research. These include:

- The W3C model for Web services architecture [Booth et al., 2004]
- Conceptual models for SWS such as OWL-S [Martin et al., 2004] and WSMO [Roman et al., 2005]
- Languages that define the semantics for these conceptual models such as OWL [Dean et al., 2003]
and WSML [Bruijn et al., 2006]

- Conceptual architectures that identify problem areas SWS architectures need to address including WSMF [Fensel and Bussler, 2002] and SWSA [Burstein et al., 2005b]

- Proposals for solutions to tasks of service discovery, mediation, composition, invocation, etc.

- Concrete designs and implementations such as IRS-III [Cabral et al., 2006], DAML-S Virtual Machine [Paolucci et al., 2003], Meteor-S [Miller and Oundhakar, 2001] and WSMX [Haller et al., 2005]

Missing from the list above is a reference architecture for Semantic Web service systems. All software systems have an underlying architecture whether it is implicit or explicitly declared. [Perry and Wolf, 1992] define software architecture as being concerned with the selection of architectural elements, their interactions and constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and form a basis for the design. They argue that software architecture provides the following benefits (i) a framework for satisfying requirements, (ii) a technical basis for design, and (iii) an effective basis for reuse.

As the architecture for Web service systems is termed Service Oriented Architecture (SOA), I use the expression Semantic Service Oriented Architecture (SSOA) to denote the architecture of a Semantic Web service system. Several independent designs for Semantic Web service systems exist – each representing differing perspectives and design goals. [Endres and Rombach, 2003] observe that, in general, reference architectures provide an abstraction of existing systems, enabling the discussion of the common aspects of implementations from a particular domain, providing an anchor point for ongoing development and future advances. There is no such reference architecture for Semantic Web services, setting out a minimal set of common architectural elements, validating this architecture against existing designs, and verifying its feasibility against independently posed problems.

This thesis sets out to address this gap in the literature by proposing a reference architecture for Semantic Web service systems, and in so doing contributes a missing part to the overall understanding of such systems. The reference architecture is presented both from an abstract birds-eye view and from a practical perspective, through the design, implementation and verification of a concrete system applied to independently set problem scenarios.

1.2 Research Questions

A number of Semantic Web services frameworks exist, each adopting a conceptual model and language formally defining the model’s semantics. Typically, the focus of such frameworks is to provide a prototype
implementation for the conceptual model being used, or for design of components targetting specifically identified problems, such as service discovery, composition or mediation. If a step back is taken to abstract from individual approaches to specific problems, the following question, which is central to this thesis, can be asked:

- **What is the essential set of elements, properties, and constraints on those properties required to define a reference Semantic Service Oriented Architecture?**

The answer to this question is derived as a set of views on different aspects of a Semantic Service Oriented Architecture including details of expected structure and behaviour so that a SSOA can fulfill its requirements. These views provide the template for how the elements of an SSOA relate to each other and the dependencies that they place on one another. Several existing SWS frameworks are available – each with a different emphasis and subset of functionality. If the reference architecture is valid then it should be possible to abstract these systems into a form that can be described in terms of the reference architecture. This leads to the question:

- **Can existing SWS framework implementations be described in terms of the reference architecture, produced to satisfy the first question?**

The implications of the choices made in defining a reference architecture are only brought into the open when a concrete design and realization of that architecture is created, applied, and validated against an independently-defined set of problems. Implementing a concrete design within the constraints of the overall Semantic Web service architectural style forces individual design challenges to be tackled. This leads to the third question:

- **Can the feasibility of a reference architecture for Semantic Web service systems be validated by the design and implementation of a concrete system and its application to a set of independently posed problems? Does this implementation require additional research to address specific design challenges?**

### 1.3 Methodology

In order to develop and validate the reference architecture presented in this thesis, a combination of top-down and bottom-up approaches are applied. To get a perspective on the motivation and intent of Semantic Web service systems, it is necessary to take a step back and look at the bigger picture. The top-down approach looks at such systems as evolving from a mix of distributed, goal-driven, service-oriented, and agent-based systems (where each agent can take the role of service provider, service requester, or
CHAPTER 1. INTRODUCTION

both). While no explicit reference architecture for Semantic Web services exists, several conceptual models and frameworks do exist in the domains of agent systems, Web services, service oriented architecture and in the Semantic Web service domain itself. In comparing and contrasting these approaches, a consensus for the requirements for Semantic Web service systems, and a generic component model emerges. This is proposed by this thesis as a first step. This generic model identifies the main elements of the architecture and indicates inter-relationships between these elements. The second step is to flesh out the model and describe the architecture using a view-based industry standard. Existing SWS systems are revisited, and described in terms of the proposed reference architecture to confirm its validity.

In parallel with the top-down approach, described above, a bottom-up experimental approach is applied to validate, and provide feedback to, the high-level view on the architecture. Demonstrating the feasibility of a reference architecture for Semantic Web services systems, requires that this architecture be instantiated and shown to be able to solve independently specified discovery, mediation and composition problems in various domains.

Achieving this requires the design and implementation of a basic infrastructure providing the plumbing for the components of the architecture (step 1), as well as the design and implementation of individual components identified as necessary by the architecture (step 2). To gain feedback on the usability of such an operational architecture design, it is useful to make it openly available to the research community and apply it to problem scenarios in different domains (step 3). At the same time, it should be tested through its application against independently-specified problems, where the outcome of the tests are verified by peer-review (step 4). The results of steps 3 and 4 provide valuable feedback used to identify design-gaps to be tackled and to reinforce the architectural views (step 5).

The instantiation of the reference architecture described in this thesis is based on the Web service Execution Environment (WSMX), a prototype implementation for the WSMO conceptual model, initially developed collaboratively by members of the WSMX Working Group\(^1\). I was a co-founder of the working group and joint author of its architecture deliverables. This initial design and implementation of WSMX represents step 1 and step 2. The WSMX architecture design was used as a key input for the architecture of the EU integrated project on Semantic Web services – DIP\(^2\) where it was used as the middleware platform to integrate individual components and apply the combined framework to use cases in three different domains. The resultant feedback and evolution of the architectural design of WSMX through DIP, ultimately led to the architecture presented in Chapter 4 and applied implementation described in Chapter 6. This corresponds to step 3.

A true picture of the validity of WSMX is provided by testing it against a set of independently-

---

\(^1\)http://www.wsmx.org
\(^2\)http://dip.semanticweb.org/ (EU FP6 - 507483)
specified problems. The Semantic Web service Challenge\(^3\) provides a test-bed implementing a number of real-world problem scenarios that participants in the challenge are invited to solve. The evaluation of WSMX as a prototype for the reference architecture in this context represents step 4. In the course of tackling the Challenge scenarios, issues arose for automated discovery, choreography and composition of services that required new designs for affected components. These designs are described in Chapter 5 and correspond to step 5.

1.4 Structure

Following this introductory section, the structure of the rest of this thesis is described in the following paragraphs. Part II provides the necessary background material to place the architecture of Semantic Web Service systems in context.

In the first part of Chapter 2, I discuss the historical background to Semantic Web service systems. There are two parallel tracks – distributed computing systems and agent-based systems. The discussion on distributed systems considers the challenges for such systems and in particular, what components are required in the middleware layer to enable these challenges to be resolved. I briefly describe technology milestones, the conceptual and architectural ideas behind them, and the desirable properties for a distributed system. Agent-based systems provide the second major background area for Semantic Web services. The major milestones are identified, focusing on their architectural design, with particular attention paid to the core components. I describe how the link between agent-systems and the distributed-system technology of Web services is made by [McIlraith et al., 2001] and how this led directly to the definition of Semantic Web services.

In the second part of Chapter 2, I review the literature on software architecture design and description. The section starts with the foundations of software architecture including definitions, followed by a description of its motivation and benefits. I then look at four different techniques for modelling software architecture, noting that there are some overlaps. This section sets the context for the architectural model for Semantic Web services presented in the core of this thesis.

In Chapter 3, I describe the traditional Web service architectural model as the basis for Service Oriented Architecture (SOA). Four shortcomings at the heart of SOA are identified. I give a high-level view of the impact of introducing a semantic layer to the SOA model which is illustrated in a sample scenario that is used as a running example throughout the thesis. The final section of Chapter 3 derives the functional and non-functional requirements for a Semantic Service Oriented Architecture from (1) the shortcomings of traditional SOA and (2) the convergence of research into distributed and agent-based

CHAPTER 1. INTRODUCTION

systems described in the background material of Chapter 2.

Part III includes the core research of the thesis. Chapter 4 describes a reference architecture for Semantic Web service systems, based on the requirements set out in Chapter 3 and influenced by the core components for distributed and agent-systems described in Chapter 2. I then describe the motivation for selecting the conceptual model as the basis for describing the architecture’s data elements. A minimal set of architectural elements required for a Semantic Web service system are described and related to each other using a combination of UML component and sequence diagrams. Finally both behaviour and communication models are presented.

In Chapter 5, I describe designs and algorithms for the core functional SSOA tasks of discovery, process mediation, and combined discovery & composition. The design and subsequent implementation of these algorithms in the context of an overall implementation of the SSOA were essential to the experimental evaluation of the architecture. This work was a joint equal collaboration between myself, Tomas Vitvar and Maciej Zaremba and has been the subject of three peer-reviewed publications: [Moran et al., 2007; Zaremba et al., 2007a; Vitvar et al., 2007b].

Part IV focuses on the implementation and evaluation of a concrete SSOA. Chapter 6 provides a detailed description of how a concrete design and implementation, WSMX, of the SSOA reference architecture was applied to a set of independently set software integration problems posed by the SWS Challenge4. The design uses event-based messaging, designed on the basis of using shared memory space technology and demonstrates a bottom-up experimental evaluation of the SSOA reference architecture described in Chapter 4. Some of the content of this chapter is based on joint work published in [Vitvar et al., 2007a].

Chapter 7 compares and contrasts WSMX with two other SWS Challenge participants that most successfully tackled the Challenge problem set through the application of semantic technology. They are the DIANE project5 [Kuester et al., 2007] from the Friedrich Schiller Universitaet (FSU) Jena, and the SWE-ET project, which is a combination of the Glue discovery engine [Valle and Cerizza, 2005] from CEFRIEL6 and the WebRatio framework [Manolescu et al., 2005; Brambilla et al., 2007] from the Politecnico di Milano7, for the discovery, mediation, composition and invocation of Semantic Web services.

Chapter 8 provides related work on existing research into Semantic Web service systems. It describes and compares the architectures the OWL-S Broker [Paolucci et al., 2003, 2004a], Internet Reasoning Service (IRS) [Motta et al., 2003], and METEOR-S [Verma et al., 2005a], representing the existing set

---

5http://hnsp.inf-bb.uni-jena.de/DIANE/ (accessed 10 Mar 2011)
6http://glue.cefriel.it/ (accessed 10 Mar 2011)
of frameworks for discovery, mediation and invocation of Semantic Web services based on the conceptual models of OWL-S, WSMO and SAWSDL. The criteria used for the comparison is based on the data, component, behavioural and communication models of the reference SSOA, described in Chapter 4. Each of the frameworks is evaluated against the functional and non-functional requirements for a Semantic Service Oriented Architecture developed in Section 3.2.
Part II

Background, Motivation & Requirements for a Semantic SOA
2. Background

This Chapter uses the historical context for Semantic Web service systems to identify common architectural patterns and functional components. It also provides a background into the research area of software architecture and sets the context for the architectural model presented in the core of this thesis.

2.1 Historical Context

2.1.1 Distributed Computing

Semantic Web services are a form of distributed system infrastructure. [Coulouris et al., 2005] define a distributed system as one consisting of components located at networked computers that communicate and coordinate their actions only by passing messages. The base motivation for distributed systems is to provide efficient access and usage of resources shared across nodes in a network. Resources are things of value that can be shared among network users. Traditionally, this included computer hardware, database records, data files and services. In fact all resources can be encapsulated into services provided to clients within the network.

The following challenges for distributed systems are identified by [Coulouris et al., 2005]: (i) heterogeneity – can the system handle a variety of protocols for data, process, and communication; operating systems, programming languages and hardware; (ii) openness – can the system be extended in a variety of ways; (iii) security; (iv) scalability – does the system remain effective as the number of users and resources grows; (v) failure handling; (vi) concurrency and (vii) transparency – e.g. access and location transparency.

The rest of this section provides a brief historical description of Distributed Computing as it evolved to Web services and Service Oriented Architectures. In this context, particular focus is placed on the middleware layer of distributed systems. [Schantz and Schmidt, 2002] define middleware as systems software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware. It is the middleware that enables distributed systems to evolve (openness) and coordinate
CHAPTER 2. BACKGROUND

how autonomous components of a system connect and interact (heterogeneity). The reference architecture for Semantic Web services proposed in this thesis is targeted at enabling solutions to heterogeneity and openness. While the application of semantics offers potential solutions to the other challenges, these are outside the scope of work described here.

Remote Procedure Calls (RPC)

RPC [Birrell and Nelson, 1984] is a client-server technology that allows a computer program to call a procedure running on another remote server process. It provides for the characteristic of location transparency as a programmer using a remote procedure does not need to explicitly code the details of the remote connection. The transport mechanism used in RPC is typically a socket connection, abstracting a communication protocol such as TCP or UDP. Using RPC, a client sends a message to a remote server which processes that message and returns a response. In RPC systems, servers describe the processes they make available using an Interface Definition Language (IDL). The IDL file is used to create code for both the client and server (called stubs) that takes responsibility for the creation and management of the remote connection, and the marshalling and unmarshalling of data sent between the client and the server. [Bloomer, 1992] provides a detailed description of IDLs and their usage. In RPC, (1) the service interface refers to the collection of procedures the server makes available to remote clients; (2) clients must know, in advance, the identifier of the remote procedure they want to use. The following paragraphs describe milestones in the evolution of RPC.

NCA The Network Computing Architecture (NCA), outlined in [Zahn et al., 1990], was one of the first distributed computing environments available to program developers, designed in the 1980s by Apollo Computer Inc. and Hewlett Packard (HP, who later acquired Apollo). The NCA was based closely on low-level RPC communication support and consisted of two major sub-systems: a network computing kernel and a network interface definition language and associated compiler. NCA supported a large number of data formats. The NCA IDL compiler outputted C programming language stubs. Interfaces defined using NCA IDL are identified by an interface name that must be known to clients. The significance of NCA is that it was one of the first mainstream applications to explicitly separate the interface from the implementation of remote components and abstract the clients of remote processes from the details of the communication protocols. An implementations of the NCA, called the Network Computing System (NCS), was developed by Apollo and HP. Both NCA and NCS remained proprietary to those companies.
CHAPTER 2. BACKGROUND

**SUN ONC** SUN has released Open Network Computing (ONC) in the same timeframe as NCA/NCS. [Zahn et al., 1990] describes ONC as *a collection of programs and services designed to aid in the development of distributed systems*. Like the NCA, ONC used the RPC technology. It had an interface definition language called Remote Procedure Call Language (RPCL) and a corresponding compiler called RPCGEN. ONC interfaces are identified using a combination of a unique program number and version that must be known to the clients. The RPCGEN compiler takes interface definitions of remote procedures and generates stub code in the C programming language. The resulting stub code also made use of a component called External Data Representation (XDR) [SUN Microsystems, 1987] which provided a uniform data format for transfer between the client and the server. The format supported differences in machine architectures such as 16 bit versus 32 bit, and little endian versus big endian byte ordering.

**DCE** The Open Software Foundation (OSF) was established to create an open standard for the implementation of the UNIX operating system. Apollo and HP contributed their work on NCA/NCS. Along with the openly available components of ONC and contributions from other members of OSF, this led to the development of the Distributed Computing Environment (DCE) [Gunter, 1995] as an answer to the specification of proprietary distributed systems. DCE is an RPC-based specification using several components from both the NCA and ONC. The RPCGEN compiler was adopted and additional modules for security and global uniqueness of servers in a network were added. DCE provided a comprehensive specification for distributed computing that tackled heterogeneity at the platform, communication protocol and data representation level. It allowed for openness by allowing extension through additional functional components and through its scheme to ensure uniqueness of server identifiers across a network. DCE laid the foundation for the distributed object technologies of CORBA and DCOM that followed.

**Distributed Object Computing**

In contrast to RPC, which provides for distributed *procedure-based programming*, RMI provides a distributed *object-oriented* client-server computing model. It allows objects in one process to invoke methods on an object living in another process. As with RPC, RMI provides for location transparency as the object making the invocation can not tell if the object it is invoking is remote or not, or the details of the communication protocols used for remote invocation. In RMI, a remote interface refers to the methods of an object that are available for invocation by other remote objects. [Coulouris et al., 2005] describes how a significant difference between RPC and RMI is that the latter can pass objects as arguments and results of methods. References to remote objects may also be passed. Like RPC, distributed objects in RMI describe the methods they make available using an IDL. Neither RPC or RMI require data media-
CHAPTER 2. BACKGROUND

tion between the client and the server as the IDL compilers ensure that the data is type-safe. Depending on the implementation many different programming languages and standard datatypes are supported by default. In RMI, the client must know the identifier of the remote object it wishes to access. It must also be able to access an object registry that can map the object identifier to a specific instance, allowing for discovery of a predetermined object across a network. The following paragraphs briefly describe three prominent distributed RMI platforms – CORBA, DCOM and Java RMI.

CORBA  CORBA [The Object Management Group, 2002], the Common Object Request Broker Architecture, is a specification for distributed objects, initiated by the Object Management Group (OMG)\textsuperscript{9} in the late 1980s. CORBA 1.0 was released in October 1991. Its goal was to provide a language and platform neutral standard for middleware (both the basic protocols and for standard pieces of infrastructure). In CORBA, Object Request Brokers (ORB) and Object Adapters provide the basic infrastructure for client-server programming. The ORB acts as a message bus that routes remote object method calls from client to server. The ORB is accessed by a client through an Object Adapter which manages the specific issues related to distributed object-oriented computing such as reference counting, object instantiation policies and security. Communication and data marshalling are the responsibility of the ORBs. Communication between ORBs is via an implementation of the General Inter-Orb Protocol (GIOP). The most prominent implementation of this is the Internet Inter-Orb Protocol (IIOP).

Like RPC, CORBA uses an IDL to describe the interfaces of objects that are to be made available across a network. The CORBA specification contains a detailed IDL specification and a set of mappings from IDL to a wide variety of programming languages. This provides for language neutrality without sacrificing type-safety. The mappings are generated by suitable IDL compilers similar in intent to those available for RPC technologies. Dynamic invocation with reduced type-safety is also possible where the stub of a service is located and bound to a client at runtime.

[Grosso, 2003] summarizes the high and low points of CORBA. He argues that CORBA achieved its essential goal – integration across multiple platforms allowing correct and efficient access to legacy data. The standardised programming language mappings enable compile-time type checking. The IIOP is a very efficient protocol and key services such as Naming (look-up of objects based on name or description) and Eventing were provided early. However, Grosso goes on to point out several drawbacks. The early specifications were not detailed enough to ensure vendor interoperability. The IIOP protocol was not specified until 1994 and not implemented until later. Overall, possibly due to the large number of members in the OMG, the specifications evolved slowly. For example up to 1997, there was still no standard for dealing with firewalls. The IDL is similar in complexity to C++ and there remains

\textsuperscript{9}http://www.omg.org

16
a perception that CORBA is a difficult specification to program against. Finally, as with RPC before it, the programming style is brittle with clients being closely coupled to servers through the IDL-based interface stubs. When the interface at object level changes, any client using that object requires a rebuild.

**DCOM**  Microsoft DCOM [Microsoft Corporation, 1998] is a distributed extension to Microsoft Component Object Model (COM) that builds an object remote procedure call (ORPC) layer on top of DCE RPC to support remote objects. A COM server can create object instances of multiple object classes. A COM object can support multiple interfaces, each representing a different view or behavior of the object. An interface consists of a set of functionally related methods. COM clients interact with COM objects by acquiring a pointer to one of a remote object’s interfaces and invoking methods through that pointer, as if the object resided in the client’s own address space. The interaction model between client and server is the same in DCOM as for CORBA. Interfaces are defined using IDL, which is used to generate client and server-side stubs. As with CORBA, dynamic invocation is possible where the stub for a remote object is acquired by a client at run-time resulting in type-checking being carried out as a run-time activity. [Chung et al., 1997] provide a side-by-side comparison of DCOM and CORBA. They point out that, although both share broadly the same architecture for distributed object computing, three significant differences are (1) the fact that DCOM allows multiple interfaces on a remote object while CORBA allows an interface to inherit from multiple other interfaces, (2) how server objects are registered and when proxy/stub/skeleton instances are created, and (3) the fact that DCOM is tightly tied to the RPC protocol while CORBA is not.

**Java RMI**  Java RMI is a distributed object model tailored specifically for the Java environment. According to [Wollrath et al., 1996], its aims were (1) to make it simple for distributed objects to be implemented and used, and (2) that the system should be extensible and maintainable. From a high-level, Java RMI looks very similar to CORBA. The interfaces of distributed objects are specified and then programmatic stubs are generated off those interfaces for use by the client and server respectively. The underlying communication mechanism is of sockets typically over TCP or UDP but this is abstracted from users of Java RMI. With respect to CORBA, the main difference is that the language neutrality of CORBA is sacrificed for the reduced complexity of using a single language and the advantages of having a unified coding framework and platform – the Java framework. An IDL is not required as Java interfaces are used instead. Marshalling and unmarshalling of Data objects, being sent over the wire, is carried out by having those classes extend the Java Serializable class. As with CORBA and DCOM, a Naming service is part of the framework. This allows clients to get a local stub to a remote object based on a unique identifier of the object within the network. Java RMI provides for seamless integration...
for distributed objects in a Java environment, including resource-cleanup of remote objects (garbage collection) and object activation. Other RMI systems support Java through language bindings via the IDL compiler but they do not provide optimized native support for Java features.

Web Services

The distributed computing frameworks of CORBA, DCOM and Java RMI had matured by the end of the 1990s. CORBA offered programming language and platform neutrality, low-latency communications, type-safety, dynamic stub binding, as well as an enterprise-ready suite of support services. Its disadvantages were a perceived complexity and the corresponding required skill levels for developers to get started, the cost of purchasing a proprietary ORB implementation, issues of accessing remote object through enterprise firewalls, and interoperability issues between ORBs from different providers. Java RMI traded complexity for language and platform neutrality, offering similar levels of support services.

The emergence of Web services as an alternative distributed computing platform is described by [Sheth and Miller, 2003] as technical evolution, yet practical revolution. Unlike Java RMI and CORBA, Web services use a remote service model similar to the RPC model of DCE. A stateless model is preferred to maximise the decoupling between Web services although it is possible to implement stateful Web services. Web service interfaces are described using WSDL [Chinnici et al., 2003] and XML. Client side code may be constructed that (1) creates XML messages, sends them to the server, and handles the response or (2) a higher level API may be used based on stubs generated from the WSDL. For (2), the resulting client code complexity is of a similar level to Java RMI or CORBA.

The use of XML as the common language for Web service data exchange solves the problem of syntactic interoperability and abstracts Web service interfaces from programming language specifics. The caveat is that type-safety at system compile-time is sacrificed or as [Gray, 2004] points out, practical details like common data representations are avoided through the use of textual representations. Although this is mitigated if client and server stubs are generated from the WSDL service descriptions.

From the outset, Web services specifications have been driven by open standards and implementations. WSDL, XML and SOAP are simple to learn and get started with, especially in comparison to CORBA. Communication is typically over HTTP which is normally allowed through firewalls. The use of both HTTP and XML text-based messages support increased interoperability but at an increase in run-time cost for Web service solutions as compared with Java-RMI or CORBA solutions. The increase is in the overhead required for handling the HTTP protocol and for the parsing of XML documents although the effects of speed related to XML parsing decrease where larger datasets are being transferred over the wire [Gray, 2004].
2.1.2 Agent Systems

From a high-level perspective, agent systems consist of independent software programs (agents) that each are capable of doing a piece of work. Agents can interoperate with each other, delegate tasks, and cooperate to get work done that otherwise could not be done by a single agent alone. Agent systems use an expressive declarative language to communicate with each other, and to describe their capabilities and requests. In contrast to the RPC and RMI based distributed systems, this means that agents remain decoupled from each other and are resilient to heterogeneous environments where software programs may be written at different times, by different people and using different programming languages. Research into agent systems was carried out in parallel to that for distributed systems. Although both aim to enable the integration of (possibly remote) pieces of software over a network, distributed systems had a more immediate requirement that it be productised and made available to the enterprise software market.

The following paragraphs briefly describe three stepping stones in the research path of agent systems toward Semantic Web services. It starts with the agent-based software engineering coined by [Genesereth and Ketchpel, 1994], progressing through the Open Agent Architectures of [Cohen et al., 1994] and [Martin et al., 1999] respectively. The final paragraph describes how [McIlraith et al., 2001] use agent systems as the basis for their definition of Semantic Web services.

Agent-Based Software Engineering

Another approach to distributed computing was to frame it as a problem of modelling communication and cooperation among autonomous entities. [Martin et al., 1999] identify four components for effective communication: (1) a transport mechanism supporting asynchronous communication, (2) an interaction protocol defining various types of communication exchange, (3) a content language permitting the expression and interpretation of messages between entities, and (4) an agreed-upon set of vocabulary.

[Genesereth and Ketchpel, 1994] identified the increased demand for autonomous software programs to interoperate. Such programs are independently written, possibly in different programming languages. They also point out that the dynamic nature of software environments means that programs can come and go. While CORBA abstracts developers from the underlying programming language, client and server are still coupled together through the interface definition in IDL. If a component interface changes, or a new component is added, subsequent rounds of programming are needed to generate the appropriate stubs and integrate them into the relevant clients to ensure that clients of the components still function correctly.

To solve this agents communicate using an expressive declarative language whose semantics are defined independently of the agents. [Genesereth and Ketchpel, 1994] define that an agent is only an
agent if and only if it communicates correctly in an agent communication language. To achieve this they proposed the ACL language [Neches et al., 1991], consisting of an inner language, KIF, [Genesereth et al., 1992] consisting of a vocabulary of words with formal definitions provided in first-order logic, and an outer language, KQML, [Finin et al., 1994] which provides a linguistic layer to provide context. The use of ACL as a universal language ensures there is a consistent understanding of the semantics of the messages between agents. Its declarative nature supports scenarios where the agent sending the message does not know anything about the interfaces of other agents other than that they support ACL. The corollary of using ACL, or any formal language, is that a specific component is required that implements the semantics of the language, enabling agents to interpret messages sent between them.

[Genesereth and Ketchpel, 1994] oppose the use of direct communications between agents. This is on the basis (1) of cost where the number of agents in the system is large e.g. in an Internet setting; (2) complexity – as every agent must support whatever code is necessary for negotiation with any other agent. To overcome these two disadvantages and to enable loose coupling of agents, they propose the use of facilitators to act as brokers handling ACL messages sent between agents. Facilitators are derived from the concept of mediators defined in [Wiederhold, 1992]. Agents provide the ACL descriptions of their interfaces to a facilitator. Multiple agents may be associated with a single facilitator and facilitators may communicate together to form a type of federation. A sample scenario is that an agent provides its ACL to its local facilitator (F1). Another agent sends an ACL message representing a request to its local facilitator (F2). F2 uses its lookup service to determine if a local agent can satisfy the request. If not, it passes the ACL message to F1 which in turn knows of a suitable agent. The capability of a facilitator to match an agent with an ACL request is dependent on the availability of a suitable logical reasoning engine that can determine if the ACL of the request and those of candidate agents match.

**Open Agent Architecture**

The Open Agent Architecture (OAA) of [Cohen et al., 1994] and [Martin et al., 1999] built on agent-based software engineering to provide an AI-influenced approach to software design that allowed for flexible loosely coupled composition of programming elements. Interoperation and cooperation between agents is provided through one or more facilitators that provide a collection of services to agents in the OAA. [Cohen et al., 1994] defines the Open Agent Architecture and provide a prototype implementation in the context of networked desktop machines and handheld devices. Although their work is closely related to that of [Genesereth and Ketchpel, 1994], they differ in balancing the expressivity of the agent communication language they use, (the prolog-based Interagent Communication Language (ICL)), with the practical constraints of implementing their design. The OAA supports distributed execution of user’s requests, interoperability of applications, addition of new agents and the incorporation of existing
CHAPTER 2. BACKGROUND

OAA works by enabling agents to communicate by using a blackboard [Nii, 2008] controlled by a server process. Blackboard approaches allow multiple processes to communicate by reading and writing data to a common store. The server is responsible for matching agents that can achieve specific goals and for coordinating the flow of communications between agents. When an agent registers with the system, it provides a logical description of its functional capability and details of its natural language vocabulary. An agent can also install a trigger on the blackboard that fires when a particular condition is met.

Agents and Web Services

The proliferation of the Web from the end of the 1990s led to an explosion of services being made available through Web browsers to human users. Users could access service on the Web through the exchange of HTML documents between the browser client and the server. Services either provided some information (e.g. weather forecast) or provided means of initiating that the server take some action (e.g. book train tickets). From the perspective of agent-systems (and artificial intelligence) research, Web services were another type of application to which the fundamentals of agent-system research and design could be applied. The difference being that the Web provided an open standards-driven platform with HTTP, HTML, URIs and XML as core standards for the distribution, identification and sharing of knowledge.

[McIlraith et al., 2001] proposed an agent-independent declarative markup language for Web services that captured the data and metadata associated with Web services along with their capabilities, properties, and interface for execution. The DAML mark-up language [Hendler and McGuinness, 2001] was used to provide a semantic description of Web service interfaces. The vision was that an agent system could exploit the declarative descriptions of services and agents respectively to customize users’ requests for automated Web service discovery, composition and execution. [McIlraith et al., 2001] used the agent-programming language ConGolog to encode reusable, sharable generic procedures. These procedures were described using DAML and were capable of invoking underlying Web services, depending on user input and constraints, via interoperation with the Open Agent Architecture (OAA) described above.

The implicit architectural style is one of model-driven programming. A user request is mapped to a generic procedure encoded using ConGolog. The model is composed from (1) the DAML descriptions of Web services relevant to the generic procedure, (2) an encoding of situation calculus relevant to the generic procedure, and (3) the user constraints for the specific request. The program is the ConGolog procedure itself and is executed using a specialized ConGolog interpreter which can communicate with OAA to provide the means for actually grounding the declarative API to the invocation of specific Web services. The significance is that the worlds of agent-systems, AI and Web services are linked.

At this point, Web services were still considered to be applications made available as HTML Web
CHAPTER 2. BACKGROUND

pages. As XML, SOAP and WSDL became the dominant standards for Web services, [McIlraith and Martin, 2003] (amongst others) noted that neither WSDL (describing the service interface) or XML (describing the data types within messages) had formal semantics. The DAML-S ontology [Martin et al., 2004] evolved from earlier efforts to provide an ontology to declaratively describe Web services. DAML-S descriptions could be grounded to WSDL allowing infrastructure to be developed for discovery, composition and execution, that could be linked to the invocation of Web services directly through WSDL descriptions rather than through, for example, the Open Agent Architecture. This led to the development of Semantic Web service broker infrastructures described in the next section.

The application of a semantic meta layer to Web services brought together AI-inspired research with industry-driven standards for the Web from W3C\(^\text{10}\), OASIS\(^\text{11}\) and others. The functionality enabled by the semantic meta-layer had evolved through distributed and agent-driven systems. The notion of agent brokering was defined by [Decker et al., 1996] as the process by which one agent with an objective comes to have that objective achieved by another agent. Agent matchmaking was defined in the same paper as allowing one agent with some objective to learn the name of another agent that could take on that objective. [Paolucci et al., 2004a] provides an analysis of what is required of a Semantic Web service broker. It includes (1) service lookup and matching of client requests to service capabilities, (2) mediation between syntactic data representations, (3) mediation across communication protocols and platforms, (4) mediation between process models, and (5) service invocation. Early examples include the Internet Reasoning System (IRS) of [Motta et al., 2003] and the OWL-S VM of [Paolucci et al., 2003]. Both of these systems are described in more detail in Chapter 8.

2.1.3 Semantic Web Services Architecture (SWSA)

The Semantic Web Services Initiative\(^\text{12}\) Architecture committee (SWSA) describes a high-level architectural framework for Semantic Web services based on abstract characterizations of protocols and functional descriptions of capabilities, as summarised in [Burstein et al., 2005b], and provided in full detail in the SWSA technical report [Burstein et al., 2005a]\(^\text{13}\). Members of the SWSA committee were drawn from industry and from principle Semantic Web service research groups of the time including OWL-S [Paolucci et al., 2003], METEOR-S [Verma et al., 2005a], WSMO [Roman et al., 2005] and IRS [Motta et al., 2003]. Work in the committee appears to have ceased in 2005 with SWSA Version 1.0.

The motivation for the SWSA was to define an interoperability model that can underpin a variety of architectures without prescribing specific implementation decisions [Burstein et al., 2005b]. To do

\(^{10}\)www.w3c.org

\(^{11}\)www.oasis.org


this, SWSA separated the overall process of discovering and interacting with Web services into three main phases and two cross-cutting functional groups. For each of the phases, abstract protocols were developed for accomplishing phase-specific requirements. An ontology is used to define the various messages used in the protocols, using performatives defined by the Foundation for Intelligent Physical Agents (FIPA)\textsuperscript{14} e.g. QUERY, REQUEST etc. The phases identified in the SWSA are (i) candidate service discovery, (ii) service engagement and (iii) service enactment. The cross-cutting functional groups are (i) community support services and (ii) quality of service. Requirements are specified for each phase organised by language, functionality and architecture. A more general set of requirements is specified for the cross-cutting functionality.

2.1.4 Summary

Semantic Web service systems are an evolution of technology with roots in both distributed computing and AI-inspired agent-based software engineering. This section reviewed the development of distributed computing research and agent technology along a timeline through to the emergence of Semantic Web services. RPC provided an abstraction for location transparency of application code. RPC is the basis for RMI used in OO development. CORBA provides a coherent specification for location and platform independent distributed computing with explicit contracts defined for components in the CORBA IDL. DCOM is a related technology from Microsoft. The perceived complexity and cost of using CORBA means that there is a trade-off between its many advantages and the size of software development being undertaken. As the Web developed as a viable platform for distributed computing, Web services emerged as units of application code, distributed over the Web, accessible using standards-based communication and messaging protocols. It is noteworthy that these standards are much less developed than their CORBA equivalents and do not offer the same levels of guarantee for run-time type safety, security, message reliability and so on. However, they provide a relatively low barrier of entry and have been embraced by industry.

Over the same time period, agent-based software system research developed with the objective of enabling self-described software components to automatically cooperate and combine to perform user-specified tasks without the need to pre-program how those tasks would be completed. Different languages were developed to described the capabilities that agents offered and requests that a user of an agent-system could make. The languages have formal semantics and can be unambiguously reasoned over using logical reasoning engines. On this basis, the notion of an agent-facilitator or agent-broker was devised to bring scalability to agent systems. Using a broker, the functions required to support communication and cooperation between agents could be encapsulated rather than requiring each agent to know how

\textsuperscript{14}\url{http://www.fipa.org/specs/fipa00037} (accessed 10 Mar 2011)
to communicate with all other agents themselves. In particular agent-based markup of Web services 
led directly to the research field of Semantic Web services. The broker-based architecture of the OAA 
amongst others is a fore-runner of the Semantic Service Oriented Architecture presented in this work.

The Semantic Web Services Architecture (SWSA) is the result of cooperation between several propo-
nents of Semantic Web services research to provide an abstract model, that can underpin architectures for 
Semantic Web service systems without prescribing detailed design decisions. The abstract infrastructure 
is conceptual without detailed information on components or their interfaces. The focus is on protocols. 
It covers a large number of research questions – some of which are quite complex in their own right e.g. 
automated negotiation, agreement and monitoring of service contracts. In the context of this work, its 
relevance lies in the requirements it identifies for the different phases of discovering and interacting with 
Web services. A potential weakness is that the level of abstraction is too high.

2.2 Software Architecture

2.2.1 Foundations

Definitions

The purpose of defining a software architecture is to provide an abstraction of a software system that 
allows (1) specific characteristics of that system to be easily identified, and (2) the system to be communi-
cated clearly. [Shaw, 1990] described this as hiding some of the details of a system through encapsulation 
in order to better identify and sustain its properties. [Perry and Wolf, 1992] provide a foundational view 
on the discipline of software architecture and lay an intuitive basis for it through comparison with the role 
of architecture in other disciplines including computing hardware and in building construction. Based 
on this they characterize software architecture as being concerned with the selection of architectural ele-
ments, their interactions and constraints on those elements and their interactions necessary to provide a 
framework in which to satisfy the requirements and form a basis for the design. [Fielding, 2000] refines 
this definition to point out that systems may be composed of different levels of abstraction and many 
phases of operation and that each of these can have its own software architecture. Examples of phases 
of operation are set-up, start-up, initialization, normal processing etc. This work is concerned with 
identifying a minimum set of elements necessary for a Semantic Service Oriented Architecture (SSOA), 
and is primarily concerned with the set-up and normal operation phases.
Motivation and Benefits

The overall motivation for defining software architecture is to reduce the cost of software development by reusing design abstractions whose characteristics are well understood and have been proven to be effective in the past. This implies avoiding re-inventing the same designs over and over again for each new system. [Perry and Wolf, 1992] identify architectural erosion and architectural drift as two problems that can arise as a software system evolves. The former implies violation of the architecture. The latter implies changes to the design of the system away from the original architecture, that makes the system more brittle and difficult to adapt.

Having an explicit software architecture in place provides a framework for satisfying the requirements of a particular system - one that can act as the basis of design and reuse of existing design elements [Perry and Wolf, 1992]. [Garlan and Shaw, 1994] describe the benefits for software architecture as including:

1. That the recognition of common paradigms can help high-level relationships to be understood and encourage new systems to be built reusing design components of existing systems

2. That the architectural representation of a system enables identification and analysis of the high-level properties of a system

These are important in the context of this work as a number of conceptual models and implementations for Semantic Web service systems exist without an explicit unifying architecture that identifies common elements and their inter-relationships, as well as the means of connecting these elements.

2.2.2 Modelling Software Architecture

Perry & Wolf Model

[Perry and Wolf, 1992] present a model that defines a software architecture as a set of elements that have a particular form and rationale. Three types of elements are identified: processing, data and connecting. Processing elements supply transformation on data elements. Data elements contain the data that is transformed. Connecting elements are those that connect other elements together. Form provides the constraints for the architecture and is defined by the properties of elements, and the relationships between elements. Rationale captures the motivation for the choice of architectural style. Perry and Wolf also suggest that the use of multiple views and architectural styles in the field of classical architecture is relevant to the description of software architecture.

Architectural Styles

[Shaw and Clements, 1997] provide a means of classification of architectural styles for software systems.
They define architectural style to mean a set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem, together with local or global constraints on the way the composition is done. This conforms to the model of [Perry and Wolf, 1992]. [Garlan and Shaw, 1994] identify a number of common architectural styles including pipe-and-filter, event-based implicit invocation, layered systems, distributed systems, amongst others. They highlight the properties of these styles and the constraints that they impose on a software system. Later in this thesis the architectural styles that apply to the design of a Semantic Service Oriented Architecture are identified to provide a means of communicating characteristics of this type of architecture and for the comparison of existing implementations against a reference architecture.

Architectural Description Languages (ADL)

In parallel to the work on devising models, styles and patterns for software architecture, considerable research took place into defining architectural description languages (ADLs). The motivation for ADLs was to provide formal languages that can be used to represent software architectures and provide the basis for the formal analysis of the properties of those architectures and/or the basis for automated code-generation. [Medvidovic and Taylor, 2000] described how there is little consensus in the research community on what is an ADL, what aspects of an architecture should be modeled in an ADL, and which of several possible ADLs is best suited for a particular problem and that distinction is rarely made between ADLs on one hand and formal specification, module interconnection, simulation, and programming languages on the other. They present a definition and classification-framework and apply this to ten ADLs. As the objective of this thesis is to describe and communicate an architecture for Semantic Service Oriented Architectures, rather than to use a formal description as the basis for code generation and/or formal analysis of such a system, ADLs are of limited usefulness in this context.

Refined Model of Fielding

Based on the existing research into software architecture, [Fielding, 2000] defines a consistent set of terminology to described a software architecture. This definition of a terminology constitutes a model for software architecture. Fielding agrees with Perry & Wolf that architecture is an abstraction of the run-time elements of a software system rather than as a description of a set of system elements. He differs with the model of [Garlan and Shaw, 1994] by explicitly including data elements in the architectural model. Fielding proposes a model consisting of

1. **Elements** – as defined in the model of [Perry and Wolf, 1992] but excluding rationale as Fielding treats this as part of the design documentation
2. **Configurations** on element interaction

3. **Properties** – both functional and non-functional – across all elements of the system

4. **Architectural styles** – as elucidated in [Shaw, 1990] and [Garlan and Shaw, 1994] amongst others

5. **Patterns** – abstracting protocols of interactions between elements

6. **Views** – the different perspectives from which an architecture can be viewed – also originally suggested in [Perry and Wolf, 1992]

In Section 3.2, I identify the architectural principles that support the functional and non functional requirements of an SSOA. Chapter 4 describes a set of views to define a reference SSOA and the component model in that chapter relates to the *Elements* part of Fielding’s refined model.

### 2.2.3 Summary

This section provided definitions of what is meant by software architecture and sets the context for the architectural model presented in the core of this thesis. The section looks at the motivation for research in this area and the benefits that an explicit software architecture description brings. There is a large corpus of research in this area. As the most relevant representative background, the foundational work of Perry and Wolf, and the focus on identification of architectural styles and patterns of Shaw, Garlan and other contemporary researchers at Carnegie Mellon University through the 1990s, is referred to. The topic of ADLs were described and two summary works provided as further reference material. Finally, the work of [Fielding, 2000] is included. This describes the REST network architecture style and contains a cohesive refinement of the model of Perry & Wolf along with aspects of the research into architectural style.
3. Motivation & Requirements for Semantic Service Oriented Architecture

3.1 Motivation: From SOA to Semantic SOA

The design of enterprise information systems has gone through great changes in recent years. In order to respond to business requirements for flexibility and dynamism both monolithic applications and applications that require a heavyweight supporting framework are being challenged by smaller composable units of functionality exposed as services. The drive is toward information systems which adopt the paradigms of Service Oriented Architectures (SOA). With the goal of enabling dynamic and adaptive business processes, SOA builds a service-level view on organizations conforming to principles of well-defined and loosely coupled services - services which are reusable, discoverable and composable.

Figure 3.1: Traditional Web Services Architecture
CHAPTER 3. MOTIVATION & REQUIREMENTS FOR SEMANTIC SERVICE ORIENTED ARCHITECTURE

Figure 3.1 shows the high level architecture for traditional Web service systems. In a typical system, a service provider describes its service using WSDL and optionally publishes this description, including metadata, to a registry. The registry offers a mechanism for querying the details of services published to it. Service requesters use the registry to locate services that match the functionality they require. If a suitable service description is found, proxy code can be generated corresponding to the interface of the service, and can be used to interact with the service using the communication and message protocols specified in the WSDL binding. This is inherited from the earlier interface definition languages of distributed systems, described in Section 2.1.1. All interactions are based on XML descriptions of the services. The semantics of the data and invocation-order of operations at the service's interface are not included in WSDL and are determined either by one or a combination of the following: (1) prior agreement between the requester and provider, (2) adherence to known standards, or (3) by best guess by the service requester.

Service oriented architectures can be further extended to support service composition by using an orchestration language such as WSBPEL [Alves et al., 2006], in cooperation with XML transformation languages such as XSLT [Clark, 1999], and a corresponding run-time infrastructure. WSBPEL processes are composed of a sequence of steps connected by defined control and data flows. Each step can correspond to the invocation of a Web service operation defined using WSDL. Transformation of the XML data between processing steps can be carried out using XSLT. Thus, WSBPEL extends rather than alters the architecture of Figure 3.1.

3.1.1 Limitations of Traditional SOA

A service oriented architecture based on the traditional Web service model, described above, has certain limitations. These are outlined in the following paragraphs.

**Restricted Service Discovery.** A requester has to know registries, contact them individually and query them until it finds a service to match its request. There is no capacity within WSDL itself to describe the capability a service offers. Although registries may enable a service to be categorized using a natural language description, unless the registries support semantic service descriptions and semantic querying functionality, discovery of appropriate services is based on the semantics inferred by the service requester. Automation of the process of discovery and match-making, and its scalability, is impossible if the descriptions of services do not have a formally-defined meaning which is machine understandable.

**Data Mediation is Difficult to Automate.** A service may be found that matches the request but that uses a different data model than that of the requester. The data sent by the requester must be
CHAPTER 3. MOTIVATION & REQUIREMENTS FOR SEMANTIC SERVICE ORIENTED ARCHITECTURE

transformed before it is received by the service. XML is used as the language for messages exchanged between a service requester and provider. The transformation language XSLT is used to transform the XML sent by one party in a Web service exchange to that expected by the other party. Although flexible and extensible, XML is a language for describing the structure and syntax of data. The semantics of data in individual XML elements can not reliably be inferred from the naming convention of XML tags. It must come from outside, either by adherence to standards or some type of agreement between the parties to an integration. Consequently, the creation of an XSLT transformation between the definition of two XML documents is difficult to create automatically and requires effort by a suitable domain expert.

Process Mediation is Manual. For service requesters and providers to integrate, the definition and sequence of the messages for each interaction must match that expected at the receiving endpoint. For example, a receiving service may expect a sequence of (1) a message containing a string followed by, (2) a message containing two strings and an integer. Two messages with these two respective structures, sent in this order, will match both the data and sequencing requirements. However there is no explicit definition of the semantics of the messages in the interaction. The sending and receiving parties must have a shared understanding of the meaning of both the data and the exchange sequence for the integration to be successful. This common understanding cannot be achieved from a WSDL description (an XML format). It must be known by the service requester based on the use of assumed global knowledge, through prior agreement, or through the use of standard protocols. It is typically encoded into fixed relationships between pre-identified services within an integration. As requirements change, the design of this kind of integration must be adjusted and the relationships between the services reset. In software terms, this can mean a cycle of development, integration and testing that relates not only to the new functionality but also to the integration infrastructure as well.

Perspective of Service Provider – Not Requester. The architecture is modelled from the perspective of the Web service provider. During service discovery, it is the requester who must frame their request in terms of the available Web services. Requesters must conform to the Web services’ data and behaviour models. The result is that requesters must think in terms of what Web services they know are available and how to use them, rather than what they want to achieve in terms of their own data and processes. This results in a tight coupling between requester and provider that is typically established at design-time which contradicts a central aim of Web service systems – that of loose coupling.

In a Semantic Service Oriented Architecture, service descriptions are extended to include the semantics
of both data and behaviour. The semantics are defined ontologically, expressed in a logical language with formal semantics which can be enforced by an appropriate reasoning engines, and used to infer information. Supporting semantic descriptions requires extending the traditional web service architecture with a semantic layer. Figure 3.2 shows a high level view of a Semantic Service Oriented Architecture, emphasising that it builds on the traditional architecture of Figure 3.1. Web service providers publish both the semantic and WSDL descriptions of their services to a service registry. The semantic description extends and links to the syntactic description, provided by the WSDL, and does not replace it.

The perspective of the Web service requester is modelled in addition to that of the Web service provider. Requesters provide a goal to describe the capability they desire of a Web service. Semantic Web service descriptions include a description of the capability they offer. The semantic layer is an infrastructure that provides the functionality for discovering Web services to match goals, mediating between mismatching data and behaviour models and choreographing the exchange of messages between the requester and provider. Thus the requester and provider are decoupled allowing the requester to focus on defining what they want to achieve, in terms of goals independently of how they achieve it, through the use of one or more Web services. The semantic layer resolves requester goals by means of
logical reasoning over goals' and services' descriptions. Ultimately, requesters do not need to be aware of processing logic but rather only care about the result and its desired quality.

3.1.2 Scenario for a Semantic SOA

The scenario introduced in this section acts as a running example which will be referred to throughout the dissertation to demonstrate various aspects of a semantic web services architecture. This scenario originates from the Semantic Web Service Challenge\textsuperscript{15}, an initiative which provides a standard set of increasingly difficult problems, based on industrial specifications and requirements.

![Diagram of Semantic SOA Scenario](image)

Figure 3.3: Running Example

As depicted in Figure 3.3, the scenario includes various service providers (Moon, Racer, Mueller etc.) offering various purchasing services for computer-related products. There is a service requester,

\textsuperscript{15}http://www.sws-challenge.org (accessed 10 Mar 2011)
called Blue, with the goal to buy a specific PC for the best possible price. The semantic layer facilitates
the discovery of suitable services and mediates the exchange of messages between Blue and the service
selected to satisfy its goal. The following are prerequisites for the scenario:

- Service requesters and service providers can send and receive messages externally through standard
  Web service endpoints defined using WSDL.

- Both the service requesters and providers use various back-end systems to support their operations.
  These back-end systems define the data and behaviour models used at the respective web service
  endpoints as appropriate. Moon uses a Customer Relationship Management (CRM) system and
  an Order Management system (OMS), each having its own data and process model. Blue uses a
  standard RosettaNet\textsuperscript{16} system.

- Engineers, representing service requesters and service providers respectively, model services and
  requests using an ontological model that supports the concept of a requester goal and the modelling
  of behaviour at both the interface of the goal requester and service provider. Different ontologies
  for data and behaviour are used by the service requester and provider. In particular, Blue sends
  purchase order requests and expects to receive a response according to the RosettaNet PIP3A4
  Purchase Order (PO) specification. Moon, in order to process the request, must perform more
  interactions with its back-end systems. These include identifying a customer in the CRM, opening
  the order in the OMS, adding all line items to the OMS from the request, and closing the order in
  the OMS. Thus, data and process interoperability issues exists between Blue and Moon – Blue uses
  the information model and choreography defined by the PIP3A4 and Moon uses the information
  model and choreography defined by the CRM/OMS systems.

- Both service providers and service requesters publish the ontologies they each use for goal and
  service descriptions to a registry available to the semantic middleware layer. In addition, they are
  responsible for publishing mapping rules between their ontologies and existing ontologies in the
  middleware system.

The scenario runs as follows. Service requesters and providers first model and represent their business
services in terms of a Semantic Web service conceptual model and corresponding language. Blue sends
a purchase order request as a goal to the semantic middleware layer which on its receipt carries out
(1) \textit{discovery}, (2) \textit{selection}, and (3) \textit{choreography of the message exchange}. During discovery, matching
is performed between the required capability (described in the goal) and the capability offered by the
\textsuperscript{16}RosettaNet (http://www.rosettanet.org) is a B2B integration standard defining standard components called Partner
Interface Processes (PIPs), which include standard inter company choreographies (e.g. PIP 3A4 Purchase Order), and
structure and semantics for business messages.
CHAPTER 3. MOTIVATION & REQUIREMENTS FOR SEMANTIC SERVICE ORIENTED
ARCHITECTURE

potential services. There may be semantic differences between the models of the goal and service descriptions. Mediation can be applied to overcome this if mappings between the models exist and have been published. Matching may be blocked if insufficient information is available in the service description. For example, Moon may ship to all European countries but may only supply pricing information on request, rather than in the service description. This information may be obtained by invoking the service interface during discovery so that matching can proceed. During selection, the best service is selected (in this case the Moon service) based on preferences provided by Blue as part of the goal description. Finally the choreography of the exchange of messages between Blue and Moon is managed so that the sequencing of messages each requires is observed. Mediation is carried out here if there is a semantic mismatch between the behaviour or data models and suitable mappings are available.

3.1.3 From Distributed and Agent Systems to SSOA

The architectures of the distributed and agent-based systems described in Chapter 2 have a common objective to link software applications, running in separate processes either locally or remotely, which may use differing data models for their expected inputs and outputs. The timeline presented shows how the technology evolved from the tightly coupled direct RPC calls between processes with strong type-safety to the looser Web service interactions with industry-accepted standards for connection and communication protocols at the cost of weak type-safety. Agent-systems focus on the use of an expressive agent-language with formal semantics, and supporting middleware, to allow collections of autonomous agents to cooperate and interact through sending and receiving messages. They have the core property of loose coupling between agents, supported by the facilitator concept identified by [Wiederhold, 1992], which evolved to the usage of broker-infrastructures providing middleware to enable scalable cooperation between large groups of independent, heterogeneous agents. The principle of brokering is fundamental to an SSOA.

A number of SSOA systems have been implemented: the Web Service Modelling Execution Environment (WSMX) [Moran and Mocan, 2004; Haller et al., 2005] (described in Chapter 6), the Internet Reasoning Service (IRS) [Motta et al., 2003], the OWL-S Broker [Paolucci et al., 2003] and METEOR-S [Verma et al., 2005a], (the latter three are described in the related work of Chapter 8). Additionally, similar conceptual models for a Semantic Web service-based architecture are outlined in the Semantic Web Service Architecture (SWSA) of [Burstein et al., 2005b] (described in Section 2.1.3) and the Semantically Enabled Service Oriented Architecture (SESOA) of [Brodie et al., 2005]. However, a gap remains between these architectural conceptual models on the one hand, and a coherent architectural design independently evaluated against a set of real-world problems on the other. The latter takes a bottom-up
CHAPTER 3. MOTIVATION & REQUIREMENTS FOR SEMANTIC SERVICE ORIENTED ARCHITECTURE

approach and is primarily concerned with proving that a certain class of integration problems can be solved using Semantic Web services and an SSOA implemented using their technology. In contrast, the conceptual frameworks of SWSA and SESOA describe high level requirements and protocols for Semantic Service Oriented Architectures but do not specify sufficient detail of what exactly such an architecture entails and the relationships between its components.

3.2 Requirements for a Semantic SOA

This section is in two parts. In the first part, I identify a set of functional requirements for a Semantic Service Oriented Architecture based on (1) the shortcomings of service oriented architectures using the traditional Web services architectural model (outlined in Section 3.1) and (2) the convergence of the research into distributed and agent-based systems (described in the background material of Section 2.1). In the second part, I briefly describe a set of non-functional requirements that are relevant to the concrete design and implementation of the reference SSOA described in Chapter 6 and relevant to the comparison of this SSOA to the related work of Chapters 7 and 8).

3.2.1 Functional Requirements of SSOA

In this section, each requirement is described briefly, including an example and a brief discussion on why it is a requirement for an SSOA.

Capability-Driven Discovery

Mechanized support is needed in discovering and comparing Web services based on the semantic description of the capabilities they offer. The Semantic Web Services Architecture (SWSA) [Burstein et al., 2005a] describes discovery as the process by which a client (service requester), identifies candidate services to achieve the client’s objectives, by interacting with peers or special purpose middle agents (matchmakers). For example, the Blue Computer Wholesaler of Section 3.1.2 needs to find services having the capability to ship computers, packaged with specific dimensions and weights, to an address in Luxembourg, Europe. Blue creates a semantic description of its goal, including the required capability, and sends it to an SSOA. The SSOA then discovers a set of suitable Web services by semantically matching the goal capability description to the capabilities of Semantic Web service descriptions available to the SSOA.

In the traditional Web services architecture, discovery is a manual process where registries (such as UDDI [Clement et al., 2004]) are designed to be searched by developers – much like searching for a business in a telephone directory. Services are selected based on the implicit inferral of what a service
CHAPTER 3. MOTIVATION & REQUIREMENTS FOR SEMANTIC SERVICE ORIENTED ARCHITECTURE

offers from its name or other metadata that may have been included in the UDDI registry entry. The semantics of the service capability and the data that it expects to send or receive are not precisely defined and rely on the use of naming conventions or a believed shared understanding of the meaning of data between the providers and potential clients of Web services. This imposes limitations on the possibility for automation and scale in expanding the discovery process. Overcoming such problems is possible where an SSOA supports service discovery, based on the use of ontological descriptions to formally define the semantics of goal and Web service capabilities using an expressive logical language (rather than ad hoc natural language). This allows for the clean separation of data (registry) and processing (discovery) to keep the discovery process flexible [Hauswirth et al., 2004]. The requirement for service discovery based on expressive logical description of capabilities is held in common with a the original motivations of the Open Agent Architecture [Cohen et al., 1994] and [Martin et al., 1999] (a technological predecessor of SSOA) discussed in Section 2.1.2.

Selection

The results of the discovery process may yield multiple services that match a given requester goal. These discovered services need to be ranked so that the most suitable of them can be selected according to objectives and additional preference criteria to the advertised capabilities [Burstein et al., 2005a], defined by the service requester. The running example of Section 3.1.2 is continued by assuming that discovery has matched the capability description of goal provided by the Blue Computer Wholesaler’s request with three Semantic Web service descriptions: Moon, Mueller and Racer. In the goal description, Blue has specified a preference for the Web service that can ship the desired product with the lowest cost. Based on this non-functional requirement, the Moon Semantic Web service is selected.

These preference criteria are based on non-functional properties of services, which specify the quality-of-service (QoS) characteristics with respect to parameters that can not be considered functional or behavioural [Garcia et al., 2008]. Examples of parameters are price, security, reliability, geographic location, terms and conditions for payment etc. It is the distinction between the discovery of services based on functional capability, and the selection of services based on non-function characteristics, that leads to the requirement for a selection step to choose the best service from the candidates identified by discovery.

Data Mediation

In a Semantic Service Oriented Architecture, the data that is exchanged between Web service requesters and providers is described using ontologies. It is possible that these ontologies have been developed independently and consequently have different conceptualizations of data in the same domain. Therefore, for an SSOA to support interoperability between service requesters and providers, mechanized support
must be available to translate the semantic content of data in the messages used in their interactions. This leads to the requirement for data mediation. It should be dynamic in the sense that once the ontologies being used by the requester and Semantic Web service are identified, translation may take place automatically, if a predefined mapping, or mechanism for translation between the two ontologies exist.

Returning to the running example of Section 3.1.2, the Blue Computer Wholesaler uses the RosettaNet\textsuperscript{17} standard to define the messages it sends and receives when making orders. The Moon PC Manufacturer, on the other hand uses the SAP R/3 Enterprise Resource Planning (SAP ERP) Purchase Order protocol\textsuperscript{18}. Both standards are defined ontologically and, as they are common standards for business-to-business (B2B) processes, a publicly available set of mappings between the two ontologies exists. At runtime these dynamic data mediation using these mappings is applied to overcome the semantic differences between the messages exchanged between Blue and Moon.

Data mediation has been recognised as a requirement for Semantic Web service systems and the related research area of agent systems. For example, strong mediation is one of the two fundamental principles of the Web Services Modelling Framework, a conceptual model for Semantic Web services [Fensel and Bussler, 2002]. The authors contend that a strong mediation service enables anybody to speak to anybody in a scalable manner. This is important because it frees Web service requesters and providers to design their goal request or service advertisement using their own ontology without needing to know with whom (or with what ontologies) they will need to interoperate in advance. Data mediation is also an aspect of the facilitator concept of the Open Agent Architecture (OAA) [Cohen et al., 1994; Martin et al., 1999] (described in Chapter 2). The requirement to link agents via a facilitator in OAA stems from an opposition to the use of direct communication between agents on the basis of cost and complexity.

**Choreography Interpretation and Execution**

The W3C Glossary [Haas and Brown, 2004] defines a Web Service Choreography as concerning the interactions of a Web service with its users, while the W3C Choreography Model Overview [Burdett and Kavantzas, 2004] defines an information model of the sequence and conditions in which messages are exchanged between all parties in a Web service interaction. In the context of Semantic Web services, both requesters and providers make available public interface descriptions, in which they define how they can be interacted with, in terms of the content and sequencing of messages they can send and receive. This aligns with the W3C definitions above and so we can consider the public interfaces of Semantic Web services to be the choreographies that they implement.

\textsuperscript{17}http://www.rosettanet.org (accessed 10 Mar 2011)
Web service requesters and providers as *choreography* interfaces. Each of these is described using a formal expressive language which enables them to be machine-understandable. For an SSOA to enable interaction between requesters and providers, it must support mechanized interpretation and execution of the choreography interfaces of both parties.

In the running example of Section 3.1.2, both Blue and Moon describe their respective public interfaces, including the content and sequencing of the messages that they expect when sending (Blue) or receiving (Moon) a purchase order. All data that is included in the messages is defined ontologically and the sequence description is defined using a formal process description language. A choreography component of the SSOA uses these machine-understandable descriptions to co-ordinate the interaction between Blue and Moon to complete the purchase order process between them.

One of the motivations for Web services is to enable integration between software systems on the Web. The description of the public behaviour of a Web service as a process is necessary to enable consumers of the service to know how to interact with it. If the description uses a formal expressive language, automated consumers of the process can understand its meaning. The timeline of distributing computing of Section 2.1.1 (from RPC to Web services) looked at how the technology evolved to enable the abstraction of the private internal data and processes of systems from the public data and processes that they expose. This separation of public and private process enables the decoupling of internal system implementation from the public interface required for integration with other systems. The importance of explicitly separating the public and private process is also articulated by [Bussler, 2001] in the context of inter-enterprise process execution where public processes are described as a *means of externalization of enterprise behavior for management issues like advertisement and discovery*.

**Process Mediation**

Both Web service requesters and providers have public interfaces which they use to send and receive messages. For a service requester and provider to be able to interoperate, after they have been matched, the process models described at their respective interfaces must be compatible – or mediated to be compatible. As the Web is by its nature heterogeneous, it is likely that independent service requesters and providers will use different process models. Therefore, process mediation (in addition to data mediation) is required for an SSOA to enable interoperability.

Continuing the running example, the RosettaNet B2B standard used by the Blue Computer Wholesaler expects an acknowledgment message to be sent back to Blue for each message sent in its interaction with Moon. All data required as content of each acknowledgment message is available from the previous Blue message. However, the SAP PO protocol, Moon uses, does not support the sending of acknowledgment messages. Process mediation is required between Blue and Moon to generate the miss-
Others have noted the need for process mediation. [Fensel and Bussler, 2002] identify process mediation as being required to potentially overcome mismatches in three areas (1) business logic, (2) message exchange protocols, and (3) dynamic service invocation. business logic mismatches between the interaction patterns of service requesters and providers. [Burstein et al., 2005a] see the need for mediation, specifically between B2B protocols. [Williams et al., 2005] frame the requirement as protocol mediation where there is a shared conceptual model of the intent and purpose of the communication, but where the mechanics of communication interaction vary. They use an abstract example of two processes X and Y wishing to communicate but using two different protocols P1 and P2. For effective communication to occur between processes X and Y, mediation between the two protocols must occur in the intersection of the capabilities of the two protocols. If the interaction between X and Y requires either one or both of behaviours to be changed, then a different type of mediation is needed. The work in [Williams et al., 2005] is inspired by earlier work, including that of [Bochmann, 1983] and [Calvert and Lam, 1989] on mediating between transport-layer communication protocols between computers networked together.

**Logical Reasoning Engine**

A logical reasoning engine is required at the heart of the architecture. As the descriptions of (1) Web services, (2) service requester goals, (3) the data they exchange in messages, and (4) the mappings between heterogeneous data models are described ontologically in a Semantic Service Oriented Architecture, a logical reasoner is necessary to enable discovery, data and process mediation, and choreography execution. There are a variety of reasoning engines available, each designed to maximise the benefits of a specific logical formalism. Ideally, a SSOA should support a layer of abstraction allowing concrete SSOA designs to work with with pre-existing reasoning engines. Such a layer would provide a common API to access any reasoner used and a transformation service to transform between the logical language representation used natively by the architecture, and the reasoner-specific input representation.

**Compatible with WSDL-described Web Services**

In the last decade, the Web’s ubiquity has meant that Web services and service oriented architecture have become an industry de facto standard for application integration, driven by the use of open Web standard specifications such as URIs, HTML, XML, WSDL etc. Therefore, a Semantic Service Oriented Architecture must try to ensure the largest possible degree of backward compatibility, so that Web services and Semantic Web services can co-exist on top of the same infrastructure. That means that there must be a direct link between Web service and Semantic Web service descriptions, and that this
must be supported by a Semantic Service Oriented Architecture. This link between the semantic and syntactic layer is called *grounding*, which can be further broken down into (1) *data grounding* and (2) *choreography grounding* [Kopecký et al., 2007a]. The former is concerned with defining and executing the mappings between data defined using XML Schema and ontologies. The latter is concerned with linking the set of messages defined in a WSDL description with the process model defined for the public (choreography) interface of a Web service defined using a formal expressive language.

For example, Blue has ordered a quantity of computers manufactured by the Moon company and wants these machines to be shipped to an address in Luxembourg. A set of WSDL-described shipping services are available including ones from Racer, Mueller, and Weasel. Each of these is provided with a semantic description where the semantic description of the messages of their respective choreographies includes a *grounding* element to link them to the corresponding WSDL documents. An example of this, in the WSML formal language, is provided later in Section 6.3.2.

### 3.2.2 Non functional Requirements for an SSOA

In addition to the functional requirements, there are a number of non-functional characteristics which become relevant when devising a concrete design and implementation of a Semantic Service Oriented Architecture. These are architectural characteristics that primarily emerge from the timeline of distributed computing and agent systems discussed in Section 2.1.1. The characteristics are provided in the numbered list below.

1. **Reusability** so that the SSOA can be used to handle an open set (across problem domains) of service-requester defined goals without the need to modify the SSOA implementation.

2. **Loosely coupled** components with standardized interfaces to (1) ensure the abstraction between interface and implementation, (2) avoid inter-component dependencies, and (3) allow specific SSOA behaviours to be realized by the combination of individual components.

3. **Simple Programmatic Interface** so that SSOA clients must only know a simple interface to communicate with the SSOA system, regardless of the interface protocols used by any Web services the SSOA subsequently invokes, on behalf of a service requester.

4. **Scalability & extensibility** so that (1) the architecture can handle increasing volumes of throughput and (2) if new functional SSOA components are required, they can be incorporated without the need to redesign the SSOA implementation.

5. **Goal-oriented** so that designs for Web service systems can be approached from the perspective of service requesters. In other words, an SSOA must allow service requesters to solve their problems.
by specifying the goals they want to achieve in terms of their own data models. This is contrast to traditional SOA that assumes service requesters will frame their requests in terms of the existing Web service descriptions.

3.3 Summary

This chapter described the traditional Web services architectural model as the basis for service oriented architectures (SOA). Shortcomings, relating to handling heterogeneity, at the heart of SOA were identified which can be addressed through the use of Semantic Web service descriptions and an appropriate execution infrastructure (Semantic Service Oriented Architecture). A high-level view of the beneficial impact of introducing a semantic layer to the traditional Web services model is presented and illustrated through a sample scenario which is used as a running example through the remainder of this thesis. Sets of functional and non-functional requirements are identified, based on the identified shortcomings of SOA and the technology timeline of distributed computing and agent systems described in Chapter 2.
Part III

Core
4. A Reference Semantic Service-Oriented Architecture

This chapter describes a reference Semantic Service-Oriented Architecture (SSOA) derived from the requirements for an SSOA identified in Chapter 3. The chapter’s purpose is to identify and communicate a common understanding of the essential parts of a Semantic Service Oriented Architecture system. The SSOA, presented here, is guided by a set of software architecture principles, and is defined according to the following perspectives: (1) a global view, (2) a data model, (3) a component model, (4) a behavioural model, and (5) a communication model.

The chapter is organized as follows. The guiding principles for the SSOA are identified in Section 4.1. Section 4.2 describes a global view which gives a birds-eye perspective on an SSOA and visualizes the relationships between its major components and external stakeholders. The data model is defined in Section 4.3 by the choice of one of an existing set of conceptual models for Semantic Web services. An overview of the features of the model and a short description of the rationale leading to its choice is presented. Section 4.4 presents the component model which describes the functionality and interface of each major component in the SSOA. In Section 4.5, the behavioural model identifies essential internal processes supported by the SSOA that are accessible to external stakeholders via an SSOA public interface. Finally, Section 4.6 discusses the communication model, briefly describing a mechanism suitable for message distribution within the SSOA and how a shared messaging space can be supported to improve the potential for scalability.

4.1 Guiding Principles

There are a number of underlying principles which guide the design of the reference SSOA architecture and its implementation. These principles support the fundamental requirements identified in Chapter 3 and promote seamless integration and provisioning of Semantic Web services. A summary of the
relationship between individual principles and requirements is given in Figure 4.1, while each of the
principles is described in the following sub-sections.

<table>
<thead>
<tr>
<th>Architectural Principles</th>
<th>Service-Oriented</th>
<th>Event-driven</th>
<th>Goal-oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capability-driven discovery</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Service selection</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Dynamic data mediation</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choreography interpretation and execution</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic process mediation</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical reasoning support</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatible with WSDL Web services</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-functional Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reusable across problems and domains</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Loosely coupled with standardised interfaces</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple programmatic interface</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Scalable &amp; extensible</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal oriented</td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 4.1: Summary of SSOA Requirements

4.1.1 Goal-Oriented Discovery and Execution Principle

Goal-oriented means that the purpose of the system is to satisfy goals that service requesters have.
Traditionally service-oriented systems are centered on the capabilities and interfaces that services offer.
They assume that clients will be designed in such a way that they will comply with the various protocols,
prescribed by the service description, for data, process and communications. A goal-oriented system is
one where service requesters can specify what they want to achieve, and delegate the responsibility of
how that can be carried out to the infrastructure. This is one of the central characteristics of the Open
Agent System of [Cohen et al., 1994] and [Martin et al., 1999] (Section 2.1.2). In a goal-oriented system,
it is highly likely that the goal-producer (service requester) and Web service provider use heterogeneous
data and/or process models. Furthermore, a Web service to match a service-requester goal may only be
discovered by the SSOA at run-time (late binding of goal to Web service). Consequently, goal-oriented
systems lead to the requirement for data, process (and possibly protocol), heterogeneity to be overcome
dynamically by the SSOA frameworks through the use of mediation techniques.
4.1.2 Service Oriented Principle

By definition, an SSOA is a specialisation of a service oriented architecture (SOA). The SOA Reference Model [MacKenzie et al., 2006] defines SOA as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. In principle, all components of the SSOA are self-contained services which communicate through the exchange of messages. They are most likely under the control of a single ownership domain (e.g. running in a single process on a single server machine) but this is not a requirement. SOA allows the modelling of application (or system) capabilities as well-defined, independent, invocable, distributable and typically coarse-grained services. Conceptually SOA does not advocate for any particular technology but it prescribes that service interfaces have to be decoupled from the actual implementation. Doing so achieves one of the main technical benefits of SOA – maintaining a loose coupling among integrated components. The benefits of loose coupling include (1) a service implementation can be replaced in a process by another implementation (as long as both implement the same interface), (2) services can be discovered on the basis of the public interface they advertise, and (3) simple services can be composed into a complex service based on their interface definitions.

4.1.3 Semantic Principle

The ability of a semantic service oriented architecture to enable the automation of Web service discovery and execution, including dealing with data and process heterogeneity, is premised on the availability of semantic descriptions. This includes the ontological definition of (1) both desired (service requester) and advertised (service provider) capabilities, (2) the data exchanged in messages between service requesters and providers, and (3) the behaviour of the public interfaces of both Web services and service requesters goals. Such semantics, defined in ontologies, and represented using expressive logical languages, enable logical reasoning engines to be used for the partial or total automation of tasks within the SSOA.

4.1.4 Event Driven Principle

Despite the loose coupling provided for by the SOA style, a requirement remains for further decoupling between components (1) to avoid components being hardwired to the interfaces of other components, and (2) to enable many-to-many communications between components. The event-driven principle [Muehl et al., 2006] leads to an architectural style prescribing that communication between components is on the basis of event notifications, where events are understood to represent changes in the state of something relevant to the system, e.g. a client goal has been received which should be used as the basis for Web service discovery. At runtime, components can publish event notifications to a communication channel.
Other components can register their willingness to consume events by subscribing to the communication channel for specific event types.

In this way components are completely decoupled from each other. Event producers do not need to be aware of the consumers of their events. The result is that the architecture gains the properties of increased (1) flexibility and (2) scalability. Flexibility is improved because components only need to know the interface of the event broker to produce or subscribe for events (or both). Several behaviours can be defined in the SSOA on the basis of the control and data flow of events published and consumed by components. The addition of an additional component in the process can be achieved by publishing an additional event for which that component is subscribed. Scalability is improved, as its possible to add multiple components of the same type to consume events in parallel as the volume of events being produced within the SSOA increases.

4.2 Global View

Figure 4.2: SSOA Overview (Adapted from [Vitvar et al., 2007a])

Figure 4.2 shows an overview of the SSOA architecture, showing the components in the SSOA, the
stakeholders, and types of external applications that interact with it. The SSOA is shown as a node (Machine A) in a networked system connected by a shared messaging space. Goals made available to the shared space that can not be resolved by a local node may be picked up for resolution by other nodes (Machines B and C) in the network. This improves the scalability of the system as additional SSOA systems can be added as the number of available Web services increases. The elements of Figure 4.2 are categorized into the following groups.

- **SSOA Components.** Components providing the functionality required of the SSOA.

- **Stakeholders.** Several types of users of the system.

- **Semantic Modelling.** Applications and tools that support the semantic modelling of goals and Web services.

- **Web Service Providers.** Provision of Web service implementations.

### 4.2.1 SSOA Components

The SSOA components provide the functional blocks of the SSOA and are described in detail in Section 4.4.

### 4.2.2 Stakeholders

Stakeholders interact with the SWS system for several different purposes. Three basic groups of stakeholders are identified: *Goal Owners, engineers, and system administrators.*

**Goal Owners** create goals and access the goal-oriented interface of the SSOA. A goal is a description of something the owner of the goal wants to achieve. It is expected that Goal Owners will be abstracted from the SSOA through specialized domain-specific applications that use their input as the basis for goal generation. For example, one such application may provide the ability to acquire products or services or to place orders through the electronic exchange of information.

**Engineers** form those stakeholders who perform development and modelling tasks relevant to the SSOA. These tasks support the whole SOA lifecycle including service modeling (provision of semantic description), publishing (of semantic description) and the creation of mappings between different domain ontologies. Different types of engineers can be involved in this process including software engineers (service creation and management) and domain experts (ontology and mapping modeling management).
System Administrators  System administrators manage the lifecycle of the components of the SSOA itself. Their responsibilities include component deployment, management and monitoring.

4.2.3 Semantic Modelling

The Semantic Modelling group contains applications and tools to support stakeholders in the formulation of problems as goals and the modelling of Web services using a given Semantic Web service ontology. This group contains goal-creating applications and developer tools.

Goal-creating applications  provide specialized domain-specific functionality and interfaces through which Goal Owners can interact with the SSOA. The interaction between a Goal Owner and the SSOA forms a process that may consist of several messages being sent in both directions (application-to-SSOA and SSOA-to-application). The application takes responsibility for helping the Goal Owner provide the necessary input data and handle the output data received from the SSOA, as appropriate in the course of this process. It may also provide a transformational layer between the SSOA and the Goal Owner. For example, the application may provide a natural language interface, allowing the Goal Owner to present goals as direct speech. For example a goal to buy shares could be presented as: buy $10,000 worth of IBM stock shares if the share price falls below $100. The application would transform the natural language into a goal described in a formal expressive language.

Developer tools  provide specific functionality for the different types of engineers, i.e. domain experts, system administrators and software engineers. The functionality of developer tools covers the whole SOA life cycle including service modeling, publishing, and management. It also includes the modelling of mappings between ontologies that can be used as part of the SSOA function of data mediation. An example of this type of tool is the open-source Web Service Modeling Toolkit (WSMT) of [Kerrigan et al., 2009].

4.2.4 Web Service Providers

Web Service providers represent the various systems that offer some capability through a Web service interface. From a system deployment perspective, it is transparent to the SSOA which physical server hosts particular services as long as there is a network connection, the services support a Web service interface, and the Web service has both a syntactic (WSDL) and a corresponding semantic description (using a Semantic Web services ontology).
4.3 Data Model

According to [Perry and Wolf, 1992], data is one of three core elements of a software architecture (the other two are connecting and processing elements). [Pressman, 1997] describes a data model as the answers to a set of specific questions relevant to any data processing system. These include (1) what are the primary data entities to be processed by the system, (2) what is the composition of each data entity and what attributes does the entity have, (3) what are the relationships between each entity and other entities and (4) what is the relationship between the data entities and the system components that act on them. The data model for a semantic service oriented architecture must provide answers to the questions above for data entities in the domain of Semantic Web services. A particular type of data model is described using a conceptual model, which according to [Brodie and Schmidt, 1982], in addition to answering the questions above, describes the semantics of a specific domain.

4.3.1 Requirements for the SSOA Data Model

The requirements for an SSOA, described in Section 3.2, imply a corresponding set of requirements for the data model. These are briefly listed below.

**Explicit Formal Semantics.** The semantics of all entities of the data model must be explicitly and formally defined in one or more ontologies so that data exchanged within the SSOA is machine-understandable, and can be reasoned over through the use of suitable logical reasoning engines.

**Flexible Logical Language.** It follows from the requirement for explicit formal semantics that the model must be representable in an expressive logical language. Ideally the language should be flexible enough to support semantic modelling using different types of logics.

**Model Service Requester Goals.** Conceptually, the public interface and desired capability of a service requester is encapsulated in the goal they want to achieve. It must be possible to model service requester goals so that the perspective of Web service requesters can explicitly be represented.

**Model Mediation.** A key selling point for Semantic Web services is the enablement of interactions between Web service requesters and providers having possibly heterogeneous data and process models, particularly where they are not aware of each other’s existence beforehand. Therefore it is desirable that mediators be included in the model. Architecturally, mediators are a type of connecting element, one of the three fundamental elements of a software architecture in the model defined by [Perry and Wolf, 1992] and subsequently refined by [Fielding, 2000] (see Section 2.2).
Model as broad a Range of Behaviour as Possible. Modelling Semantic Web services and service requester goals involves modelling the data exchanged at their interfaces and the public behaviour they expose. Ideally, a conceptual model for SSOA must facilitate the modelling of the largest possible set of behaviours.

Backward Compatibility with Web Services. The data structures and interface endpoints for the vast majority of existing Web services are described using XML and WSDL. It is desirable that the data model used for the Semantic Web services of a SSOA be compatible with pre-existing WSDL descriptions.

4.3.2 Existing Conceptual Models for Semantic Web Services

The data elements of an SSOA represent knowledge that is exchanged between the various SSOA components during its run-time operations. (The representation and exchange of knowledge in a Semantic Web service system is the subject of the foundational paper on Semantic Web services of [McIlraith et al., 2001]). The appropriate way to model the data of knowledge-based systems is to use an ontology which as Gruber describes, is a shared specification of a representational vocabulary for a shared domain of discourse [Gruber, 1993]. Therefore, when considering the most appropriate data model to use for an SSOA, I consider the existing ontologies formally specifying conceptual models in this domain.

Four primary conceptual models describing Semantic Web services exist. These are the Web Services Modelling Ontology (WSMO) [Roman et al., 2005], Web Ontology Language for Services (OWL-S) [Martin et al., 2004], the Semantic Web Services Framework (SWSF) [Battle et al., 2005], and Semantic Annotations for WSDL (SAWSDL) [Kopecký et al., 2007b] (formerly WSDL-S [Akkiraju et al., 2005]). Of these WSMO, OWL-S, and SAWSDL are the most prominent. SAWSDL takes a bottom-up approach of embedding semantic annotations in WSDL and is agnostic to the actual semantic modeling language. WSMO and OWL-S, on the other hand, are based on a top-down approach of modelling the semantics of Web services first, and then grounding the semantic descriptions to existing WSDL service descriptions.

4.3.3 WSMO as the Conceptual Model for the SSOA

Based on the criteria in Section 4.3.1, WSMO is chosen as a representative conceptual model for the SSOA reference architecture as it fits very well with the criteria described above. In the next sections, I briefly describe the four top-level elements of the WSMO model (see [Roman et al., 2005] for a complete description), followed by a rationale for its choice. A detailed description of WSMO and the other three conceptual models (and their respective languages) is outside the primary scope of this thesis. However, for completeness, descriptions of each are included in Appendix A for further reference.
• **Ontologies** provide the formal definition of the information model for all aspects of WSMO. Two key distinguishing features of ontologies are, the principle of a shared conceptualization and, a formal semantics (defined by the Web Service Modelling Language (WSML) [Bruijn et al., 2006]).

• **Web Services** are essentially defined by the functional **capability** they offer and one or more **interfaces** that enable a client of the service to access that capability. Service capability is modeled (1) using preconditions and assumptions to define the state of the information space and the world outside that space before execution, and (2) postconditions and effects defining those states after execution. An example of a precondition is that a shipping address contains a valid US zip code. An example of an assumption is that the address actually exists as a destination to which goods can be shipped. Interface descriptions in WSMO have two subelements: choreography and orchestration. A service choreography defines how to interact with the service. A WSMO service orchestration description, on the other hand, is intended to allow the realization of a service to be described as an orchestration of goals or other services linked using WSMO mediators. The WSMO model for orchestration is incomplete as it provides no declarative means to bind a service’s choreography to its orchestration (if an orchestration exists).

• **Goals** describe the objectives a service requester wants to achieve. WSMO goals are described in terms of desired information as well as the state of the world which must result from the execution of a given service. The WSMO Goal is characterized by a requested capability and a requested interface choreography.

• **Mediators** describe elements that aim to overcome structural, semantic or conceptual mismatches that appear between different components within a WSMO environment. WSMO distinguishes between four categories of mediator, ontology-to-ontology (ooMediator), goal-to-goal (ggMediator), Web service-to-goal (wgMediator), and Web service-to-Web service (wwMediator). A detailed description of each of these within the conceptual framework for mediators in WSMO is available in [Stollberg et al., 2006].

4.3.4 Rationale for Choice of the WSMO Model

Figure 4.3 gives a high-level summary of how WSMO, OWL-S and SWSF fit the requirements for the data model. A + symbol denotes the requirement is supported; a +/– symbol denotes that the requirement is supported with some understood drawbacks. A – symbol indicates the requirement is not explicitly supported (although work-arounds are possible). The following points describe the essential factors motivating the choice of WSMO as the most suitable conceptual model to use when describing
a semantic SOA reference architecture. A detailed comparison of the conceptual models of OWL-S and WSMO is provided by [Lara et al., 2004].

- **Flexible Logical Language.** The semantics of WSMO are defined by WSML which is a coherent family of logical languages, with variants layered in terms of the expressiveness each variant supports (see [Bruijn et al., 2006]). The flexibility comes from the possibility of choosing the logical language that best suits the requirements when modelling WSMO elements without having to change syntax. This allows users to make the trade-off between provided expressiveness and the implied complexity for ontology modeling on a per-application basis [Bruijn et al., 2006]. In contrast, for example, although OWL-S provides an ontology to describe different aspects of Web services, it relies on a combination of languages to do this including OWL [Dean et al., 2003], SWRL [Horrocks et al., 2004] and KIF [Genesereth et al., 1992].

- **Explicit Modelling of Goals.** The conceptual model of WSMO focuses on the support for maximal decoupling between Semantic Web service requesters and providers. WSMO explicitly separates the roles of service requester and provider. Goals as well as Web services are first class concepts. This supports the SSOA functional requirement of capability-driven service discovery, composition and invocation. It allows the service-requester’s perspective of a semantic SOA to be about specifying the goal they wish to achieve in their own terms. Goals are not modelled explicitly in either OWL-S or SWSF (although they can be indirectly modelled).

- **Explicit Modelling of Mediators as First-Class Elements.** This reflects the reality that the majority of Web services, and the domain ontologies used to describe the data and processes they expose, will be heterogeneous and autonomous. Semantic SOA systems have the responsibility to broker where possible, the various forms of mismatch that can occur when resolving a service-requester goal. Mediators are a type of connecting element as described in Sections 2.2.2. They also

![Figure 4.3: Comparison of SWS Conceptual Models](image-url)
correspond to the concept of facilitator used in the Open Agent Architecture [Cohen et al., 1994], discussed in Section 2.1.2. Mediators provide explicit modelling elements to overcome mismatches in terms of both data and process. This complies with the SSOA functional requirements of Data and Process Mediation.

• **Flexible Behaviour Modelling.** In WSMO, the public processes exposed by both Web service and goal interfaces are described using their respective choreography elements. Choreography elements use a state-based model inspired by the methodology of Abstract State Machines [Gurevich, 1993]. The choice of ASM as the theoretical basis is motivated by the WSMO desire for a logical formalism that is expressive enough to model any aspect around computation [Roman et al., 2005]. This is supported by the foundational ambition of ASM – *Every algorithm is an ASM as far as the behavior is concerned. In particular the given algorithm can be step-for-step simulated by an appropriate ASM* [Gurevich, 1993]. In principle, a state-based interface where (1) the states the interface can take and (2) the transition rules that can be applied to the interface to cause its state to change, can be expressed with formal semantics allows for the interface itself to be used as part of the service discovery process.

• **Supports Modelling of Multiple Interfaces.** WSMO supports the modelling of multiple interface (and therefore choreography elements). An example of the importance of this is where a Web service supports one interface that supports its described capability, and a second providing non-functional information relating to the capability that is time-sensitive or dependent on input data provided by a potential service requester (e.g. price for a shipping request on a particular date). In contrast, OWL-S allows for only a single service model ontology element for each Web service.

### 4.3.5 Choreography and Orchestration in WSMO

In the WSMO model, the interface element of a Web service description may have two sub-elements: choreography and orchestration. The choreography element is common to both goal and Web service descriptions. It describes the external interface used to communicate with a Web service implementation or the application implementing the message management for a particular goal (e.g. the Goal Generating Application in Figure 4.2).

The orchestration describes how the service itself may implement its capability through a composition of goals and/or other Web services. A visualization of choreography and orchestration is provided in Figure 4.4. The choreography and orchestration elements are modelled on the Abstract State Machine methodology of Gurevich [Gurevich, 1993]. They have a state signature (which describes the state at a
given point in time) and a set of transition rules (which can result in changes to a choreography’s state). The transition rules are defined as logical expressions.

It is common for organizations to model their internal business processes in terms of a discrete sequence of steps connected by control and data flow. In a process language such as WSBPEL [Alves et al., 2006], each of these steps can be a Web service invocation. The requirement for a WSBPEL process to be reusable is satisfied by the fact that it can be wrapped up into a Web service with a WSDL description using the WSBPEL abstract process structure. Enabling a goal to be used at each step of a process that itself is exposed as a Web service is the motivation for the WSMO orchestration element.

Figure 4.4: Choreography and Orchestration of a WSMO Web Service [Roman et al., 2005]

The WSMO model allows for a Web service to have multiple interfaces. Each one of these will have its own choreography. For example, as described later in Section 5.1 (based on joint work at [Zaremba et al., 2007a]), to assist with discovery, a Web service may define a specific data-fetch interface to allow for data to be fetched from a Web service during the discovery process itself.

4.4 SSOA Component Model

The definitions of software architecture of [Perry and Wolf, 1992] and [Fielding, 2000], in Section 2.2.2, both define software architecture as a set of architectural elements including components (processing elements) and connectors (connecting elements). The overview of Figure 4.2 identifies the components required for the SSOA and grouped them into the categories of (1) service bus, (2) platform, (3) repos-
CHAPTER 4. A REFERENCE SEMANTIC SERVICE-ORIENTED ARCHITECTURE

...itories, and (4) broker. This section describes the functionality, interfaces and associations of these components. To provide the big picture, a single diagram showing the associations between all components is given first, followed by the descriptions and interfaces of each component, grouped by category.

The functional requirements for an SSOA, detailed in Section 3.2.1, and the guiding principles described in Section 4.1, provide the basis for the SSOA components described in the following sections.

4.4.1 SSOA Component Associations

The components and their inter-relationships are visualized in Figure 4.5 using a Unified Modeling Language (UML) [Rumbaugh et al., 2005] class-diagram. Lines connecting boxes represent named associations between classifiers. Arrowheads are used to indicate the direction of the association. For example, the Discovery component provides service descriptions to the Selection component.

Figure 4.5: SSOA Component Model
4.4.2 Platform Components

Logical Reasoning

Logical reasoning enables queries that may require logical inferencing to be made over a knowledge base (asserted facts and axioms) defined by ontologies using a defined formal logic model. An essential requirement of the SSOA is to provide flexibility of matching and interaction between service requester goals and Web services that use differing data and process models. This is enabled by providing rich semantic descriptions of the entities involved using a formal logical language. Flexibility requires an engine that can understand the semantic descriptions, including the axioms and relations defined for them, and be able to answer questions on the knowledge those descriptions represent. This is the role of the Logical Reasoning engine. Figure 4.6 shows its interface.

| Operations: | void registerOntology(Ontology anOntology)  
|             | void registerOntologyFromUri(URI anOntologyURI)  
|             | List<OntologyInstances> evaluate (Query aQuery)  |
| Provided to: | Discovery, Data Mediation, Process Mediation, Choreography  |
| Input artifacts: | Ontologies are registered with the reasoner to provide the knowledge base against which queries are evaluated. The ontologies can either be provided directly to the reasoner or the reasoner can load the ontologies from provided URIs. Queries are composed in the logical language supported by the reasoner.  |
| Output artifacts: | On evaluation of a query the reasoner returns a possibly empty list of ontology instances corresponding to the query answer.  |

Figure 4.6: Logical Reasoning Interface

Typically, the use of a particular reasoner is tightly dependent on the formalism used by the semantic model. From the perspective of the architecture, the logical reasoning component is more powerful if it supports an abstraction layer that provides a common interface to interact with a number of different reasoners. An example of this is the WSML2Reasoner\(^\text{19}\) which supports the following reasoners and formalisms: IRIS\(^\text{20}\) (Datalog), KAON2\(^\text{21}\) (Descriptions Logic), and MINS\(^\text{22}\) (Datalog with negation and function symbols).

Service Registry

This represents a registry of semantic service descriptions that have been published to the SSOA. Web Service Providers register semantic service descriptions to the Service Registry component. The Discovery component retrieves service descriptions to use as part of its goal-matching algorithm. Figure 4.7

\(^{19}\)http://tools.sti-innsbruck.at/wsml2reasoner (accessed 10 Mar 2011)  
\(^{21}\)http://kaon2.semanticweb.org (accessed 10 Mar 2011)  
\(^{22}\)http://tools.sti-innsbruck.at/mins (accessed 10 Mar 2011)
describes the interface for the Service Registry.

| Operations: | void registerServiceDescription(Service aServiceDescription)  
|            | void registerServiceFromUri(URI aServiceDescriptionURI)  
|            | List<Service> retrieveAll ()  
|            | Service retrieveService(URI aServiceURI) |
| Provided to: | Discovery, Service Providers |
| Input artifacts: | Semantic service descriptions are registered to the service registry either by provision of the full description or a URI at which the description can be located.  
|                | An individual semantic service description can be retrieved from the registry on provision of its URI. |
| Output artifacts: | A set of all semantic service descriptions or a specific semantic service description. |

**Figure 4.7: Service Registry Interface**

---

**Data Mappings Store**

The Data Mappings Store is a repository of mappings between ontologies. It is used by the Data Mediation component. Responsibility for the creation of mappings can be taken by Web Service Providers, or by the Software Engineers and Domain Experts shown in the Semantic Modelling group of Figure 4.2. The interface for this component is described in Figure 4.8.

| Operations: | void addMappings(URI srcOntologyURI, URI targetOntologyURI, List<Mapping> newMappings)  
|            | void updateMappings(URI srcOntologyURI, URI targetOntologyURI, List<Mapping> updatedMappings)  
|            | void removeMappings(URI srcOntologyURI, URI targetOntologyURI)  
|            | List<Mapping> getMappings(URI srcOntologyURI, URI targetOntologyURI) |
| Provided to: | Data Mediation, Stakeholder providing mappings e.g. Web Service Provider |
| Input artifacts: | srcOntologyURI is a URI identifying the source ontology for the mappings.  
|                | targetOntologyURI is a URI identifying the target ontology for the mappings.  
|                | newMappings are new mappings created between the source and target ontologies.  
|                | updatedMappings are existing mappings that have been updated between the source and target ontologies. |
| Output artifacts: | The getMappings operation returns a list of mappings defined between the given source and target ontologies. |

**Figure 4.8: Data Mappings Store Interface**

---

**4.4.3 Service Bus Components**

The Service Bus provides the message-based infrastructure to facilitate component integration within the SSOA while maintaining loose coupling between those components. It delegates some of this responsibility to four subcomponents that are described later in this section: (1) Communication Manager, (2) Lifting & Lowering Adapters, (3) Event Broker, and (4) Execution Controller.
CHAPTER 4. A REFERENCE SEMANTIC SERVICE-ORIENTED ARCHITECTURE

The Service Bus also provides the integration point between the internal behaviour of the SSOA and its external stakeholders (namely the Goal Owners and Web Service Providers) by defining an external interface for the SSOA via the Communication Manager subcomponent described below. Three essential behaviours of the SSOA exposed by this interface are: (1) goal-driven service discovery, (2) receive incoming data as part of a goal-Web service interaction, and (3) goal-driven service discovery and invocation. The detail on these behaviours is provided in Section 4.5.

Communication Manager

The Communication Manager is responsible for handling communication between the SSOA and external stakeholders such as the Goal Owners and Web Service Providers. It should support common industry access protocols and mechanisms for message exchange (such as XML/HTTP, SOAP/HTTP, JMS) along with relevant security protocols (such as TLS or SSL). The Communication Manager provides a \textit{simple programmatic interface} (see the non-functional requirements of Section 3.2.2) in the form of service endpoints to which (1) Goal Owners can dispatch goals and related data, and (2) Web service providers can send responses to invocations of their service interface. It also provides an interface, used by the Choreography component, for data to be sent from the SSOA as part of a service invocation. Details of the Communication Manager interface is provided in Figure 4.9.

![Figure 4.9: Communication Manager Interface](image)

Lifting and Lowering Adapters

The messages exchanged between internal components of the SSOA are expressed in terms of domain ontologies and represented using a corresponding logical language. WSDL-based Web services provide a
set of operations on their interfaces that exchange XML messages defined by one or more XML Schema. Thus messages being sent from and received by the SSOA must be translated between XML and the relevant semantic language. The term, lowering, is used to describe the translation of data represented in a semantic language to the structural language of XML. Conversely the term, lifting, is used to describe the translation of data from XML to a semantic language. The responsibility for these translations is provided by the Lifting & Lowering Adapters component. Individual adapters are typically, but not necessarily, domain specific and may be created by Web Service Providers at the time the semantic description of the service is created. The interface for this component is given in Figure 4.10.

```
Operations: XMLData lowerSemanticLangToXML(Ontology ontologyFragment, URI uriTargetXmlSchema)
            Ontology liftXMLToSemanticLang(XMLDocument xmlDocument, URI uriTargetOntology)

Provided to: Communication Manager

Input artifacts: ontologyFragment is input data represented as a fragment of an ontology that can contain a combination of concept definitions, concept instances, axioms, constants and non-functional parameters.
                uriXmlSchema is a URI identifying the target XML Schema for the lowered XML data.
                xmlData is input data represented as XML.

Output artifacts: For lowering, the output is XML data.
               For lifting, the output is an ontology fragment in a semantic language.
```

Figure 4.10: Lifting Lowering Adapter Interface

**Event Broker**

The Event Broker is responsible for providing the event-based messaging system at the heart of the Service Bus. Messages sent and received via the Event Broker are normalized and independent of a specific protocol. SSOA components can publish events to the Broker and subscribe to receive notifications for specific events. As all components have an association with the Event Broker, it is not helpful to include it in the component model of Figure 4.5, and thus is omitted to avoid clutter. If the SSOA is deployed as a node in a distributed network communicating over a shared messages space (as suggested in Figure 4.2), then the handling of distributed events also forms part of the Event Broker’s responsibility.

**Execution Controller**

This is a sub-component of the Service Bus which is responsible for controlling the initiation and execution of the internal processes supported by the SSOA. For example, these processes include goal-based Web service discovery and invocation. Client stakeholders of the SSOA request SSOA functionality through the interface of the Communication Manager component. The Communication Manager, in turn uses the interface exposed by the Execution Controller. Once an internal SSOA process is started by the
### Operations:
- void subscribe(EventSubscription eventSubscription, Component componentToCallback)
- void publish(Event event)

### Provided to:
Each component in the SSOA system has the ability to publish events to the Event Broker and to subscribe for event notifications from the Event Broker.

### Input artifacts:
Event subscriptions indicate the event-type that components wish to be notified for. The subscribe interface method requires the provision of the component instance that should be notified if the event being subscribed for occurs. Events have well-defined type and contain a payload which is the message that is to be exchanged between components.

### Output artifacts:
On failure the event subscription or publish operation should throw a programming-language exception.

---

### Execution Controller Interface

Execution Controller, it works in tandem with the Event Broker to co-ordinate the control and data flow required to execute the process. The interface for the Execution Controller is shown in Figure 4.12

### Operations:
- void initDiscoveryBehaviour(Goal theGoal, Ontology inputOntologyFragment)
- void initReceiveDataBehaviour(ContextID theContext, URI serviceOrGoalURI, Ontology ontologyFragment)
- void initAchieveGoalBehaviour(Goal, Ontology inputOntologyFragment)

### Provided to:
Communication Manager

### Input artifacts:
ContextIDs are used to identify a set of interactions between a Goal Owner and a Web Service provider with the purpose of achieving the goal. The serviceOrGoalURI identifies the party sending/receiving data to/from the SSOA as part of an ongoing conversation between a Goal Owner and Web Service Provider for a specific goal. The inputOntologyFragment is input data represented as a fragment of an ontology that can contain a combination of concept definitions, concept instances, axioms, constants and non-functional parameters.

### Output artifacts:
No output

---

### 4.4.4 Broker Components

#### Discovery

The Discovery component is responsible for finding Web services that match the requirements defined in the capability of given goals. In the WSMO model, the semantic descriptions of both goal and Web service descriptions include a capability element. For a Web service, the capability describes what the Web service offers. For the goal, the capability describes what the Goal Owner wants to achieve (see also Section 4.3 describing the WSMO data model). In both cases in SSOA, the individual elements of the capability are described using logical expressions. During the run-time operation of the Discovery component, these expressions are evaluated for both the goal and Web Service description (using the Logical Reasoner). If the semantic descriptions of the goal and Web service use different ontologies, the
Discovery component may need to use Data Mediation to overcome this mismatch. The interface for Discovery is given in Figure 4.13.

<table>
<thead>
<tr>
<th>Operations:</th>
<th>List&lt;Service&gt; discover(Goal theGoal, Ontology inputData)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provided to:</td>
<td>Instances of the Goal Driven Service Discovery Execution process.</td>
</tr>
<tr>
<td>Input artifacts:</td>
<td>The Goal describes the Goal Owner’s requirements in terms of a desired capability expressed in a logical language. The input data is optional and may be provided to Discovery to enable a more refined matching of the Goal against available Web service descriptions.</td>
</tr>
<tr>
<td>Output artifacts:</td>
<td>A list of Web service descriptions whose respective capabilities match that of the Goal.</td>
</tr>
</tbody>
</table>

Figure 4.13: Discovery Interface

Selection

The Selection component is responsible for filtering the set of discovered Web services matching a particular goal down to a ranked list of Web services and then returning the highest ranking of these. Selection is based on the requirements of the Goal Owner. These requirements can be specified (1) as part of the goal description itself using non-functional properties, or (2) using an ontology fragment that defines specific selection criteria e.g. required level quality of service (QoS), cost etc. Data Mediation may be necessary if the ontology used to specify the selection criteria differs from the ontologies used by the discovered Web service descriptions. Figure 4.14 provides the interface description for Selection.

<table>
<thead>
<tr>
<th>Operations:</th>
<th>Service select(Goal theGoal, List&lt;Service&gt; discoveredServices, Ontology rankingOntology)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provided to:</td>
<td>Instances of the Goal Driven Service Discovery Execution process.</td>
</tr>
<tr>
<td>Input artifacts:</td>
<td>The Goal describes the Goal Owner’s requirements. A list of Web services that have been determined to match the Goal. A fragment of a ranking ontology defining the criteria by which the Selection component is to order the Web services.</td>
</tr>
<tr>
<td>Output artifacts:</td>
<td>The highest ranked Web service that matches the given Goal.</td>
</tr>
</tbody>
</table>

Figure 4.14: Selection Interface

Data Mediation

The Data Mediation component is responsible for overcoming data mismatches between Web Service Providers and Goal Owners. All data used within an SSOA environment is defined in terms of concepts defined in one or more ontologies. Ontologies can not always be assumed to be shared in a single domain. If data from two ontologies in the same domain are being used then some form of ontology alignment is
CHAPTER 4. A REFERENCE SEMANTIC SERVICE-ORIENTED ARCHITECTURE

needed. Several approaches to data mediation exist: database schema matching [Castano et al., 2001; Doan et al., 2002] and the survey by [Shvaiko and Euzenat, 2005]; ontology alignment [Choi et al., 2006; Ehrig and Sure, 2004]; ontology merging [Noy and Musen, 2002]; and ontology mapping [Klein, 2001]. In general, mappings are produced which define how an instance from one ontology can be expressed in terms of instances from another ontology. These mappings are represented as logical statements that can be executed by the Logical Reasoner.

The usefulness of data mediation is dependent on the accuracy of the mappings which in turn is dependent on the input of domain experts. There is a trend to provide graphical tools that allow the creation of mappings in a semi-automatic manner [Mocan and Cimpian, 2007]. These mappings are then available for execution by a component such as Data Mediation component as needed. Within the architecture of SSOA, it represents the primary connector element enabling data to be passed between Goal Owners and Web Service Providers who do not necessarily use the same ontology. The interface for Data Mediation is shown in Figure 4.15.

<table>
<thead>
<tr>
<th>Operations:</th>
<th>Ontology mediate(URI srcOntologyURI, URI targetOntologyURI, Ontology srcOntologyFragment)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provided to:</td>
<td>Discovery, Selection, Process Mediation</td>
</tr>
<tr>
<td>Input artifacts:</td>
<td>srcOntologyURI is a URI identifying the source ontology for the mappings. targetOntologyURI is a URI identifying the target ontology for the mappings. srcOntologyFragment is a fragment of the source ontology that contains terms that needs to be mediated to the concepts defined in the target ontology.</td>
</tr>
<tr>
<td>Output artifacts:</td>
<td>A fragment of the target ontology containing the mediated data.</td>
</tr>
</tbody>
</table>

Figure 4.15: Data Mediation Interface

Choreography

<table>
<thead>
<tr>
<th>Operations:</th>
<th>void registerChoreography(Service selectedService, ContextID theContextID) void registerChoreography(Goal goalDescription, ContextID theContextID) void addData(Ontology ontologyFragment, ContextID theContextID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provided to:</td>
<td>Process Mediation, instances of the Goal Driven Service Discovery Execution process.</td>
</tr>
<tr>
<td>Input artifacts:</td>
<td>goalDescription is the semantic description of the given Goal. selectedService is the semantic service description of the Web service selected to match the given Goal. ContextIDs are used to identify a set of interactions between a Goal Owner and a Web Service provider with the purpose of achieving a given Goal. ontologyFragment is input data represented as a fragment of an ontology that can contain a combination of concept definitions, concept instances, axioms, constants and non-functional parameters.</td>
</tr>
<tr>
<td>Output artifacts:</td>
<td>No output</td>
</tr>
</tbody>
</table>

Figure 4.16: Choreography Interface
CHAPTER 4. A REFERENCE SEMANTIC SERVICE-ORIENTED ARCHITECTURE

The Choreography component co-ordinates the message exchange between – (1) the SSOA and Goal Owners, and (2) the SSOA and Web Service Providers – for a given pair of matching goal and Web service descriptions. These descriptions include choreography elements that define the data that each party can send and receive as messages and the control and data flow governing the sequencing of those messages. The Choreography component uses these descriptions to determine what action should be taken when data intended for the Web service or goal is made available to the SSOA. As the data model chosen for the SSOA is WSMO, this section describes the Choreography component in terms of that model. It is important to note that the need for a Choreography component remains valid where an alternate data model such as OWL-S is used.

The Choreography component receives the matching goal and Web service descriptions, creating an internal representation of the choreographies of each. When data is made available to the SSOA from either the Web Service Provider or the Goal Owner, it is passed to the Choreography component. It is then processed in the context of the relevant choreography description through the evaluation of that choreography’s transition rules. Based on the outcome of the evaluation, the component may initiate the sending of messages from the SSOA to a Web service or goal. The interface of the Choreography component is shown in Figure 4.16.

**Process Mediation**

![Mismatch Patterns handled by Process Mediator](image)

The WSMO choreography description of a Web service defines a process consisting of the messages that the Web service is prepared to send and receive, and the ordering of those messages. When a
goal is discovered that matches the Web service’s capability description, the choreography of the goal defines a process that is expected to complement that of the Web service. However, since both the Goal Owner and Web Service Provider are independent and autonomous, the two process descriptions may not exactly match each other either (1) in terms of make-up of the messages or (2) the expected sequence of message exchange. The Process Mediation component is responsible for handling the resolution of potential mismatches between the choreographies of matched goals and Web services. Examples of five types of mismatches that can be identified and resolved automatically through Process Mediation are identified by Cimpian in [Cimpian, 2007]. Figure 4.17 depicts these graphically. These are:

- **Stopping an unexpected message** (Figure 4.17a): When one party sends a message not expected by the other party, the mediator stops the message.

- **Inversing the order of messages** (Figure 4.17b): When one party sends messages in an order different to that which the other party expects, the mediator ensures that messages are supplied in the correct order.

- **Splitting a message** (Figure 4.17c): When a party sends a message which the other party expects to receive in multiple different messages, the mediator splits the message and ensures that all messages are supplied to the receiving party.

- **Combining messages** (Figure 4.17d): When a party expects to receive a message which is sent by the other party in multiple different messages, the mediator combines those messages and ensures that the combined message is supplied to the receiving party.

- **Generating a message** (Figure 4.17e): When one party expects to receive a message which is not supplied by the other party, the mediator generates the message and supplies the message to the receiving party.

<table>
<thead>
<tr>
<th>Operations:</th>
<th>void addData( Ontology ontologyFragment, ContextID theContextID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provided to:</td>
<td>Communication Manager</td>
</tr>
<tr>
<td>Input artifacts:</td>
<td>ontologyFragment is input data represented as a fragment of an ontology that can contain a combination of concept definitions, concept instances, axioms, constants and non-functional parameters. ContextIDs are used to identify a set of interactions between a Goal Owner and a Web Service provider with the purpose of achieving a given Goal.</td>
</tr>
<tr>
<td>Output artifacts:</td>
<td>No output</td>
</tr>
</tbody>
</table>

Figure 4.18: Process Mediation Interface
The Process Mediation component is responsible for managing the flow of data between messages that arrive into the SSOA and the Choreography component. Incoming data received by the SSOA is passed to the Process Mediation component along with a Context ID to identify the specific interaction between a goal and Web service to which the data relates. From a high level view of the SSOA Receive Incoming Data behaviour (described in Section 4.5.2), the state machines described by the choreographies of both the goal and Web service are expected to be advanced by the Choreography component on the arrival of incoming data. This is triggered by the Process Mediation component sending data to the addData operation of the Choreography component twice – once for the Web service and again for the goal. However before each of these messages are sent, the Process Mediation component uses the choreography descriptions of the goal and Web service descriptions to determine if (i) data mediation needs to be applied and (ii) if the data in the message is expected by the process of either choreography at this time or has to be held back (see patterns of Figure 4.17). The interface for the Process Mediation component is shown in Figure 4.18.

**Orchestration**

The role of the Orchestration component is to be able to handle the execution of a Web service which is defined as an orchestration of other Web services or goals. Such a service description must include a choreography defining how the an agent invoking the service can interact with it. Further, the service description must include a means of binding the input and output data, defined for the service in the choreography, with the inputs and outputs specified for the orchestration. However, as pointed out in Section 4.3.5, such a binding is undefined in the WSMO model. For that reason, the specification of an interface for orchestration is omitted from the presented SSOA reference model. This gap in the WSMO data model is a point of interest for future research and work on the SSOA. The following paragraph briefly uses an example to briefly illustrate a use case for orchestration in an SSOA system. I will describe later in Section 8.1 how the Internet Reasoning System of [Domingue et al., 2008] extend the WSMO model to allow for a specific implementation of orchestration in their Semantic Web service engine.

Typically the internal logic of a Web service is private and there is no need for a service orchestration to be defined. However, where the internal logic of a Web service is not required to be private, there can be advantages to using a WSMO orchestration description to allow a single Web service description act as an abstraction for a set of related services. For example, consider a traffic emergency medical response service which consists of several individual steps, whose precise execution is dependent on the location of the emergency. The first step could be to determine the closest medical team; the next to determine traffic conditions between the team and the location of the incident; the next step to notify emergency hospitals in the region to be on standby for an incoming case etc. To model this example
without using an orchestration would require a Web service to be defined with all possible combination
of locations and services for an emergency incident known in advance. Using a Web service defined as
an orchestration of goals provides the possibility for the goals to be resolved to the most relevant Web
services at the time the services are required, providing for increased flexibility at run-time.

4.5 Behavioural Model

The aim of the SSOA is to support automated goal-based Web service invocation. From a high-level
perspective, there are three behaviours that the SSOA supports for this to be achieved. The first is to
be able to match a given goal to a specific Web service. The second is to receive data in the context of
a message exchange between a Goal Owner and a Web Service Provider. The third is a combination of
the first two where a goal is provided to the SSOA, a matching Web service discovered and all message
exchanges carried out so that the capability defined in the goal is realised. Thus three essential processes
must be supported by the SSOA Behavioural Model. These are: (1) goal-driven service discovery, (2)
receive incoming data as part of a goal-Web service interaction, and (3) goal-driven service discovery
and invocation. The first two are described in this section and illustrated using UML sequence diagrams
[Rumbaugh et al., 2005]. The third is a direct combination of (1) and (2).

Each process has a very similar initiation phase. A Goal Owner (or application operating on its
behalf) invokes an operation on the SSOA interface exposed by the Communication Manager component
of the Service Bus. For processes (1) and (3), the Communication Manager generates a Context ID that
will uniquely identify the interaction between the SSOA and the Goal Owner for the provided goal. The
Communication Manager then initiates the creation of an instance of the appropriate SSOA behaviour
in the Execution Controller component of the Service Bus, and associates it with the Context ID. For
process (2), if the incoming data is in XML, the Communication Manager uses the Lifting & Lowering
Adapters component to lift the data into ontological form before initiating the creation of an instance of
the appropriate SSOA behaviour and associating it with a provided Context ID. From this point on for
all processes, the Execution Controller works to co-ordinate the behaviour instance with messages being
exchanged between components via the Event Broker.

4.5.1 Goal-Driven Service Discovery

The Discovery process shown in the sequence diagram of Figure 4.19 starts with a Goal Owner
invoking the \texttt{discoverWebServices} interface operation exposed by the Communication Manager. A Con-
text ID is created and returned to the Goal Owner. The Communication Manager sends a message to
the Execution Controller to create an instance of the goal-driven service discovery process, passing in the goal description and an optional ontology fragment that contains any input data provided along with the goal. The Execution Controller takes over control by publishing events to the Event Broker, to which the various SSOA components subscribed for those events respond.

The first event is for the Discovery component, passing the goal description and the input data. Assuming that this component responds with a non-empty set of Web services, the next event published is for the Selection component, passing the goal, a list of matching Web service descriptions and the ranking ontology to be used. Note that, to keep this description concise, the Event Broker is not shown in Figure 4.19 and further details of event creation are omitted in the rest of this, and subsequent, process descriptions. Selection returns the highest ranked Web service ordered using the terms of the provided ranking ontology. This Web service and goal, associated with a Context ID, are stored by the SSOA in persistent storage.

The next step in the process is for the matching goal and Web service to each be registered with the Choreography component for the given Context ID. The Choreography component maintains the state
of the interaction between the goal and Web service. At registration-time the interaction is in an initial state, as no data has been sent by either Goal Owner or Web Service Provider. After discovery, the Goal Owner may wish to commence the invocation of the Web service by sending data to the SSOA along with the Context ID. This is described in detail in the receive incoming data process below. Ultimately, incoming data in a goal-Web service interaction results in the state of each of the goal and Web service choreographies being updated (for a given Context ID).

4.5.2 Receive Incoming Data

Figure 4.20: Receiving Incoming Data Behaviour

Figure 4.20 shows the sequence diagram for the receive incoming data process. This is used when a Goal Owner or Web Service Provider wishes to send data to the SSOA as part of the invocation of a Web service discovered to match a provided goal. The process is started when the receiveData operation
of the Communication Manager interface of the SSOA is invoked, passing in the SSOA Context ID for which the message is intended, a URI identifying the source goal or Web service (from where the data originated), and the data itself either in XML or as an ontology fragment. If the data is in XML, the Communication Manager requests the Lifting & Lowering Adapters component to lift the data to an ontological form.

The Communication Manager sends a message to the Execution Controller to initiate an instance of the receive incoming data SSOA process, passing the context, the URI identifying the source goal or Web service, and the data in ontological form. This process has just one step defined for the Execution Controller to manage – the publishing of an event for Process Mediation. Subsequently, the Process Mediation component works with the Data Mediation component and the Choreography component for each data input it receives. No further control is required of the Execution Controller.

Data mediation is required if the ontologies used in the goal and Web service descriptions differ. For example, if the ontologies differ and if the incoming data has been sent from the Web service, the data is mediated to transform its representation into terms of the ontology used in the goal description. Once the incoming data is available to the Process Mediation component in terms of both the Web service and goal ontologies, it sends two messages to the Choreography component. The first contains the context, a URI identifying the Web service description, and the incoming data in terms of the Web service ontology. The second contains the context, a URI identifying the goal description and the incoming data in terms of the goal ontology.

On receipt of the incoming data, the Choreography component evaluates the rules declared for each of the goal and Web service choreographies. If any of these rules evaluates to true, the component may send one or more messages to the Communication Manager for onward sending to either the Web service or the Goal Owner. If the Communication Manager receives a message to send data in ontological form to a Web service expecting XML messages, it first uses the Lifting & Lowering Adapter component to lower the data from the ontological form to XML before sending it on.

4.5.3 Goal-Driven Service Discovery and Invocation

This process represents end-to-end goal-driven service discovery and invocation. It consists of combining the goal-driven service discovery process with looping the receive incoming data process until the processing of the choreography elements of both the goal and matching Web service come to an end. The process is initiated by a Goal Owner invoking the achieveGoal interface operation of the Communication Manager, passing in a goal and an ontology fragment containing the initial input data.
4.6 Communication Model

The reference SSOA uses event-based communication between the various components to avoid hard-wired inter-component bindings which would restrict scalability. Each SSOA component is defined to have an associated event type which corresponds to the functionality that component offers. When an SSOA component is started, it subscribes to the Event Broker component (on the Service Bus) for events of its associated type. SSOA components must also be able to request the invocation of other components by publishing events to the Event Broker. To abstract individual components from a requirement to know the details of how to subscribe for, receive and send events, the Event Broker is responsible for providing a layer of *wrappers* which act as adapters between individual components and the eventing mechanism itself.

The Event Broker has the responsibility to interact with whatever transport layer is being used to distribute events. During the SSOA startup, each component subscribes to an event-type template. When events of a particular type are published, any components subscribed to that event type are notified. Events are published via the Event Broker. In the concrete implementation for this SSOA model described in Chapter 6, the actual exchange of events is performed via shared memory (e.g., a Tuple Space such as JavaSpaces [Freeman et al., 1999]) which provides a persistent shared memory space enabling interaction between components without the direct exchange of events between them. This interaction is performed using a publish-subscribe mechanism.
4.7 Summary

In this chapter, I have described a reference SSOA, addressing the requirements of Section 3.2, in terms of a global overview and a set of four models for data, components, behaviour, and communication. The description is semi-formal using UML diagrams where appropriate to illustrate the models. The components identified for the architecture are derived from the middleware components that evolved over the timeline of existing distributed and agent-systems research. I describe the motivation for selecting the WSMO conceptual model as the basis for describing the data elements used in the system. An essential set of architectural components required for a Semantic Service Oriented Architecture are described and related to each other using a UML component diagram and interface descriptions. The elements are grouped into categories (defined as part of the global view) to better understand their functionality and dependencies on other elements. The evaluation of the feasibility of the architecture requires a concrete implementation. Chapter 5 provides details of specific design and algorithms for three key components while Chapter 6 describes the application of a concrete implementation of the reference SSOA against a set of independently specified problems in the domain of Business-to-Business (B2B) integration.
5. Algorithms for Three Key SSOA Features – Discovery, Composition and Choreography

In the introduction of Chapter 1, I emphasized that the description and evaluation of an SSOA reference architecture required a combination of top-down and bottom-up perspectives. The bottom-up approach includes a focus on designing and implementing components of the reference architecture so that independently-set problems, relevant to the integration of heterogeneous software applications, could be solved in a verifiable way. The component designs highlighted in this chapter are chosen for particular attention as they are (1) essential to the verification of the SWS architecture’s feasibility against real-world problems, and (2) required special consideration and experimentation to devise.

Each of the designs has been implemented and used as part of the Web Service Modeling Execution Environment (WSMX) [Moran and Mocan, 2004; Haller et al., 2005] implementation applied to the scenarios presented by the Semantic Web Service Challenge23. The design of the algorithms, development of the Java code, and application and evaluation against the SWS Challenge problem set, is the product of joint work between myself, Maciej Zaremba, and Tomas Vitvar and resulted in four peer-reviewed publications: [Moran et al., 2007; Zaremba et al., 2007a; Vitvar et al., 2007b; Zaremba et al., 2007b], and one book chapter: [Vitvar et al., 2008]. The components, for which I provide algorithms and designs in this chapter are:

- Service discovery – based on [Zaremba et al., 2007a]

- Light-weight service composition – based on [Moran et al., 2007]

- Process mediation between public interfaces – based on [Vitvar et al., 2008]

5.1 Discovery

In many cases Web service providers may not provide sufficient information in their capability descriptions for a concrete match to a given goal to be made. This may be because certain information associated with the service is dynamic and varies frequently over time, or that there may be a competitive disadvantage in publishing the data on the Web as part of the service description. [Keller et al., 2005] present an approach to service discovery, using the WSMO conceptual model, that can handle dynamically-provided information during concrete service matching but are limited to the data provided by the goal owner. This section describes how the definition of a specific data-fetch interface on a Web service allows for dynamic data to be retrieved from the service provider in the course of the service discovery process. It is a design that supports late-binding between goal owners and service providers.

5.1.1 WSMO Approach to Discovery

[Keller et al., 2005] define the core model applied by the WSMO Working Group to semantic Web service discovery. In this model an abstract service description represents a set of related services where a service is defined as the provision of something of value. For example, a shipping company could advertise an abstract service description that covers a set of distinct shipping services between various cities. An abstract service description may be complete but may not necessarily be correct. For example, the shipping company may advertise a service that will ship items from an origination city to a destination city where both are within Europe (description is complete). However there may also be locations in Europe to which the service provider is unwilling to ship (description is not fully correct).

Abstract discovery means matching the required postconditions of a goal to those defined by an abstract service, without taking account of the constraints possibly imposed by input data provided with the goal. It allows for the filtering of candidate services to a smaller more manageable set appropriate for further evaluation, taking account of input data provided by the goal owner. [Keller et al., 2005] label this phase as service contracting. Before the abstract service discovery and service contracting phases can take place, a service requester goal description must be created by finding an existing, suitable, high-level goal and then refining it. Consequently, two additional phases are included in the WSMO service discovery model: (1) goal discovery and (2) goal refinement. A summary of all four phases of WSMO discovery are listed below:

1. Goal discovery: Locate a pre-defined goal that fits the requester desire. The pre-defined goal is an abstraction of the requester desire in a more generic and re-usable form.

2. Goal refinement: The goal is refined taking account of specific information provided by the the
service requester.

3. **Abstract Service Discovery**: Using the capability descriptions of the WSMO goal and available Web service descriptions (capabilities contain the conditions that define the states for before and after execution), Web services that may be able to fulfill the service request are identified. After this phase, there is not yet a guarantee that the capability of matching services will be sufficient for the request.

4. **Service Contracting**: Each identified matched service description is checked for their ability to satisfy the specific constraints defined in the Goal description, possibly taking account of input data. The outcome of this phase is expected to be an ordered set of services that satisfy the service requester’s request.

This model assumes that all information required to match a Web service to a goal is available from its description during the *service contracting* phase. In this section, the approach to this phase is extended with support to dynamically fetch additional required data from Web services at run-time (through invocation). The first three phases of service discovery are assumed to have already taken place. The extension recognises that some service descriptions may be intentionally incomplete for various reasons. For example, some information will be context-sensitive such as pricing and trading-partner arrangements, or information relevant to the description may vary over time. The purpose is to increase the possibility of matching a goal to a Web service whether as a stand-alone service discovery exercise or as part of flexible late-binding of a service-requester’s goal to a service-provider’s description.

### 5.1.2 Algorithm for Discovery with Data-Fetch

This section describes an algorithm relevant to the final phase for Semantic Web service discovery identified above as *service contracting*. It is the result of joint work that has been published at [Zaremba et al., 2007a].

**Matching Relation**

A matching relation is declared as:

\[
  s \leftarrow \text{matching}({\mathcal G}, {\mathcal W}, {\mathcal B_{gw}}),
\]

where \( {\mathcal G} \) and \( {\mathcal W} \) are a goal and a service description respectively and \( {\mathcal B_{gw}} \) is a common knowledge base for the combined information of the goal and service descriptions. The knowledge base may also contain data obtained through fetching data from the service provider in the course of the service discovery.
The result $s$ of the matching function can be: (1) match when the match was found (and all required data is available in $B_{gw}$), (2) nomatch when the match was not found (and all required data is available in $B_{gw}$), or (3) nodata when some required data in $B_{gw}$ is not available and thus the matching function cannot be evaluated.

$G_O$ and $W_O$ are symbols representing the information available from the goal and Web service descriptions respectively. It is assumed that all required information for the goal is directly available in the description $G_O$. The data-fetching step is performed against the service when the matching function cannot be evaluated because some required data is missing (result of match is nodata). After data-fetch, the knowledge base is defined as:

$$B_{gw} = G_O \cup W_O \cup \{y_1, y_2, \ldots, y_m\},$$

where $\{y_i\}$ is the additional data fetched from the service. The construction of the knowledge base is shown graphically in Figure 5.1.

For the implementation of the matching relation, the logical expression for the goal capability description is regarded as a query definition. This query expression is evaluated against the knowledge base $B_{gw}$ for the goal and Web service description under consideration. If the query is evaluated as true by the logical reasoning engine (all variables included in the capability description can be resolved) then
the service is considered to match the provided goal.

Algorithm

The algorithm listed in Algorithm 1 operates on inputs, produces outputs, and uses internal structures as follows:

Input:
- Web service, $W$, with its set of static, ontologically-defined data, $WO$. The rule-base that defines the choreography of the data-fetch interface is denoted as $L$. For each rule $r \in L$, the data associated with the rule effect $r^\text{eff}$ is specified as $r.data$. The rule action $r.action$, with values add, update, and delete means that if the rule is executed, the specified action is performed, i.e. data is added, updated or deleted from the knowledge base $B_{gw}$.
- Goal description $G$ with its associated data $GO$, and $G^\text{eff}$ as its set of specified capability effects.

Output:
- Boolean variable $s$ indicating the result of the matching function between $W$ and $G$. True means match while false means nomatch.

Uses:
- Knowledge base $B_{gw}$ representing the information available to a logical reasoning engine against which the matching function is evaluated.
- Processing memory $M$. This is independent of $B_{gw}$ and is used to handle the processing logic of the data-fetch choreography. This is further explained below.
- Boolean variable modified indicating whether the knowledge base has been modified during an iteration of processing the data-fetch interface.

The knowledge base $B_{gw}$ is initialised with the data provided with the goal and the static data provided with the Web service description. The working memory $M$ is also initialised with a copy of $B_{gw}$. The memory $M$ is used for the processing of the rules specified by the data-fetch interface so that its choreography can execute. These rules may involve adding and removing control information that is not relevant when evaluating a match. This allows that information added to $B_{gw}$ is not unintentionally removed.

Once $B_{gw}$ has been initialised, the first attempt at matching the goal to the Web service is carried out. If the result is nodata then the algorithm fetches data from the Web service via the data-fetch
Algorithm 1 Instance-based Discovery with Data Fetching

1: $B_{gw} \leftarrow G_O \cup W_O$
2: $M \leftarrow B_{gw}$
3: repeat
4: $modified \leftarrow false$
5: $s \leftarrow matching(G, W, B_{gw})$
6: if $s = nodata$ then
7: while get $r$ from $L$: holds($r_{cond}, M$) and $r.data \in G_{eff}$ and not $modified$ do
8: if $r.action = add$ then
9: add($r.data, M$)
10: add($r.data, B_{gw}$)
11: $modified \leftarrow true$
12: end if
13: if $r.action = remove$ then
14: remove($r.data, M$)
15: end if
16: if $r.action = update$ then
17: update($r.data, M$)
18: update($r.data, B_{gw}$)
19: $modified \leftarrow true$
20: end if
21: end while
22: end if
23: until $s \neq nodata$ or not $modified$

interface. For each rule defined in that interface, it checks whether the data in the rule’s effect will provide information referenced by $G_{eff}$ but missing from $B_{gw}$. For example $G_{eff}$ may refer to the price of a desired product which is unavailable in the $B_{gw}$ but for which a rule exists that will result in the price being fetched. The intent is to minimise the calls to the Web service so that only data necessary to complete the match is fetched.

If the data-fetch operations result in new data being added, updated or removed to $B_{gw}$, the $modified$ flag is set to true and another attempt at matching the goal to the Web service is attempted. This cycle ends when no data can be fetched from the interface or the matching function can be evaluated (the result is $match$ or $nomatch$).

5.1.3 Sample WSML Description of a Data-Fetch Interface

A fragment of an ontology defining a purchase quote request, purchase quote response, package and product are shown in Listing 5.1. These are used in the WSML description for a data-fetch interface shown in Listing 5.2. This interface includes two independent rules. The first (line 6) states that on receipt of a purchase quote request, the service will provide a purchase quote response. For this example, it is assumed that the service requester has provided sufficient information with their goal to populate an instance of the purchase quote request. The second rule (line 13) describes how to get a quote for a shipment. This rule will only be used if the requested product is available (determined through the
relation `isAvailable` and the Mueller Web service can ship to the specified address (determined through the relation `isShipped`). Listing 5.3 shows the description of the `isShipped` relation.

```
1 concept Product
2   nfp dc#description hasValue "A product available for purchase" endnfp
3   price ofType decimal
4   available ofType boolean
5 concept Package
6   nfp dc#description hasValue "A package suitable for shipping" endnfp
7   length ofType decimal
8   width ofType decimal
9   height ofType decimal
10  weight ofType decimal
11 concept PurchaseQuoteRequest
12   nfp dc#description hasValue "A request for a quote for a given product" endnfp
13   requiredProduct ofType Product
14 concept PurchaseQuoteResponse
15   nfp dc#description hasValue "A quote for a given product" endnfp
16   package ofType Package
17   product ofType Product
18   totalPrice ofType decimal
```

Listing 5.1: Sample WSML Ontology Fragment for Data Fetch

```
1 interface WS MullerDataFetchInterface
2   nfp interfacePurpose hasValue "discovery" endnfp
3 choreography WS MullerDataFetchChoreography
4 ...
5 transitionRules WS MullerDataFetchTransitionRules
6 /* Rule 1: Request for product quote */
7 for all {?purchaseQuoteReq} with (?purchaseQuoteReq memberOf PurchaseQuoteReq)
8   do add(# memberOf PurchaseQuoteResp)
9 endForall
10 /* Rule 2: Request for shipment quote */
11 for all {?shipmentQuoteReq} with {?reportQuoteResp[package hasValue ?package] memberOf PurchaseQuoteResp and
12   ?reportQuoteResp[to hasValue ?to] memberOf ShipmentQuoteResp and
13   isAvailable(?purchaseQuoteResp) and isShipped(?to, ?package)}
14   do add(# memberOf ShipmentQuoteResp)
15 endForall
16 ...
```

Listing 5.2: Sample Data-Fetch Interface [Vitvar et al., 2007b]

```
1 /* isShipped relation in the common ontology */
2 relation isShipped(ofType mo#ShipmentOrderReq)
```

78
CHAPTER 5. ALGORITHMS FOR THREE KEY SSOA FEATURES – DISCOVERY, COMPOSITION AND CHOREOGRAPHY

/* implementation of the isShipped relation in the Muellers ontology */
axiom isShippedDef
  definedBy
    shipmentOrderReq[mo#to hasValue ?to, mo#package hasValue ?package]
      memberOf mo#ShipmentOrderReq and
    ?to[mo#city hasValue ?city] and
    isShippedContinents(?city, mo#Europe, mo#Asia, mo#Africa) and
    (?package [mo#weightKg hasValue ?weightKg] memberOf mo#Package) and
    (?weightKg < 50)
  implies
    isShipped(shipmentOrderReq).

Listing 5.3: Sample Data-Fetch Interface [Vitvar et al., 2007b]

5.2 Composition

The approach to service discovery can be extended to allow for a simple form of service composition. Where a goal can not be matched to a Web service, an attempt is made to decompose the goal. Each derived sub-goal is attempted to be matched against the available services. If a match is made, the information associated with both the sub-goal and the matching service is added to the overall knowledge base used as the basis for subsequent service matching. Simple goal decomposition is based on separating conjuncted concept instance variables in the logical expression describing the goal’s capability element. For example, Listing 5.4 is a WSML description of a goal with three concept instance variables defined for use in the context of the SWS Challenge.

postcondition
  definedBy
    (?x[name hasValue "Mac Book 13", processorType hasValue intelCoreDuo, processorGHz hasValue ?procGhzX, price hasValue ?priceX, color hasValue Black, hddGB hasValue ?hddGBX, memoryMB hasValue ?memMBX] memberOf Notebook
    and ?procGhzX = 2.0
    and ?memMBX >= 512
    and ?hddGBX >= 80
    and
    ?y[price hasValue ?priceY, resX hasValue ?resX, resY hasValue ?resY] memberOf WebCam
    and ?resX >= 640 and ?resY >= 480
    and
    ?z[name hasValue "Sleeve 13", price hasValue ?priceZ] memberOf Accessory
    and ?price = (?priceX + ?priceY + ?priceZ)
    ).

Listing 5.4: WSML Capability Postcondition for a Composite Goal
In this example, the clauses corresponding to the variables ?x, ?y and ?z at lines 3, 11 and 15 respectively are separated and each is treated as an individual goal capability postcondition.

5.2.1 Algorithm

The algorithm wraps around service discovery and is the result of joint work that has been published at [Moran et al., 2007]. It extends service discovery in two ways (1) iterative goal decomposition (if possible) and (2) updating the knowledge base used for matching with the information for already-matched goals and Web service descriptions until all goals have been matched. The latter means that constraints, defined for services found to match one sub-goal, are maintained when the discovery process is applied to any subsequent sub-goals.

Input:
- The service requester’s goal \( G \).
- A set of candidate Web services \( W_c \).

Output:
- A set of services \( W_m \) found to match part or all of the requester’s goal.

Uses:
- A set of goals \( G_u \) that hold any goals that have yet to be matched – initialized with the service requester’s goal.
- A variable \( m \) that holds the result of each individual discovery call. It can have the values, \textit{match} or \textit{nomatch}.
- A knowledge base \( B_r \) that holds the ontologies and corresponding instance data for each Web service that match the goal \( G \) or any subgoal.
- The ontological information associated with a specific service and goal: \( w_{i(o)} \) and \( g_{i(o)} \) respectively.
- A discovery function \textit{discover()} (see algorithm 1).
- A decomposition function \textit{decompose()} as described above.
- A set of decomposed goal \( G_d \).

\( G_u \) is represented by the abstract data structure of a stack. A stack is a last-in, first-out structure with two operations - push and pop. Push adds an element to, and pop removes an element from the stack.
### Algorithm 2: High Level Algorithm for Composition

1. $G_u.push(G)$
2. $W_m \leftarrow \emptyset$
3. $B_r \leftarrow \emptyset$
4. while not $G_u$.isempty() do
5.   $m \leftarrow nomatch$
6.   $g_i \leftarrow G_u.pop()$
7.   while $m \neq match$ and $w_i \leftarrow W_c.getnext()$ do
8.     $m \leftarrow discover(g_i, w_i, B_r)$
9.     if $m = match$ then
10.        $B_r \leftarrow B_r \cup w_i(o) \cup g_i(o)$
11.        $W_m.push(w_i)$
12.     else
13.        continue
14.     end if
15.   end while
16. if $m \neq match$ then
17.   $G_d \leftarrow decompose(g_i)$
18.   if $G_d \neq \emptyset$ then
19.     for all $g_j \in G_d$ do
20.       $G_u.push(g_j)$
21.     end for
22.   else
23.     $W_m \leftarrow \emptyset$
24.   exit
25. end if
26. end if
27. end while

The algorithm starts by adding the requestor’s goal $G$ to the set of unmatched goals $G_u$ and initialising the set of matching Web services $W_m$. The aggregated knowledge base used for the composition $B_r$ is also initialised. At line 5, a loop through the set of unmatched goals begins. The working variable $m$, used to record if a goal has been matched to a Web service, is initialised to $nomatch$. After a goal $g_i$ has been popped from this list, a nested loop through the set of candidate Web services $W_m$ is started at line 7. The discovery function (as implemented in Algorithm 1) is then called iteratively for the goal $g_i$ and each candidate Web service (line 8). If a match is found, the service is added to the set $W_m$ and its supporting ontologies and instances are added to the knowledge base $B_r$. If no match, the loop continues with the next Web service in $W_m$.

If the result of discovery is $nomatch$ at the end of the nested loop, the decomposition function, described above, attempts to break the goal into sub-goals (line 13). If sub-goals are available they are pushed onto the stack of unmatched goals $G_u$ (line 16). If the goal can not be decomposed, the set $W_m$ is reset to the null set and the algorithm exits. This is to ensure that the result of the composition is $nomatch$ unless all clauses of the original goal $G$ can be matched. If the result of the nested loop is $match$, then the outer loop continues until all goals in the set $G_u$ have been processed.
During the composition process, the knowledge base $B_r$ maintains the ontologies, instance values and constraints for each of the services and goals already matched. This knowledge base, along with the goal and Web service, provide the input for each call to the discovery function. This is important as the matching of a service to an earlier sub-goal may add constraints to the knowledge base affecting the matching of subsequent sub-goals. At the end of the process, the set of Web services $W_m$ will either be a null set or will contain one or more Web services that satisfy the postcondition of the server-requester’s goal.

**Note:** A change is required to the algorithm for discovery to enable its usage for composition. Line 1 of Algorithm 1 which reads:

$$B_{gw} \leftarrow G_O \cup W_O$$

should be replaced with the line below:

$$B_{gw} \leftarrow G_O \cup W_O \cup B_r$$

The difference is the inclusion of $B_r$. This is the knowledge base built up as each service taking part in the composition is discovered.

### 5.3 Process Mediation

Section 4.4.4 describes the requirement for process mediation where two complimentary partial process models, describing the expected message exchanges for the goal owner and service provider respectively, do not match each other. Five types of mismatch, identified by [Cimpian, 2007] are listed. These are repeated below:

a. **Stopping an unexpected message.** When one party sends a message not expected by the other party, the mediator stops the message.

b. **Inversing the order of messages.** When one party sends messages in an order different to that which the other party expects, the mediator ensures that messages are supplied in the correct order.

c. **Splitting a message.** When a party sends a message which the other party expects to receive in multiple different messages, the mediator splits the message and ensures that all messages are supplied to the receiving party.

d. **Combining messages.** When a party expects to receive a message which is sent by the other party in multiple different messages, the mediator combines those messages and ensures that the combined message is supplied to the receiving party.
e. **Generating a message.** When one party expects to receive a message which is not supplied by the other party, the mediator generates the message and supplies the message to the receiving party.

This section provides a concrete design for process mediation for a semantic service oriented architecture, which can handle patterns (a) to (d). It is the result of joint work that has been published at [Vitvar et al., 2008]. In this design, pattern (e) is not catered for as additional out-of-scope information would be required by the system to determine how and when to create missing messages. The design spans the two components listed below (for an overview of the relationship between components of the SSOA see Figure 4.5).

1. The **process mediation** component which accepts incoming messages and determines if data mediation is required before dispatching messages to the choreography engine component.

2. The **choreography engine** component which enacts the control flow defined in the respective goal and service choreography descriptions (using a logical reasoner), and then dispatches messages to the communication manager, to be sent out to the goal owner or service provider as appropriate.

Part 1 of the design is defined by the sequence diagram of Figure 4.20 and described in Section 4.5.2. If data mediation is required then it is assumed that the data mappings for this have been defined and are available. Part 2 is implemented by the choreography engine component. It is assumed that a logical reasoning engine capable of evaluating WSML queries is available. Each WSMO choreography description is a *state-based model inspired by the methodology of Abstract State Machines (ASMs)* [Roman et al., 2007]. An algorithm for the operation of the choreography engine is described below.

### 5.3.1 Algorithm for the Choreography Engine

A WSMO choreography describes the public interface of a Web service in terms of an ontologized Abstract State Machine (ASM) (full details of the syntax and semantics of WSMO choreographies are provided in [Roman et al., 2007]). The choreography operates over a knowledge base that is initially populated by any ontology definitions imported in the description. Imported ontologies may include concept definitions, instance data, and axiom definitions. Each WSMO choreography is described using:

1. **A state signature.** This defines the concepts used by the choreography for incoming, and outgoing messages and control elements)

2. **A set of transition rules.** Transition rules describing the allowed state-changes for a WSMO choreography may take one of the following forms:
CHAPTER 5. ALGORITHMS FOR THREE KEY SSOA FEATURES – DISCOVERY, COMPOSITION AND CHOREOGRAPHY

- if Condition then Rules endIf
- forall Variables with Condition do Rules endForall
- choose Variables with Condition do Rules endChoose

Each rule-condition is a logical expression which may involve variables that are bound to instances of concepts when the rule is evaluated. The evaluation of rules can lead to the knowledge base being modified which may in turn lead to a message being dispatched to a Web service or goal owner. In terms of the WSMO ASMs, messages are represented as instances of ontologically-defined concepts. All concepts used in a choreography are declared in its state signature and associated with a mode. Mode in is used for concepts that represent incoming messages for a service (\(W.X.\Sigma_I\) where \(W\) represents the service description, \(X\) represents the choreography element, and \(\Sigma_I\) represents the incoming messages). Concepts with mode out represent outgoing messages (\(W.X.\Sigma_O\)). Concepts with mode static (\(W.X.\Sigma_S\)) represent internal structures used by the choreography for its control flow.

The algorithm below assumes that after a service has been matched to a goal, both choreography descriptions have been registered with the choreography engine. It also assumes that incoming data has been made available to the choreography engine triggering it to evaluate its set of transition rules. The same algorithm holds for execution of goal choreographies and Web service choreographies. Each time an incoming message is received by the SWS system, the information contained in the message is made available to the two registered choreographies. Such a message may be a response to an earlier invocation on a Web service. All incoming messages are handled in the same way regardless of the underlying message exchange pattern (e.g. synchronous or asynchronous). From the system’s point of view, the state of each conversation is maintained by the choreography instances of each of that conversation’s participants.

Uses

- A service (or goal) description, \(W\) including its choreography description \(W.X\) and corresponding set of transition rules \(W.X.L\). Rules have conditions and effects denoted \(r_{cond}\) and \(r_{eff}\) respectively. Conditions are sets of logical expressions made up of concepts that can be bound to instance data at runtime. The effects of a rule are a set of actions that occur as a consequence of the rule’s condition holding \(r_{cond}\). Actions can be to add or remove information in the knowledge base.

- A Knowledge base \(KB\) represents an instance of a choreography \(W.X\) initialized with information \(W.O\) – the descriptions of the concepts declared in the state signature and any instances that have been added to the knowledge base for the current conversation.

- A symbol \(A\) representing all actions to be executed as a consequence of each evaluation of the
rules for the choreography. Each element of $A$ has the same definition is a rule effect $r_{\text{eff}}$. $A$ has methods $A.\text{add}$ and $A.\text{remove}$ for adding and removing actions to/from the set.

- The function $\text{send}()$ which initiates the sending of a message out to either a service provider or goal owner corresponding to the choreography description being executed.

- The symbol $W.X.D_{\text{pending}}$ which is used to distinguish the incoming data that triggered the processing of the choreography from any data that is added to the choreography knowledge base as a result of evaluating the rules.

**Algorithm 3** Algorithm for Execution of Choreography Rules

1: processChoreography
2: \{Evaluating rule’s conditions and sending data\}
3: for all $r$ in $W.X.L$ : $\text{holds}(r_{\text{cond}}, KB)$ do
4: \hspace{1em} $A.\text{add}(r_{\text{eff}})$
5: for all $c$ in $r_{\text{cond}} : c \in W.X.\Sigma_I$ do
6: \hspace{2em} $\text{send}(c, W)$
7: \hspace{1em} end for
8: end for
9: \{Performing delete actions\}
10: for all $a$ in $A$ : $a.\text{action} = \text{delete}$ do
11: \hspace{1em} $KB.\text{remove}(a.c)$
12: \hspace{1em} $A.\text{remove}(a)$
13: end for
14: \{Performing add of internal concepts\}
15: for all $a$ in $A$ : $(a.\text{action} = \text{add} \text{ and } a.c \in W.X.\Sigma_S)$ do
16: \hspace{1em} $KB.\text{add}(a.c)$
17: \hspace{1em} $A.\text{remove}(a)$
18: end for
19: \{Performing add of any output data received from service invocation\}
20: for all $a$ in $A$ : $(a.\text{action} = \text{add} \text{ and } a.c \in W.X.D_{\text{pending}})$ do
21: \hspace{1em} $KB.\text{add}(a.c)$
22: \hspace{1em} $A.\text{remove}(a)$
23: end for
24: return $D$

The algorithm evaluates each rule of the choreography by processing its condition and effect. There are four stages:

- **Performing rule conditions and sending data:** Each time the choreography engine is triggered, the conditions in the headers of the rules defined for the choreography are evaluated. Instance data received in the incoming message is bound to variables in the rule headers. If a rule’s conditions evaluate to true, the effects of that rule are added to the set of actions, $A$. For each symbol in the rule’s condition (lines 4-6) defined in the choreography state signature with mode in, a message is
dispatched (to the communication manager where it is sent on to the appropriate end-point).

- **Performing delete actions:** In lines 9-12, all effects with the *delete* action are carried out resulting in the data, specified by these effects, being removed from the knowledge base.

- **Adding instances of internal concepts:** This refers to effects that add data to a knowledge base necessary for the handling of control flow of a choreography e.g. data may be added to reflect the state that a message has been sent to a Web service. For example, the next expected state could be that the response message has been received.

- **Performing add of any data received back from a service invocation:** The choreography can trigger the sending of data to a service. But even if the message exchange pattern for that invocation is a synchronous request-response, the response data must pass through the message-receiving mechanism of the execution environment before it is provided to the choreography engine. Taking a purchase order (PO) request/response scenario as an example. During the evaluation of a choreography’s rule set, an input symbol in one rule’s condition \( c \in \mathcal{W}.X.\mathcal{I} \) triggers the sending of a PO request to a service. The effect of the rule \( r\text{eff} \) states that a PO response be added to the list of actions to be carried out. However this can only happen once the service responds with the corresponding PO response data. This is the purpose of lines 18-22 in the algorithm.

**Evaluation**

The relationship between the concepts defined in the choreography state signature and the actual messages sent over the wire (e.g. XML/SOAP) is defined by grounding. Co-ordinating the enactment of grounding is part of the responsibility of the communication manager component but the grounding definitions themselves are included in the choreography descriptions. Grounding in WSMO involves translating messages from a semantic representation (e.g. WSML) to a representation ready for inclusion in a message sent over the wire (e.g. XML). This is explained in detail in [Kopecký et al., 2006]. The semantics for grounding, in the context of process mediation, are described further in [Vitvar et al., 2008].

In terms of the process mediation patterns of figure 4.17, data mediation ensures that all new data coming from one service is translated to the other’s service ontology. Thus, no matter from where the data originates, it is always available for use by both the service and goal. From the point view of process mediation, data mediation mappings can also handle message splitting (pattern c) and message combination (pattern d). As the mediated data is always added to the knowledge bases for both choreographies, patterns (a) and (b) are handled through the processing of the choreography rules.
particular, the fact that a message will be stopped (pattern a) means that the message will never be used by the choreography because no rule will use it (the message remains in the memory until the end of the algorithm). In addition, the order of messages will be inverted (pattern b) as defined by the choreography rules and the order of ASM states in which the conditions of the rules hold. This means that the algorithm automatically handles the process mediation with the help of data mediation through the rich description of choreographies without the need for a centralized workflow component.

The algorithm results in all data being added to both choreographies, rather than selectively adding only data which could be of potential use, i.e. the data that is useful for evaluating a specific rule’s condition. This is intentional (but may not be optimal) as WSML allows for intentional definitions, axioms, to be included in the ontologies. These form part of the knowledge base for the choreographies, and so incoming data, not specified directly in a choreography rule, may still affect the evaluation of that rule indirectly through such axioms. The evaluation of the potential use of data would thus require a certain amount of logical reasoning that would add to the required processing time and possibly influence scalability. The investigation of this optimization is left open for future work.

5.4 Summary

In this chapter I described algorithms and designs used to implement three key components of the SSOA reference architecture. The three component designs are chosen as they are (1) essential to the verification of the SWS architecture’s feasibility against real-world problems, and (2) required special consideration and experimentation to devise. In each case, novel designs have been presented which have been the subject of independently peer-reviewed publications. The next chapter describes how the prototype implementation of the reference architecture uses these and other components to solve the problems published by the SWS Challenge.
Part IV

Implementation, Evaluation and Related Work
6. SSOA Implementation & Evaluation against Independently-Set Integration Scenarios

In the introduction of Chapter 1, I indicated that the implications of the choices made in defining a reference architecture are only brought into the open when a concrete design and realization of that architecture is created and applied, and validated against an independently-defined set of problems. This is reflected in the third research question addressed by this thesis:

- Can the feasibility of a reference architecture for Semantic Web service systems be validated by the design and implementation of a concrete system and its application to a set of independently posed problems?

The Web Services Execution Environment (WSMX)\textsuperscript{24} [Haller et al., 2005; Haselwanter et al., 2006; Mocan et al., 2006; Vitvar et al., 2007a] is an open-source project\textsuperscript{25} which provides a prototype implementation of the SSOA described in Chapter 4. This Chapter describes how WSMX was applied to the real-world problem scenarios posed by the Semantic Web Services Challenge (introduced in Section 3.1.2). This public initiative is organized around an ongoing series of workshops in which participants apply their research designs against specific published problems. The design and code of each participating entry must be made public and the success of these designs against the published problems is evaluated by peer-review. Special attention is paid to how the different approaches react to changes in the scenarios. Ideally, participating systems should be able to handle modified scenarios without any

\begin{footnotesize}
\begin{itemize}
\item 24\url{http://www.wsmx.org} (accessed 10 Mar 2011)
\item 25\url{http://sourceforge.net/projects/wsmx/} (accessed 10 Mar 2011)
\end{itemize}
\end{footnotesize}
change that would necessitate the system to be rebuilt and redeployed.

WSMX was instantiated (by the WSMX Working Group) as a collaborative open-source effort in the context of the European Union DIP integrated project (FP6 - 507483) and acted as a foundational input to the Organization for the Advancement of Structured Information Standards (OASIS) Semantic Execution Environment Technical Committee (SEE TC). It provides a reference implementation for the conceptual model defined by the WSMO Semantic Web services ontology, including native support for the WSML formal language, and explicit support of both strong mediation and the ontological separation of service requesters and providers. During each iteration since its first design (see [Moran and Mocan, 2004; Zaremba and Moran, 2004]), the set of SSOA components comprising WSMX, have been designed and implemented with increasing degrees of maturity. Examples of specific designs were presented in Chapter 5. Other examples include the design for data mediation of [Mocan and Cimpian, 2007] and quality-of-service based discovery of [Vu et al., 2006b].

The Chapter is organised into five main sections. In Section 6.1, the structure and organization of the Semantic Web Service Challenge (SWS Challenge) is described. Sections 6.2, 6.3 and 6.4 respectively, describes how WSMX was applied to the mediation, discovery and composition problems specified by the SWS Challenge. For each problem, the results of applying the WSMX solution were evaluated against a test framework supplied by the SWS Challenge. Section 6.5 provides the details of these experimental results.

### 6.1 The Semantic Web Services Challenge

The running example, provided by the SWS Challenge [Petrie et al., 2007a,b], introduced in Section 3.1.2, defines a set of progressive increasingly complex integration problems, based on industrial specifications and requirements, for the discovery, mediation, composition and invocation of Web services. The problems are set in an overall usage scenario for B2B integration.

The central scenario used in the Challenge defines a manufacturer-buyer relationship between two fictitious trading companies called Moon (Manufacturer) and Blue (Buyer). It is illustrated in Figure 6.1. Moon maintains a set of back-end legacy systems with Web service interfaces to manage its order processing. These include a Customer Relationship Management system (CRM) and an Order Management System (OM). The scenario describes how Moon has signed agreements to exchange purchase order (PO) and purchase order confirmation (POC) messages with Blue using the RosettaNet PIP 3A4.  

---

Figure 6.1: SWS Challenge Scenario Overview (reproduced from SWS Challenge Website)
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

specification. Internally, Moon uses its own XML schema to define the messages sent to its two back end systems. A test bed, set up for the Challenge, provides the WSDL and implemented endpoints for Moon’s backend systems. All of the problems to be solved by participants are structured using the test bed. There are three categories of problems. These are discovery, mediation and composition and are described in the following subsections. The Challenge also defines constraints that need to be observed. These include:

- Time limits on response actions.
- Verification of customer details in incoming RosettaNet POs.
- Observance of the rules governing purchase orders. For example, a single RosettaNet PO may have individual line items that each have different addresses. The Moon legacy systems can not create a single order where line items have different addresses. One solution is deconstruct the RosettaNet PO into multiple POs, each with its own address, on the Blue legacy system.

<table>
<thead>
<tr>
<th>Evaluation Levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success Level 0</td>
<td>The task can be completed by the technology. Can be determined automatically by the Challenge test bed.</td>
</tr>
<tr>
<td>Success Level 1</td>
<td>Changes to the problem specification could be handled only by changing and recompiling/rebuilding the execution code.</td>
</tr>
<tr>
<td>Success Level 2</td>
<td>Changes to the problem specification could be handled by only changing the description of the models and data instances used for the test scenario.</td>
</tr>
<tr>
<td>Success Level 3</td>
<td>Changes to the scenario could be handled without any changes to either the underlying code or data models.</td>
</tr>
</tbody>
</table>

Figure 6.2: SWS Challenge Evaluation Criteria: Success Levels

Within the problem areas of mediation, discovery and composition, there are different problem levels, each of which can be evaluated to different success levels, listed in Figure 6.2. The first level is achieved if the technology can solve the basic problem outlined for the scenario problem area. The other three levels are subject to evaluation by peer review during the SWS Challenge workshops which typically take place once or twice a year. These success levels reflect how the participant technology reacts to changes introduced to the base scenario e.g. a change to the information or process models.

6.1.1 Usage of The RosettaNet B2B Standard

B2B standards in general aim at common agreement among business partners on messages and processes for inter-company integration. The B2B standard used to define the business process supported by the
Blue (Buyer) in the SWS Challenge is RosettaNet \(^{29}\). This standard defines components called *Partner Interface Processes* (PIPs), which include definitions for standard inter-company business processes, including the structure and semantics for business messages, and the secure transportation of these messages over the Internet. In particular, PIP 3A4 defines a process for the exchange of purchase order (PO) request and confirmation messages between trading partners and is used by Blue. The PIP defines four messages, PO Request, PO Confirmation and two signal messages to acknowledge the receipt of the request and confirmation respectively.

### 6.2 Mediation Scenario

#### 6.2.1 Initial Problem

![Figure 6.3: Mediation Requirements from Blue’s Perspective](image-url)

The initial problem is to enable Blue to successfully make an order with Moon where each party uses different data and process models. Blue uses RosettaNet to define the data model and PIP 3A4 to define the process. Both must be matched to Moon’s self-defined data (using XML Schema) and process model (using a UML sequence diagram). Both data and process mediation are necessary to enable the two parties to interact.

Figure 6.3 and Figure 6.4, redrawn from the Challenge Web site\(^{30}\), show the sequence diagrams for Blue’s RosettaNet process model and that defined for Moon. The RosettaNet PO messages sent from Blue need to be refactored into the various individual messages that are required by Moon. Additionally, different acknowledgment messages are expected or generated by Blue and Moon’s respective systems.

---

\(^{29}\)www.rosettanet.org, (accessed 10 Mar 2011)

Blue requires acknowledgement messages to signal that PO messages it has sent have been received and understood. These acknowledgement messages are not defined by Moon (although all required information is available during the message exchange) and so must be provided through mediation. From the perspective of Moon, each time a line item is added to an order in the OMS system, it sends an acknowledgement message. These need to be aggregated into a single RosettaNet purchase order acknowledgement.

### 6.2.2 Solution to the Initial Problem

The Challenge provides implementations and WSDL descriptions for the Web services of both Moon and Blue. The SWS-based solution to the scenario involves using the Semantic Web service technology to mediate between the data types of PIP 3A4 and the XML schema used by Moon, and ensuring that the message exchanged by each party are correctly matched and choreographed. WSMX is used to represent the mediator component shown in Figures 6.3 and 6.4. Both the data and process mediation require semantic descriptions of messages represented in WSML. Consequently, lifting and lowering of messages between the structural (XML) to the ontological (WSML) level is necessary.

There are two phases to solving the mediation problem: (1) setup and (2) runtime. During the setup phase, the required ontologies, semantic descriptions of Web services and goals, data mediation mapping
rules and lifting/lowering adapters are created. During the runtime phase the interactions via Blue and Moon are executed via WSMX. The complete solution, including all associated material, is publicly available online at the Workshops section of the SWS Challenge website\textsuperscript{31}.

\section*{Setup Phase}

The following prerequisites need to be addressed when using WSMX as a SWS solution to the mediation scenario. Note that [Kotinurmi et al., 2006] describe a similar set of pre-requisites for an abstract example applying WSMX to a B2B integration situation.

- \underline{Message Ontologies.}
  - A WSMO ontology to represent the fragment of the RosettaNet information model for PIP 3A4 used by the challenge.
  - A WSMO ontology to represent the information model corresponding to the combined XML schema for Moon's legacy systems.

- \underline{Data mediation rules between the message ontologies.}

- \underline{A WSMO goal to represent what Blue, as a service requester, wants to achieve and the message exchange it supports.} The choreography interface description for the goal corresponds to the RosettaNet PIP 3A4 process description. The capability contains sufficient information to match the goal to Moon’s systems.

- \underline{A WSMO Web service description representing a combination of both the CRM and OMS services.} The choreography interfaces defined for the combined service grounds messages to operations defined in the individual WSDL descriptions of the CRM and OMS services. This is a simplification in the model to achieve a simple form of service composition. It is appropriate for this scenario as the required services are known up-front and the focus is on resolving data and process heterogeneity between Blue and Moon (rather than on discovery and composition of unknown Web services).

- \underline{A RosettaNet-to-WSMX adapter for the Blue system.} The adapter is responsible for:
  - Creating and dispatching WSMO goals that correspond to individual RosettaNet purchase order requests.
  - Lifting and lowering the RosettaNet XML messages to corresponding WSML messages

\textsuperscript{31}http://sws-challenge.org/wiki/index.php/Workshops (accessed 10 Mar 2011)
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

- Implementing an interface with the same RosettaNet PIP 3A4 protocol as the Blue system – allowing the messages between Blue to Moon and pass through after they have been lifted or lowered.

- Provide the signal messages that the RosettaNet protocol requires. This is an example of a type of process mediation – missing messages (cf. Section 5.3) – that is not currently resolved by the WSMX prototype.

- Lifting and lowering adapters to carry out the syntactic transformation between XML and WSML for messages sent between WSMX and the various Web services representing Moon backend systems.

6.2.3 Runtime Phase

```xml
<!-- Lifting rules from XML message to WSML */
...
instance PurchaseOrderUID memberOf por#purchaseOrder
por#globalPurchaseOrderTypeCode hasValue "<xsl:value-of select="dict:GlobalPurchaseOrderTypeCode"/>
por#isDropShip hasValue
  IsDropShipPo<xsl:for-each select="po:ProductLineItem">
    por#productLineItem hasValue ProductLineItem<xsl:value-of select="position()"/>
  </xsl:for-each>
<xsl:for-each select="core:requestEvent">
  por#requestedEvent hasValue RequestedEventPo
</xsl:for-each>
<xsl:for-each select="core:shipTo">
  por#shipTo hasValue ShipToPo
</xsl:for-each>
<xsl:for-each select="core:totalAmount">
  por#totalAmount hasValue TotalAmountPo
</xsl:for-each>
...

<!-- message in WSML after transformation */
...
instance PurchaseOrderUID memberOf por#purchaseOrder
por#globalPurchaseOrderTypeCode hasValue "Packaged product"
por#isDropShip hasValue IsDropShipPo
por#productLineItem hasValue ProductLineItem1
por#productLineItem hasValue ProductLineItem2
por#requestedEvent hasValue RequestedEventPo
por#shipTo hasValue ShipToPo
por#totalAmount hasValue TotalAmountPo
...
```

Listing 6.1: Lifting in XSLT and resulting WSML message
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

Figure 6.5: Runtime Activity for the Mediation Problem
In this section, all interactions between the Blue and Moon systems via WSMX are described using the UML sequence diagram of figure 6.5. The diagram is broken up into five parts, each of which is described in the following sections.

1 – Sending Request.

The process starts by providing the endpoint of a RosettaNet-WSMX adapter to the Challenge testbed. The responsibility of the adapter is to create a WSML goal including (1) a capability description to purchase the items included in the PO request, (2) the choreography interface supported by the goal owner – corresponds to the RosettaNet PIP 3A4, and (3) to lift and lower RosettaNet messages between XML and WSML. Listing 6.1 shows a fragment of the lifting-rules in XSLT and the resulting WSML message. A RosettaNet PO request message is generated from the testbed and sent to the adapter endpoint. On receipt of the PO request, an acknowledgement signal is sent from the adapter back to the Blue Web service.

The WSML description of the goal is sent to WSMX through the AchieveGoal interface operation exposed by the Communication Manager interface described in Section 4.4.3. In return, a context identifier is received that is used to identify the messages exchanged (conversation) for that specific goal. The context is important as multiple independent conversations can be running inside WSMX concurrently. Individual messages received from either a goal owner or a service provider are associated to specific conversations using the context value.

2 – Discovery and Conversation Setup.

The AchieveGoal entrypoint is an SSOA interface operation implemented and made public by the Communication Manager component. On receipt of the goal, this component initiates an instance of the goal-driven service discovery and invocation behaviour on the service bus with the goal, and a WSML ontology fragment representing data associated with that goal as input parameters (a description of this behaviour can be found in Section 4.5.3).

The WSML descriptions are parsed into Java objects using the opensource WSMO4j API. This parsing step is required each time WSMX receives WSML messages. After parsing, service discovery is carried out to identify services matching the requested capability. Discovery is trivial for this use case since only one service in the repository (the combined CRM/OMS service description) can be matched with the goal, thus it is not necessary to deal here with selection of services based on other attributes e.g. quality of service.

Once a service is discovered, both the requester and provider choreography interfaces are registered with the choreography engine. Both choreographies are set to an initial state waiting for incoming
3 – Conversation with Requester.

In this example the instance data for the goal is sent to WSMX using the achieveGoal entry point. The WSML data is parsed with the goal and held in cached storage by the service bus until both choreographies have been registered.

Once these registrations are complete, the goal input data is provided to the process mediator. The process mediator is responsible for analysing the ontological structure of the messages to determine (1) if data mediation is required and (2) what information should be added to the choreography engine for the instances of goal and Web service respectively (cf. Section 5.3).

In the Challenge scenario, data mediation must be performed on the goal data to create equivalent data instance in terms of the ontology used by the service provider. For this, the goal data instances (which include a reference to the goal provider’s ontology) and the target ontology for the service provider are passed to the data mediator. Data mediation is performed by executing the mapping rules between both ontologies (these mapping rules were created and registered during the setup phase). The process mediator then updates the memory space of the goal owner’s choreography (the buyer – Blue) with the information for the purchase order request. It then evaluates how the data resulting from the data mediation should be added to the memory space for the service provider’s choreography (the manufacturer Moon). The design and implementation of the data mediation component used by WSMX is the work of Adrian Mocan and Emilia Cimpian. Details are not included here but are available in [Mocan and Cimpian, 2005] and [Mocan and Cimpian, 2007].

4 – Conversation with Provider (opening order, add line items, closing order).

After the goal and service choreographies have been updated, the choreography engine processes each choreography to evaluate if any transition rules have been fired. In the scenario, the goal’s choreography remains in a waiting state as no rule can be evaluated at this stage. For the service’s choreography, the choreography engine finds the rule shown in the listing 6.2 (lines 14-21). Here, the choreography engine matches the data in memory with the the antecedent of the rule and performs the action of the rule’s consequent (i.e. update/delete of the memory). The rule says that the message SearchCustomerRequest with data, searchString, should be sent to the service owner (this data had been previously added to the choreography memory after data mediation). The data, searchString, corresponds to the customerId from the goal’s ontology). The decision to send out the message is determined by the choreography engine interpreting the grounding for the message, specified in the WSMO service choreography state signature, and then passing the message to the communication manager via the service bus.
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

The communication manager uses the lifting/lowering adapter for the CRM/OMS service to lower the outgoing message from WSML to XML before sending the message to the searchCustomer entrypoint of the CRM/OMS Web service. Some messages to and from the service bus are omitted for clarity. In the meantime the choreography for the service is put in a waiting state until the arrival of the SearchCustomerResponse message from the service provider.

Listing 6.2: Requester’s Service Choreography

Listing 6.2 shows the fragment of the provider’s choreography and selected rule described above. The choreography is described from the service’s point of view. The rule says that the service expects to receive the SearchCustomerRequest message and send the reply SearchCustomerResponse message. In the StateSignature section (lines 3-11), concepts for the input, output and controlled vocabulary are defined. Input concepts correspond to messages sent to the service, output concepts correspond to messages sent out of the service, and controlled concepts are used for controlling the states, and transition between states during processing of the choreography. Each concept used is prefixed with a namespace definition (e.g. moon, oasm) corresponding to the imported ontologies (lines 4, 5). The choreography is part of the service definition which in addition also contains definition of non-functional properties and capability. For brevity, these elements are not included in the listing.

In the adapter, lowering the WSML message to XML is performed using the lowering rules for the
CRM/OMS ontology and the CRM XML schema. This XML (searchString) is sent to the Web service representing the CRM system. It searches for the customer and returns an XML customerObject message to the communication manager which uses the CRM/OMS adapter to lift the message from XML to WSML (to avoid cluttering the diagram, Figure 6.5 is simplified to show the flow of messages is first to the adapter, then the communication manager). Once lifted to WSML, the message is sent on to WSMX, where it is parsed, evaluated by the process mediator (which coordinates any request for data mediation), before the data is added to the provider’s choreography memory.

Once the ontology of the provider’s choreography is updated, its set of transition rules is evaluated resulting in a createNewOrder message being sent to the Moon OMS Web service (same process as when sending the searchCustomerRequest described above). The execution of the Web service results in an orderID message from the OMS system being received by WSMX and added to the memory of the Web service provider’s choreography.

After the order is created (opened) in the OMS system, the individual items to be ordered need to be added. These items were previously sent from Blue in one message as part of the RosettaNet request (i.e. a collection of ProductLineItems) which must be now sent to the OMS system individually. As part of the data mediation in step 3, the collection of items from the RosettaNet order request were split into individual items whose format is described in Moon’s ontology. The process mediator also added these items into the Moon’s choreography for the CRM/OMS service at that point. After the orderID has been added, the rules for Moon’s service choreography are evaluated again resulting in the firing of the rule for sending addLineItem message with data of one lineItem from the choreography memory. Since there is more then one line item in the memory, this rule will be evaluated several times until all line items from the ontology have been sent to the OMS Web service. When all line items have been sent, the next rule to be evaluated in the provider’s choreography is to close the order in the OMS system. The closeOrder message is sent out from WSMX to the OMS service. The OMS service acknowledges this with a closeOrderAck message which is accepted by the WSMX communication manager but not required by the Buyer goal choreography.

5 – Conversation with Requester (order confirmation, end of conversation).

To confirm that the order is closed, the OMS service sends an orderConfirmation message. After lifting and parsing of the message, the process mediator drives data mediation of the confirmation message to the requester’s ontology and then evaluates that it needs to be added to the memory of the goal owner’s (Blue) choreography. At this stage, the next rule of the goal choreography can be evaluated which indicates that a purchaseOrderConfirmation message needs to be sent to the Blue Web service endpoint.
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

After the message is sent, no additional rules can be evaluated in Blue's choreography, thus the choreography gets to the end-of-conversation state. Since both the goal and Web service choreographies are both in the state of end-of-conversation, the choreography engine notifies the execution semantics and the conversation is closed.

6.2.4 Modified Problem

The Challenge scenario was modified to introduce changes leading to implications on the requirements for both data and process mediation. The following changes were made to Moon's legacy systems

- Blue uses a different endpoint for receipt of incoming messages.
- Addition of an additional production management (PM) system.
- The order management system is renamed to the stock management (SM) system.
- The redefinition of the order management process.

The new order management process has additional steps. When Moon receives a purchase order request, it can check for the requested stock using the stock management system. If stock is unavailable, the PM system should be queried to check if the required products can be manufactured in time and within the given price constraints. If this is the case, this production should then be scheduled.

An additional change was the introduction of individual addresses for line items within a single RosettaNet PO message. This was possible under the terms of the initial problem as proposed by the Challenge but not tested as part of the initial evaluation. The Moon system is expected to react by creating individual purchase orders for each set of line items belonging to a particular address. The net expected result is that all line items are handled correctly and the results are aggregated into a single RosettaNet purchase order confirmation message returned to Blue.

Solution for the Modified Scenario

Some modifications were necessary to the initial solution. These are categorised as declarative (D), where changes were made to descriptions only (services, goal, ontologies or mediators), and code (C), where changes to the execution code were required. The ideal is that only declarative changes are necessary. All but one of the changes below are in this category.

- (D) The Blue endpoint for sending and receiving RosettaNet messages in the test bed changed requiring the goal description to be updated so that the POC message was grounded to the modified WSDL endpoint.
• (D) Extended the WSMO ontology for Moon’s legacy systems to include the concepts used by the production management system.

• (D) Extended the mediation mappings to include new mappings for the production management system.

• (D) Modified the choreography description of the Web service representing Moon’s legacy systems to reflect the business logic of the production management system introduced to the process.

• (D) The initial solution did not handle cases where individual line items could have distinct addresses. Handling this aspect was achieved through modifying the choreography of the Moon Web service.

• (C) A lifting/lowering adapter between WSML and XML for the production management system had to be created. The lowering adapter was coded as an additional Java class while the lifting required the creation of an XSLT transformation.

### 6.3 Discovery Scenario

#### 6.3.1 Initial Problem

The discovery scenario is orthogonal to that for mediation. Moon acts as an e-market place for the purchase and shipment of products. Five different service providers, each offering various purchasing and shipment options, are defined. Each provide different availability and pricing for their respective services with constraints on package destination, weight, dimension and shipment date. Service descriptions are provided which are a mixture of natural language text and WSDL service descriptions.

```plaintext
1 Rates on Request (cf. invokePrice operation within the WSDL)
2 Only packages weighing 50 lbs or less are shipped
3 Ships to Africa, North America, Europe, Asia (all countries)
4 Constraints on Collection:
5    − There must be at least an interval of 90 minutes for collection.
6    − Collection is possible between 7am and 8pm.
7    − Collection can be ordered max 2 working days in advance.
8 Delivery Time:
9    − Ships in 2/3 (domestic/international) business days if collected by 5pm.
```

Listing 6.3: Muller Shipper Service Description

Each incoming order request to Moon involves packages with different properties to be shipped to various destinations. Modelling the WSMO descriptions of the shipment services to include a *data-fetch* interface fits this scenario as it allows dynamic information (e.g. varying price based on weight) to be
explicitly modelled as part of the service choreography interface. This information is then used during
WSMX discovery. For example, the Mueller service has price information that can only be fetched via a
Web service operation. This operation is modelled as part of that service’s data-fetch interface. Listing
6.3 provides a textual description that defines conditions on the usage of the service.

The WSDL defining the functional interface can be inspected at the Challenge website\(^{32}\) and defines two
operations:

- **ShipmentOrder.** The input message is a ShipmentOrderRequest which contains fields for `fromAddress`, `shippingDate`, `packageInformation` and `addressTo`. The package information consists of integer values for the `quantity`, `weight`, `length`, `height` and `width`. The output is a ShipmentOrderResponse which provides fields for `pickupDate` and `price`.

- **InvokePrice.** The input is the `country` to ship to and `packageInformation` defining the details of the package(s) to be shipped. The output is a `price`.

The WSDL description alone is insufficient to determine whether or not the Mueller service meets the
requirements of a requester. The additional textual part of the service description defines the necessary
conditions that must hold for the service to be provided e.g. only packages weighing 50lbs or less are
shipped; pricing is determined on the basis of individual orders. Only the fact that there is an additional
operation for the retrieval of pricing information could be included in the WSDL but without any context
as to how it could be automatically used during service matching.

### 6.3.2 Solution to Discovery using WSMX

The components of the solution are illustrated in Figure 6.6. The actual message exchange required

for the discovery behaviour corresponds to that shown in the sequence diagram of Figure 4.19. Moon is modelled as an e-market place with two parts. The first is an adapter with a WSDL-described service interface that accepts discovery requests corresponding to those specified in the Challenge scenario. The second part is an SSOA framework (WSMX) that can carry out the discovery process and make invocations on any matching provider services.

The adapter is responsible for transforming service requester information to and from WSML. Incoming requests to the adapter are transformed into goals and sent to WSMX where matching services are sought and invoked. Returned data is passed to the adapter and transformed into the format required by the service requester.

The service providers focus on providing rich descriptions of what they offer. Semantic Web descriptions for the five services defined by the Challenge are made available to WSMX (named Mueller, Runner, Walker, Racer and Weasel).

**Artifacts in the Solution**

The solution is designed using the following artifacts:

- Two common shipping ontologies are used by all Web service descriptions associated with the Moon Computer Manufacturer. The *ShipmentOntology* defines concepts related to shipping while the *ShipmentProcessOntology* defines messages used by the individual services.

- WSMO goal descriptions are created for each of the various shipment requests defined by the Challenge.

- WSMO semantic Web service descriptions are defined for each of the shipping services provided by the Challenge. Each WSMO service description grounds to a corresponding WSDL description for a shipping service.

- Lifting and lowering adapters are required to translate between WSML and XML instances.

```latex
1 /* ShipmentOrderRequest definition */
2 concept ShipmentOrderRequest
3 from ofType so#ContactInfo
4 shipmentDate ofType so#ShipmentDate
5 package ofType so#Package
6 to ofType so#ContactInfo
```

**Listing 6.4:** *ShipmentOrderRequest* relation

The ShipmentOntology (prefix: *so*) defines concepts for the shipping domain such as *package, dimensionalWeight* etc. used by the messages exchanged by the shipping services. The ShipmentProcessOntology
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

defines messages used in processes associated with shipping such as ShipmentOrderRequest and ShipmentOrderResponse. For example, listing 6.4 shows the definition for a ShipmentOrderRequest.

In addition, the ShipmentProcessOntology is used to declare named common relations for services and goals. Each Web service description using a relation provides an axiom to formally define that relation. This allows reuse of the relation while allowing each Web service to provide its own distinct conditions in the definition. Listing 6.5 shows the definition for the isShipped relation from the ShipmentOntology and its implementation defined for the Mueller shipping service. The relation isShipped is true for Mueller if the destination city for the shipment is in Europe, Asia or Africa, and the dimensional weight of the package is less than 50kg.

```plaintext
1 /* isShipped relation in the common ontology */
2 relation isShipped(ofType mo#ShipmentOrderReq)
3
4 /* implementation of the isShipped relation in Muellers ontology */
5 axiom isShippedDef
6  definedBy
7  ?shipmentOrderReq[sop#to hasValue ?destContact, sop#package hasValue ?package] memberOf sop#ShipmentOrderReq and
8  ?destContact[so#address hasValue ?destAddress] and
9  ?destAddress[so#city hasValue ?city] and
10  isShippedContinents(?city, so#Europe, so#Asia, so#NorthAmerica, so#Africa) and
11  ( (?package [so#weight hasValue ?weight] memberOf so#Package) and (?weight =< 50) ) and
12  ( (?dimensionalWeight[so#weight hasValue ?dimweight] memberOf so#DimensionalWeight) and (?dimweight =< 50) )
13  implies
14  sop#isShipped(?shipmentOrderReq).
```

Listing 6.5: isShipped relation

Applying the Discovery Phases

Section 5.1.1 listed four stages of discovery for WSMO defined in [Keller et al., 2005]: (1) goal creation, (2) goal refinement, (3) abstract service discovery and (4) service contracting. These are applied to the Challenge scenario. A shipment request with conditions is defined which is formalized into a WSMO goal description (phases 1 and 2). Web service descriptions that match the abstract goal are discovered based on a comparing the semantic description of Web service non-functional properties, and the logical conditions defined in the goal and service capability descriptions (stage 3). Finally, the Web services are evaluated against ranking criteria semantically described in the goal description, and ranked in order, where the highest ranked service is placed first in the list (stage 4). Note, that according to the the SSOA model presented in Chapter 4, phase 4 is the responsibility of the Selection component.
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

Run-time Execution

The interface description of each candidate service is examined to see if (a) there is sufficient data available from the requester to invoke the functionality and (b) if the constraints on the output will satisfy the specific constraints specified in the data that accompanied the requester’s goal. The resolution of (a) depends on examining the choreography descriptions of the requester goal and the provider service. Section 5.1 described how this can be achieved using an ontological knowledge base to store the available information and then querying it to determine if a match can be made. For (b), it may be necessary to request additional information from the service provider at runtime. For example, it may be necessary to get an exact price for shipment on a particular day to a particular address. If the service provides a data-fetch operation on its interface to provide this information, the discovery component can invoke that operation and add the data that it gets in return to the knowledge base. Continuing with the Mueller Web service as the running example (as used earlier in Listing 6.3), goal B1, defined by the Challenge,

<table>
<thead>
<tr>
<th>Discovery Based on Destination and Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>To: Szyslak, Tunisia</td>
</tr>
<tr>
<td>Package dimensions: (l/w/h) 40/10/10</td>
</tr>
<tr>
<td>Package weight: 30 lbs</td>
</tr>
<tr>
<td>SWS Challenge Services that match:</td>
</tr>
<tr>
<td>Mueller, Runner and Walker</td>
</tr>
</tbody>
</table>

Listing 6.6: Goal B1: Discovery Based on Destination and Weight

Listing 6.7 lists the capability element of the WSML description for the goal described above. The capability contains only a post-condition which means that Web services will match the goal if the stated post-condition holds after the Web service has executed. The capability in Listing 6.7 is interpreted as: the requirement for a Web service where (1) its invocation results in an instance of the concept ShipmentOrderResponse (defined in the ontology identified by the namespace sop), and (2) that the relation isShipped must be true (this relation is defined by an an ontology used in the service description).

<table>
<thead>
<tr>
<th>capability GoalB1Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>postcondition</td>
</tr>
<tr>
<td>definedBy</td>
</tr>
<tr>
<td>( ?x[sop#pickupDate hasValue pickupDate, sop#price hasValue ?price] memberOf sop#ShipmentOrderResponse and</td>
</tr>
<tr>
<td>sop#isShipped(shipmentOrderReq) ).</td>
</tr>
</tbody>
</table>

Listing 6.7: Goal B1 Capability

---

CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

At run-time both the Goal and an instance of a ShipmentOrderRequest are provided to the WSMX discoverWebService interface implemented by the Communication Manager (see Section 4.4.3). This results in the creation of an instance of goal-driven service discovery behaviour on the WSMX Service Bus. In the running example, the WSML Goal and instance data for the ShipmentOrderRequest are parsed into Java objects and loaded into memory by WSMX. During the discovery phase Web services known to the WSMX service registry are considered for matching. For example, the Mueller Web service matches the capability including the evaluation of the isShipped relation through the isShippedDef axiom of listing 6.5.

The initial goal could be matched to a service based on the provided capabilities, instance data and the definition for evaluating the isShipped relation. Goal C1 defined by the Challenge introduces a constraint on the price of the service (price < Euro 20). However, information on the pricing for each service is dependent on details of the shipment request being made. In other words, an operation on the service needs to be invoked so that this price information can be retrieved. A snippet from Goal C1 showing the revised Goal capability definition is provided in listing 6.8.

```java
1 capability GoalC1Capability
2  postcondition
3  definedBy
4   (?x[sop#pickupDate hasValue pickupDate, sop#price hasValue ?price] memberOf sop#ShipmentOrderResponse and
5   ?y[sop#price hasValue ?price] memberOf sop#PriceQuoteResp and
6   sop#isShipped(shipmentOrderReq) and
7   ?price < 20).
```

Listing 6.8: Goal C1 Capability

The Mueller Web service provides an operation to get a price quotation based on a submitted shipping request. This operation is included as part of the data-fetch interface in the Mueller WSMO service description, shown in listing 6.9 (note: this was also used for the example described in Section 5.1.3). At run-time, the discovery component of WSMX checks if the Mueller service description has a data-fetch interface. As this interface exists and the required input data for the interface is available, WSMX invokes the operations on the interface and uses the returned information to further the Web service discovery process. Listing 6.9 includes two rules. The first line specifies that on receipt of a purchase quote request (PurchaseQuoteReq), the Web service will provide a purchase quote response (PurchaseQuoteResp). The second rule uses the purchase quote response and is therefore dependent on the first rule. Rule 2 states that (1) if a purchase quote response and a shipment quote request are available to WSMX, and (2) if both the isAvailable and isShipped relations evaluate to true, then a shipment quote response will be provided by the data fetch interface of the Web service.
The **grounding**, which links the WSML description to the WSDL description for the Mueller Web service, is included in Listing 6.9.

```xml
wsmlVariant "http://www.wsmo.org/wsml/wsml−syntax/wsml−rule"

webService muellerShippingService
importsOntology
capability muellerCapability
... interface muellerInterface ... 

choreography WSMuellerDataFetchChoreography
stateSignature 
in sop#PurchaseQuoteReq withGrounding
"http://sws−challenge.org/shipper/v2/mueller.wsdl#wsdl.interfaceMessageReference(mueller/PurchaseQuote/In)"
out sop#PurchaseQuoteResp withGrounding
"http://sws−challenge.org/shipper/v2/mueller.wsdl#wsdl.interfaceMessageReference(mueller/PurchaseQuote/Out)"
in sop#ShipmentQuoteReq withGrounding
"http://sws−challenge.org/shipper/v2/mueller.wsdl#wsdl.interfaceMessageReference(mueller/ShipmentQuote/In)"
out sop#ShipmentQuoteResp withGrounding
"http://sws−challenge.org/shipper/v2/mueller.wsdl#wsdl.interfaceMessageReference(mueller/ShipmentQuote/Out)"

transitionRules WSMuellerDataFetchTransitionRules
/∗ Rule 1: Request for product quote ∗/
forall {?purchaseQuoteReq} with (?purchaseQuoteReq memberOf PurchaseQuoteReq)
) do
  add(# memberOf PurchaseQuoteResp)
endForall

/∗ Rule 2: Request for shipment quote ∗/
forall {?shipmentQuoteReq} with (?shipmentQuoteReq?package hasValue ?package memberOf sop#PurchaseQuoteResp
and
?shipmentQuoteReq?to hasValue ?to memberOf sop#ShipmentQuoteReq
and
sop#isAvailable(?purchaseQuoteResp) and sop#isShipped(?to, ?package)
) do
  add(# memberOf mo#ShipmentQuoteResp)
endForall
```

Listing 6.9: Mueller Data Fetching Interface

As described in Section 5.3.1, a description of WSMO Grounding can be found at [Kopecký et al., 2006]. In this case, there are four elements in the WSML state signature for the data-fetch choreography which ground to four distinct messages in the corresponding WSDL Web service description. For example,
the PurchaseOrderReq concept, defined in the ontology ShipmentProcessOntology with namespace sop, is grounded to the in message of the PurchaseQuote operation of the mueller interface of the WSDL description. The designation in and out indicate that the messages are incoming or outgoing with respect to the Web service.

6.4 Composition Scenario

The third part of the SWS Challenge deals with service composition problems. The scenario involves requests to purchase multiple elements of computer hardware. These requests lead to the following three types of service composition (reproduced from the Challenge Wiki\[^34\]):

- **Uncorrelated composition**: The requests contains several products that may or may not need to be purchased from different providers

- **Correlated composition**: The requests contains several products that may or may not need to be purchased from different providers plus not all pairs of products are suitable and compatible to each others

- **Composition with global optimization**: The requests contains several products that may or may not need to be purchased from different providers. There are also global optimization and constraints e.g. a maximum total cost for all items.

<table>
<thead>
<tr>
<th>Hardware Category</th>
<th>Mac Book 13&quot;</th>
<th>HP NX9000</th>
<th>HP Docking Station</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category</strong></td>
<td>Notepad</td>
<td>Notepad</td>
<td>Docking station</td>
</tr>
<tr>
<td><strong>Specification</strong></td>
<td>13&quot; flat screen; 1.83GHz Intel Core Duo; Memory:512MB; DDR2SDRAM; HDD:60 GB; color: white</td>
<td>15.0&quot;/XGA (1024 x 768) Pixel; AMD Turion 64 X2; 1600 MHz; 512MB DDR2-RAM; 80GB HDD; Double Layer DVD+/-; RW/DVD-ROM; ATI Radeon Xpress 1150; Ethernet 10/100/1000BTX 56K Modem; WLAN 802.11a/b/g; Bluetooth; Windows XP Professional</td>
<td>For NX9XXXX series: (00000009, 00000007)</td>
</tr>
<tr>
<td><strong>GTIN</strong></td>
<td>00000001</td>
<td>00000005</td>
<td>00000017</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>$1099.00</td>
<td>$1057.00</td>
<td>$239.00</td>
</tr>
</tbody>
</table>

Figure 6.7: Rummage Supplier Products

One task is chosen for this section as a representative example requiring correlated composition. Figure 6.7 describes three of the products made available by a supplier called Rummage. Figure 6.8

Chapter 6. SSOA Implementation & Evaluation Against Independently-Set Integration Scenarios

<table>
<thead>
<tr>
<th>Hardware</th>
<th>HP NX9420</th>
<th>Mac Book 13&quot;</th>
<th>HP Docking Station</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Notepad</td>
<td>Notepad</td>
<td>Docking station</td>
</tr>
<tr>
<td>Specification</td>
<td>17&quot; WXGA+ TFT (1440x900); Centrino Duo 1.83GHz; 1GB DDR2-RAM; 100GB HDD; DVD+/-RW; Windows XP Professional; WLAN; BlueTooth</td>
<td>13&quot; flat screen; 2.0 GHz Intel Core Duo; Memory: 2GB; DDR2SO-DIMM; HDD: 120 GB; color: black</td>
<td>Docking station for X41 (00000008)</td>
</tr>
<tr>
<td>GTIN</td>
<td>0000 0007</td>
<td>0000 0010</td>
<td>0000 0016</td>
</tr>
<tr>
<td>Price</td>
<td>$1249.00</td>
<td>$2049.00</td>
<td>$269.95</td>
</tr>
</tbody>
</table>

Figure 6.8: Bargainer Supplier Products

describes those for a second such supplier called Bargainer. A number of client goals are specified in natural language, each of which can only be satisfied by some type of service composition. Listing 6.10 reproduces one of these goals (goal C2) that requires correlated composition to order both a laptop and docking station with the constraint that each is compatible with the other. A preference is specified that where multiple matching pairs are discovered, the cheapest should be selected.

Listing 6.10: Goal requiring Correlated Composition

1. Purchase any Laptop with attributes of:
2. at least 512 MB RAM
3. at least 40 GB HDD
4. and
5. a compatible docking station
6. if multiple combinations are available, prefer cheapest option.

6.4.1 Solution to Composition

The approach to service discovery, including runtime data fetching, is extended to tackle the problems of simple composition presented by the Challenge. The composition is described as simple because it assumes that the order in which the services are invoked is not important. The criteria for the Challenge evaluation are that the constraints on individual services and those defined in the tasks (represented as goals) are met and that the constraints between particular services are also satisfied. The first step of composition, for this example from the WSMX point of view, provides a set of laptop-docking_station pairs. This is used to construct an orchestration description which can be executed by the WSMX middleware. The full details of the approach are published at [Moran et al., 2007].

A common ontology is defined to define shared concepts used in the descriptions of goals and services, such as Location, Notebook, DockingStation, etc. In addition, this ontology is used to specify named
relations. Specific ontologies for goals and services declare axioms that define the relations to represent their conditions. As before for discovery, a set of relations is defined in the common ontology which represent the axioms that a service may need to define. Axioms provide the definitions for these relations with specific conditions. For example, listing 6.11 shows the simple declaration for the \textit{isCompatible} relation in the common ontology and how it can be defined in the service ontology.

\begin{verbatim}
1 /* isCompatible relation in the domain ontology */
2
3 relation do#isCompatible (ofType do#Notebook, ofType do#DockingStation)
4
5 /* implementation of the isCompatible relation in the service ontology */
6
7 axiom isCompatibleDef definedBy
8 ?notebook[do@GTIN hasValue ?gtinX]
9 memberOf do#Notebook and
10 ?dockingstation[do@supportsGTIN hasValue ?gtinY]
11 memberOf do#DockingStation
12 and ?gtinX = ?gtinY) implies
13 do#isCompatible(?notebook, ?dockingstation).
\end{verbatim}

Listing 6.11: \textit{isCompatible} relation

The relation \textit{isCompatible} is true if the notebook sold by the service provider can be used with one of the available (\textit{DockingStation}). This axiom can be used in the goal query to check compatibility of the two components.

Semantic descriptions are created, using WSMO, for the services provided by the Challenge (described using free text and WSDL). Service descriptions include data-fetch interfaces for the dynamic properties services may have that are only available on request at run-time. The capability of service descriptions may use their own logical definition of the \textit{isCompatible} relation to specify constraints of the service (e.g. IBM laptops only fit IBM docking stations etc.).

For each of the tasks specified by this part of Challenge, a WSMO goal is specified. An example for the correlated composition goal (listing 6.10) is shown below in listing 6.12. This specifies that we are looking for a correlated notebook and docking station with specified constraints on memory and hard-drive capacity and that we have a preference on the combined price (lower is better).

Independent discovery is carried out to find matching services for the notebook and the docking station. The information obtained for all candidate services is merged into a single knowledge base. The postcondition of the goal forms a query that is applied to this knowledge base using the reasoning engine. A set of zero or more pairs of matching correlated service pairs are returned in order of the specified preference (price). The cheapest option is selected and an orchestration of the two services created to
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

make the order of the notebook and the docking station. As mentioned earlier, the orchestration is simplified on the assumption that there is no invocational dependency between the two services and the order in which they are invoked is unimportant.

```
1 Goal GoalPurchaseHardware
2   nfp
3   _"preference" hasValue "?price"
4   ...
5 endnfp
6 ...
7 capability GoalPurchaseHardwareCapability
8 postcondition
9   definedBy
10     (?x[po#price hasValue ?priceX, po#GTIN hasValue ?gtinX, po#hddGB hasValue ?hddGBX, po#memoryMB hasValue ?memMBX] memberOf po#Notebook
11       and ?memMBX >= 512
12       and ?hddGBX >= 40 and
13       ?y[po#price hasValue ?priceY, po#supportsGTIN hasValue ?gtinY] memberOf po#DockingStation
14       and ?gtinX = ?gtinY
15       and ?price = (?priceX + ?priceY)
16   ).
17 ...
```

Listing 6.12: User Goal in WSMO

For the purpose of the Challenge, a solution to the composition scenario can be provided based on extending the data-fetch discovery algorithm under the assumption that invocation order was unimportant. For many processes, this is not the case and there is a wealth of research in AI planning, workflow and business process management examining rich composition techniques. These include the work of McIlraith et al. [McIlraith and Son, 2002] modelling requests and services using first-order situation calculus, McDermott [McDermott, 2000] in planning using PDDL and DAML-S, Van der Aalst using workflow patterns [Aalst et al., 2003], and Osman et al. [Osman et al., 2005] on bridging workflow with Semantic Web service based composition.

From a functional perspective, the solution provided was able to successfully complete all composition tasks specified by the Challenge. The focus was on being able to handle both constraints on the services themselves and constraints on the goals that represented client requests. It was also possible to handle correlation constraints specified between service definitions.
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

6.5 Experimental Results of the WSMX SSOA in Integration Scenarios

This section describes the results of applying the WSMX SSOA reference implementation against the problems defined by the SWS Challenge. The Challenge organizers provide the description of the goals to be solved and the available web services using a mixture of WSDL and free text. Solving the problems using the WSMX SSOA reference implementation requires modelling these goals, web services, and the ontologies that they require, using WSMO. The problems are grouped into the following five categories:

- Mediation
- Discovery with simple constraints (single or multiple)
- Discovery with constraints requiring arithmetic computation
- Discovery requiring service invocation
- Discovery including simple composition

Each scenario is briefly described in the following sections. The details of the individual problems and their solution in the context of WSMX is described tabularly. Where appropriate WSML snippets are included to illustrate specific points. The overall mediation, discovery and simple composition scenarios and a detailed description of how WSMX was applied to them are described in Sections 6.2.1, 6.3 and 6.4 respectively.

6.5.1 Mediation

The mediation scenario involves two parties (a buyer, Blue, and a manufacturer, Moon) whose systems need to interact but which use different data and behaviour models at their web service interfaces. A description of the changes required by the WSMX SSOA approach to cater for changes made to the initial scenario are described in Section 6.2.4. A summary of the evaluation of WSMX against the scenario is described in Figure 6.9.
## 6.5.2 Discovery Problems

<table>
<thead>
<tr>
<th>Prob ID</th>
<th>Result</th>
<th>Brief Description</th>
<th>Artifacts Created or Modified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2a: Goal A1</td>
<td>Level 2</td>
<td>Deliver a package with specific dimensions to an address in: Goal A1: Tunisia. Goal A2: Luxembourg.</td>
<td>Shipment ontology – shipping concepts. Shipments process ontology – common messages related to shipping processes. Geographic location ontology. Constant ontology instances e.g. the addresses of companies etc. WSMO goal description including a relation isShipped that must be true for matches. WSMO service descriptions for each of the available services including grounding definitions to relevant WSDLs. An axiom to define the isShipped relation is defined for each service to represent specific constraints on service provision. Additional axioms to define the countries to which the services will ship. Lifting/Lowering adapters.</td>
<td>All ontological information was loaded to a WSMX knowledge base at runtime. The capability of the goal was used as a query against this knowledge base yielding the correct result. Service constraints were defined using axioms represented in WSML.</td>
</tr>
<tr>
<td>1a</td>
<td>Level 1 (code changes required)</td>
<td>Changes to data model of initial scenario. Introduction of a Product Management System and associated data model. RosettaNet data model changed for purchase order requests and responses. Each Moon order line item may contain a unique address.</td>
<td>Goal description modified to ground interface to new endpoints. Moon's ontology extended for production management system. Mediation mappings extended. New code for a lifting/lowering adapter for the production management system.</td>
<td>Only the coding for the lifting/lowering adapters had to be changed. All other changes were handled by changes to the declarative descriptions. With further refinement, new or modified adapters could be dropped into the WSMX SSOA system at runtime.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changes to process model used by Moon's back-end systems: Order Management System renamed to Stock Management; New Production Management (PM) system added. Each order line item may contain unique address.</td>
<td>Choreography description for Moon's web service modified to include process steps for production management system. A further modification to the mediation mappings and Moon service choreography description allowed the handling of splitting a single incoming RosettaNet request message, where individual line items had unique addresses, into a set of Moon order request messages.</td>
<td>Changing the description of Moon's web service choreography was sufficient to enable WSMX handle the additional logic specified for the new PM system and the individual line item addresses.</td>
</tr>
</tbody>
</table>

Figure 6.10: Evaluation of Discovery with Simple Constraints

A set of goals is defined requiring service discovery including defined constraints. Some of these can be satisfied from the public service description. Others require interaction with the service during discovery. Services descriptions in WSDL and free text are provided representing five different shipping companies.
Discovery with Simple Constraints

Two problem goals are defined requiring a package with a trivially small weight to be shipped. The first delivery destination is to Tunisia, Africa and the second is to Luxembourg, Europe. The results of the shown in Figure 6.10.

Discovery with Constraints Requiring Computation

<table>
<thead>
<tr>
<th>Prob ID</th>
<th>Result</th>
<th>Brief Description</th>
<th>Artifacts Created or Modified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2b:</td>
<td>Level 2 (Only model and data instance changes)</td>
<td>Find services that can ship packages with specific dimensions and weight to addresses in: Goal B1: Tunis, Tunisia. Goal B2: Bristol, England. (Discovery based on two constraints – destination and weight)</td>
<td>As for problem 2a above. Axiom to define the calculation of dimensional weight. Ontology instances for European cities and axioms to define that a given city is in a given country.</td>
<td>The WSMO capability definitions of both goals and web services allows for a conjunction of multiple constraints. The notion of dimensional weight is applied to bulky objects that may be physically light wrt their size. It is calculated using the formula ((L^*W^*H)/166). The maximum value of weight and dimensional weight is the value considered.</td>
</tr>
</tbody>
</table>

Figure 6.11: Evaluation of Discovery with Constraints Requiring Calculation

Two problem goals are defined. The first defines a package whose nominal weight in pounds (lb) along with the required shipping destination are constraining factors with respect to the five available shipping services. The second goal involves shipping a package whose calculated *dimensional weight* is greater than the nominal package weight. The results of the two goals are defined in Figure 6.11.

Discovery with Constraints Requiring Invocation

<table>
<thead>
<tr>
<th>Prob ID</th>
<th>Result</th>
<th>Brief Description</th>
<th>Artifacts Created or Modified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2c:</td>
<td>Level 1 (Code changes required)</td>
<td>Find services that can ship packages with specific dimensions and weight and a constraint on the price to addresses in: C1: New York, USA. C2: Bristol, England. (Discovery based on two static constraints and a further constraint that requires service invocation during discovery)</td>
<td>As for problem 2a, 2b above. An axiom – isCityOnContinent is defined. The Mueller web service description includes a data-fetch interface to allow price information be retrieved from the service by invocation. The other web services have flat fees based on destination continent. Axioms are used to define instances of a PriceQuoteResp concept including rules to calculate the price of a shipment based on the values provided with the goal.</td>
<td>The success level is 1 because a lifting/lowering adapter was implemented for the Mueller web service and this required a rebuild (see notes). The other changes were only to the descriptions of the various artifacts and correspond to success level 2. Modelling the web services for this part of the challenge involved a more significant use of the WSML axioms.</td>
</tr>
</tbody>
</table>

Figure 6.12: Evaluation of Discovery with Constraints and Invocation

Three new shipment goals are defined. In each case the constraints include package weight, destination and a maximum allowed price. The five web services have different pricing strategies. The Mueller service
requires a service invocation to provide a price for a specific shipment – this can be modelled using a
data-fetch interface as described in Section 5.1. The other four each use individual formulas to calculate
the shipping price based on information provided with the shipping request. A summary of the evaluation
of these goals is shown in Figure 6.12.

For example, Goal C3 requires a single package to be delivered to Bristol with dimensions of 10x2x3
inches, a package weight of 20lbs, and a required price of less than $120. Listing 6.13 shows the WSML
goal capability consisting of a postcondition defining the required price constraint and that that isShipped
relation must hold as true.

```
1 ... 
2 postcondition 
3 definedBy 
4 ?x[sop#price hasValue ?price] memberOf sop#PriceQuoteResp and 
5 ?price < 120  
6 sop#isShipped(shipmentOrderReq).  
7 ... 
```

Listing 6.13: WSML Postcondition for Goal C3

The Racer, Runner, Walker and Weasel web services ship to various countries and have flat rates
+ a charge per lb weight + an additional collection charge. The rates vary by continent. The price
calculation for these services is defined using axioms. For example, Listing 6.14 shows the axiom for
the Racer service for calculating the shipment price of a package in Europe. (Note: the ceil axiom is a
built-in axiom in the reasoner for rounding up the weight value to the nearest integer.)

```
1 ... 
2 axiom shippingEuropeDef 
3 definedBy 
4 priceQuoteEurope[sop#price hasValue ?price] memberOf sop#PriceQuoteResp 
5 := 
6 ?shipmentOrderReq[sop#to hasValue ?temp] memberOf sop#ShipmentOrderReq and 
7 ?temp[so#address hasValue ?to] and 
8 ?to[so#city hasValue ?city] and 
9 so#cityIsOnContinent(?city, so#Europe) and 
10 ?package [so#weight hasValue ?weight] memberOf so#Package and 
11 builtin#ceil(?weight, 7c) and (?price = ( ((?c + 6.75) + 41) + 12.50)). 
12 ... 
```

Listing 6.14: WSML Axiom for Price Calculation – Racer Web Service
Listing 6.15: WSML Transition Rules and Axiom for Mueller Service

The Mueller service only provides package-shipping prices through invoking their web service. This is modelled using an additional contracting interface in the Mueller WSML description. The transition rules state that if a shipment order request is present and the `isShipped` relation is evaluated to be true then an instance of a `PriceQuoteResp` will be made available to the requester. Both the transition rules and the definition of the isShipped relation are shown in Listing 6.15. The definition of the constraints prescribed for the Mueller service are included in the the definition of the `isShipped` axiom. This is true of each of the five shipping services.

Discovery Based on Detailed Product Specifications

The discovery scenario changes to one of purchasing computer hardware using the most appropriate web service. Tabular and WSDL-based descriptions of three web service providers are provided. The products sold across the providers are similar with slight differences in specification and price. Each item is identified by a global product ID (base on Global Trade Identification Numbers (GTIN))\(^{35}\).

\(^{35}\)http://www.gtin.info/ (accessed 13 Nov 2009)
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

<table>
<thead>
<tr>
<th>Prob ID</th>
<th>Result</th>
<th>Brief Description</th>
<th>Artifacts Created or Modified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>3a: Goals A1, A2</td>
<td>Level 2 (Only model and data instance changes)</td>
<td>Find services that can provide a very specific description of a computer hardware product – Apple MacBook. (Discovery based on a set of very specific constraints including some requiring arithmetic evaluation)</td>
<td>Computer hardware product ontology Web service descriptions for the three service providers A set of data instances for the products offered by each of the service providers Goal descriptions – the capability defines the required product type.</td>
<td>All ontological information was loaded to the WSMX knowledge base at runtime. The capability of the goal was used as a query against this knowledge base yielding the correct result. The only complexity to consider was the greater number of constraints.</td>
</tr>
</tbody>
</table>

Figure 6.13: Evaluation of Discovery with Detailed Product Specification

The first set of goals require the discovery of web services that can provide very specific computer hardware products. Each goal includes multiple constraints that may require mathematical evaluation and/or service invocation during discovery. Although the complexity of the service descriptions is greater, these goals are treated by WSMX exactly as the goals described in Sections 6.5.2 and 6.5.2. A summary of the evaluation of these goals is shown in Figure 6.12.

**Discovery with Detailed Specification including Preferences**

The computer hardware products have attributes (e.g. price) that can be used to rank them for suitability against a given goal. Goals may contain multiple ranking criteria. For example, a goal may be defined such that a laptop should have the maximum RAM at the lowest price.

<table>
<thead>
<tr>
<th>Prob ID</th>
<th>Result</th>
<th>Brief Description</th>
<th>Artifacts Created or Modified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>3b: Goals B1, B2</td>
<td>Level 2 (Only model and data instance changes)</td>
<td>Find services that can provide a very specific description of a computer hardware product – Apple MacBook. If multiple providers discovered, select the cheapest. (Discovery based on a set of very specific constraints including ranking by price)</td>
<td>As for problem 3a above. An ontology for QoS that defines axioms for the concepts of LowerBetter and HigherBetter. The non-functional parameters of the goals are modified to specify the parameter to be used for ranking and the whether the axiom LowerBetter or HigherBetter is used.</td>
<td>The solution to this problem set was possible because simple ranking is supported in the design of the default discovery component. The property of the SSOA design where each of the components themselves is modelled as a service allows the richer QoS Discovery component developed by Vu et al. to be used as an alternative if desired (for more complex problem scenarios).</td>
</tr>
</tbody>
</table>

Figure 6.14: Discovery with Detailed Specification including Preferences

The discovery component of WSMX allows a ranking element to be specified as a non-functional parameter of WSML goal capabilities. The parameter identifies both a variable whose value is to be used for ranking results, along with a definition of LowerBetter (lower value = lower rank) or HigherBetter (higher value = better rank) axioms defined in a quality-of-service (QoS) ontology based on the rich QoS
ontological framework developed by Vu et al. [2006b]\(^{36}\). The evaluation of the goals including preferences is summarized in Figure 6.14.

**Discovery with Service Composition**

<table>
<thead>
<tr>
<th>Composition Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncorrelated</td>
<td>The requests contains several products that may or may not need to be purchased from different providers.</td>
</tr>
<tr>
<td>Correlated</td>
<td>The requests contains several products that may or may not need to be purchased from different providers plus not all pairs of products are suitable and compatible to each others</td>
</tr>
<tr>
<td>Correlated with global preference</td>
<td>The requests contains several products that may or may not need to be purchased from different providers plus there are global optimization goals and constraints. A power minimum or a price maximum are examples of constraints that should not be violated. Within this solution space, if it is specified that as long as functional requirements are met, then the cheapest price is best, then that defines an equivalence class of best solutions.</td>
</tr>
</tbody>
</table>

Figure 6.15: Composition Types Specified by the SWS Challenge

<table>
<thead>
<tr>
<th>Prob ID</th>
<th>Result</th>
<th>Brief Description</th>
<th>Artifacts Created or Modified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>3b:</td>
<td>Level 2 (Only model and data instance changes)</td>
<td>Each Challenge goal requires a combination of services to satisfy it. Three types of service composition are tested (1) unrelated composition, (2) correlated composition and (3) correlated composition with global preferences.</td>
<td>As for problems earlier: an ontology for QoS that defines axioms for the concepts of LowerBetter and HigherBetter. Product ontology sufficient to define the 19 product definitions. An isCompatible relation and a corresponding axiom to define the relation for services that have restrictions on the operation of some of their products with other products (possibly from other service providers). Web service descriptions for the three service providers. Descriptions of the goals specified for composition including the use of non-functional parameters to indicate ranking variables and the use of the isCompatible relation as needed. The non-functional parameters of the goals are modified to specify the parameter to be used for ranking and the whether the axiom LowerBetter or HigherBetter is to be used.</td>
<td>The solution to this problem set reuses many of the techniques from the previous goals. The type of composition is simple and intended to show that the problems can be solved using the SSOA prototype with relatively simple semantic modelling. It is not intended as a fully-fledged automated solution to service composition.</td>
</tr>
</tbody>
</table>

Figure 6.16: Discovery with Composition of Services

The final experimental set of problem scenarios evaluated via the SWS Challenge involves service discovery requiring three different styles of composition listed in Figure 6.15. The Challenge goals are modelled in WSML with capability postconditions having multiple clauses joined by logical conjunction. Each clause defines a variable that is a member of a class with specific constraints that needs to be

\(^{36}\)WSML available at: [http://lsirpeople.epfl.ch/lhvu/download/qosdisc/](http://lsirpeople.epfl.ch/lhvu/download/qosdisc/)
resolved. The clauses are separated by the discovery component and each is attempted to be resolved individually.

For uncorrelated discovery with composition, the first discovered service for each clause is used. Where a ranking is declared for a specific clause, the top-ranked service for that clause is used. Where there are global preferences across all clauses (e.g., get the cheapest combination of services that satisfy the criteria), all matching services and their associated ontology definitions and instances are loaded into a knowledge base used for the discovery. The entire postcondition is then used to form a logical query applied across this knowledge base. A summary of the results for WSMX against the service composition problems is shown in Figure 6.16.

Listing 6.16: WSML Goal Postcondition for Challenge Goal C3

Listing 6.16 lists the postcondition element of the WSML goal representing the Challenge goal C3. In this case the three clauses correspond to the resolution of the three variables ?x, ?y and ?z, where ?x is a member of the Notebook class, ?y a member of the WebCam class and ?z is a member of the Accessory class. Note that the design for composition used by the WSMX SSOA implementation is described in Section 5.2. Section 6.4 walked through an example based on resolving the Challenge Goal C2 (correlated composition).

The order in which the services are invoked is assumed to be unimportant. In many cases an indeterminate service invocation sequence will not be useful for business processes. (Section 6.4 cites relevant research into more rigorous composition techniques.) However, in the context of the scenarios presented
CHAPTER 6. SSOA IMPLEMENTATION & EVALUATION AGAINST INDEPENDENTLY-SET INTEGRATION SCENARIOS

by the Challenge, the WSMX SSOA solution was sufficient to solve the problems. The approach is simple and illustrative. A QoS discovery module designed and implemented by Vu et al. [2006b], is available to plug into the WSMX SSOA prototype illustrating that minimal effort is required to provide rich component implementations as long as they adhere to the defined interface for that component type.

6.6 Summary

This chapter provided a detailed description of how the WSMX open-source implementation of the SSOA reference architecture was applied to a set of independently set software integration problems posed by the SWS Challenge. It demonstrates a bottom-up experimental evaluation of the SSOA reference architecture described in Chapter 4. The SWS Challenge provided a pragmatic set of problems and a robust experimental test bed to allow candidate solutions be evaluated on how they could handle changes to specific real-world scenarios using semantics without necessarily having to re-write executable code for each change. Of the ten participants in the Challenge, the WSMX approach was evaluated as having solved the second largest number of problems. Only the DIANE project from the University of Jena\(^\text{37}\) solved more while the joint SWE-ET project from the Politecnico di Milano\(^\text{38}\) and CEFRIEL\(^\text{39}\) solved only two less. Chapter 7 compares these three approaches.

\(^{37}\)http://hmnp.inf-bb.uni-jena.de/DIANE/
7. Comparative Evaluation of WSMX as an SSOA to Peer Systems
- DIANE and SWE-ET

This Chapter describes and evaluates two Semantic Web service projects which provide solutions to the problems posed by the Semantic Web Services Challenge (SWS Challenge)\textsuperscript{40}. They are the DIANE project\textsuperscript{41} [Kuester et al., 2007] from the Friedrich Schiller Universitaet (FSU) Jena, and the SWE-ET project, which is a combination of the Glue discovery engine [Valle and Cerizza, 2005] from CEFRIEL\textsuperscript{42} and the WebRatio framework [Manolescu et al., 2005; Brambilla et al., 2007] from the Politecnico di Milano\textsuperscript{43}, for the discovery, mediation, composition and invocation of Semantic Web services.

Both frameworks were founded on research into service description, publication and discovery. Initially DIANE started with the OWL-S ontological model while Glue started with WSMO. They subsequently evolved to either (1) create a new optimized conceptual model for discovery (DIANE) or (2) extend the WSMO model (Glue) to enhance their solutions to discovery. Additionally, in the course of active participation in the SWS Challenge, both frameworks grew from only providing solutions to discovery, to include solutions to mediation, composition and invocation of Semantic Web services. The Chapter has the following structure. Each of DIANE and SWE-ET are described in terms of (1) the data model they use, (2) their architecture, and (3) a brief evaluation of their results in applying their research to the three SWS Challenge-problems of mediation, discovery and composition. Finally, there is a comparison of the approaches of DIANE, SWE-ET to that of WSMX (as described in Chapter 6) to the SWS Challenge problem set.

\textsuperscript{40}http://sws-challenge.org/ (accessed 10 Mar 2011)
\textsuperscript{41}http://hnsp.inf-bb.uni-jena.de/DIANE/ (accessed 10 Mar 2011)
\textsuperscript{42}http://glue.cefriel.it/ (accessed 10 Mar 2011)
\textsuperscript{43}http://www.webratio.com/ (accessed 10 Mar 2011)
7.1 DIANE

DIANE is the project name given to an integrated approach to service discovery, mediation, composition and invocation from the Friedrich Schiller Universitaet (FSU) Jena. At the core of DIANE is the DIANE Service Description (DSD) language, for describing Web service offers and requests, and the associated graph-based matching algorithm. The following sections describe the data model and architecture of DIANE and summarizes its usage against the problems published by the Semantic Web Services Challenge.

7.1.1 DIANE: Data Model

![DSD Service Description for a Printing Service](image)

DIANE uses the DIANE Service Description (DSD) language [Klein et al., 2005; Kuester and Koenig-Ries, 2006] (based on earlier work by [Klein and Koenig-Ries, 2004]) to define set-based descriptions of requests and offers for Web services. It places major importance on the definition of service effects and, to a lesser extent, preconditions, and uses these as the primary basis for its discovery algorithm [Kuester et al., 2007]. The definition of elements of DSD in terms of sets of objects is central, where membership of a set can be by type, or by fulfillment of defined properties. The work was initially motivated as an alternative to service matching based on the comparison of service and request signatures, termed signature matching, predominant in OWL-S discovery approaches [Klein et al., 2005]. The input and output messages that comprise the signature are modelled in DIANE using variables and are integrated into the sets defining requests and offers.

A graphical example of a service description is shown in Figure 7.1. In this, a service is defined as
a set of objects with the Shipped effect, where this effect has the attributes toAddress, shippingTime, cargo and price which in turn are also defined as sets of objects. The attribute price is defined as an OUT variable which means that the value for price is provided as an output by the Web service.

The following conditions to define membership of an object set are possible: (1) type, (2) direct conditions, (3) property conditions (in the case of a complex type). This is augmented by strategies (i) a type check strategy e.g. default is ”equals”, but can be ”super” or ”sub”, (ii) connection strategy which changes the way in which the individual results of the property conditions are connected - default is conjunctive, but disjunction and negation are allowed, (iii) missing strategy specifies the behaviour in the case of a missing strategy.

Service offers are frequently configurable which means that service discovery needs to handle this. In DSD both offers and requests can be modelled to include variables. The semantics of including a variable in a request is that the value of this variable should be populated in the course of matching the request to offers. Additionally DIANE focuses on the likelihood that different offers may match requests with different degree of exactness. They allow requests to be formulated so that properties of effects can be specified in terms of preferences e.g. price $<=$ 50 dollars, preferably lower.

7.1.2 DIANE: Architecture

The DIANE architecture, used to solve the SWS Challenge problems, is shown in Figure 7.2. It consists of the DSD Middleware, a RosettaNet translator on the left and a WSBPEL process execution engine on the left. The DSD Middleware is the central component of Figure 7.2. It consists of the DSD Matcher Module which implements the discovery algorithm. The DSD Matcher includes the DSD Planner Module as a sub-component. This feeds into the discovery process where a request has multiple independent requests (as is the case in the composition scenario of the Challenge). The DSD Execution module is responsible for interpreting and using the WSDL Grounding description to (1) mediate data as necessary before a service invocation, and (2) make the service invocation and return the result as DSD instance data to the owner of the service request.

The RosettaNet-DSD translator on the left hand side is responsible for translating RosettaNet XML requests into corresponding DSD requests, and for translating DSD response instances into corresponding RosettaNet XML messages. This corresponds to a type of Lifting/Lowering adapter, in terms of the SSOA model. On the right hand side is the service provider, responsible for providing the DSD service description, including a WSDL grounding description, and for implementing the service endpoints. The WSDL grounding is the part of a DSD service description that links the description to a WSDL service definition. The grounding specifies the service endpoint, as well as an XML template which is used to
create the message that is required by the Web service (it is assumed that the Web service invocation requires a single atomic message). The grounding also defines mappings: (1) from the variables included in the DSD service description to elements of the WSDL message expected by the Web service and (2) from the elements of the WSDL message returned by the Web service to variables in the DSD service description. It’s also possible to include rules that perform data mediation in the grounding definition or to carry out this type of mediation through the invocation of a specified translator Web service.

A WSBPEL process execution engine is included on the right hand side of Figure 7.2. DSD supports simple stateless choreography for Web services and does not provide direct support for the description of services that require the exchange of more than one message in their public interface (defined by the service choreography element in WSMO). This is a requirement for the mediation scenario of the SWS Challenge. To overcome this, a WSBPEL process is manually created to mediate between the single in and out message per service, which the DSD middleware can send, and the multiple in and out messages required for the scenario.

7.1.3 DIANE: Mediation

Broadly speaking, data mediation is treated by the DIANE middleware as a part of the discovery problem. It occurs in two places. The first is as a preprocessing step using a lifting/lowering adapter such as that shown in Figure 7.2. This translates between non-semantic representation, e.g. a RosettaNet XML message and a semantic service request in DSD. It is the equivalent of the SSOA Lifting & Lowering Adapters (see Section 4.4.3). The implementation of this adapter for the SWS Challenge was as a Web service coded in Java. The second place for data mediation is in the grounding, through (1) the inclusion of mapping rules or (2) the declaration of external mediation services in the grounding description which links a DSD offer to a WSDL service description. This limited support for data mediation has the
consequence that the lifting & lowering adapters used by the DIANE middleware typically must be designed to both (1) lift or lower between semantic and non-semantic representations of data, and (2) translate the data into terms of a domain which best suits the Web service whose offers are described and available in DSD.

As mentioned in the previous section, the DIANE middleware best supports service offers that do not have complex choreographies – in other words, ones that are defined with a single input and output. As DSD does not handle stateful choreographies, WSBPEL is used to handle the mediation between (1) the public process for handling the sending of a RosettaNet request and the receipt of the response, and (2) the public process exposed by the Web services of the Moon company, defined in the SWS Challenge. When the Challenge scenario for the mediation problem changed, DIANE could provide a solution by adjusting the WSBPEL process definition to take account of the changes. The external interface to clients of the WSBPEL process did not need to be changed.

7.1.4 DIANE: Discovery

The primary focus of DIANE is on service discovery. Both offers and requests are specified as graphs with the assumption that any required data mediation has already taken place so that the graphs stem from similar ontological concepts. The matching algorithm, first published in [Klein and Koenig-Ries, 2004; Klein et al., 2005], walks through the tree beginning with the root element. Graphs for the request and each respective service offer are compared node by node. One constraint is that the request graph should not contain any cycles as the comparison is driven by the structure of the graph. The matching coefficient of leaf nodes are calculated based on the fuzzy containment value of the offer element set in the corresponding request element set. The matching values of inner nodes of the graph are calculated from the matching values of their child nodes. Overall the match between a request and an offer is normalized to the range [0,1] - the closer to 1, the better the match.

An extension to the original discovery algorithm is made to DIANE for the problems posed by the SWS Challenge. In these problems, some information necessary for matching is only available by runtime invocation of the service. DIANE tackles this by modelling two types of service interaction – estimation and execution. Estimation interactions are for the purpose of data retrieval. Execution invocations trigger the effects defined for the service to be realized.

Rating elements are used in DSD service requests to define sets for effects by weighted calculation based on the value of variables populated by the individual offers. This provides a flexible way of defining membership for the sets that constitute the required effects of a request. In [Kuester et al., 2007], the example provided is of a shipping element set where the weighted (toward the price) rating calculation
is provided by the formula: $0.3 \ast \text{shipping time} + 0.7 \ast \text{price}$ (see Figure 7.1). This rating value is
used in selecting the best-fit offer where multiple offers match the request to some degree.

The algorithm is efficient but depends on the graph structure, for both the request and offers, at least
stemming from the same ontological type. It also requires that child elements of the graph match-off
ontologically. Its possible that a request and offer may each have multiple effects, and also possible
to specify a connecting strategy to determine whether or not a match is successful in this context.
Additionally, through modelling the message flow of the signatures using variables, the approach allows
for services to be invoked after discovery.

7.1.5 DIANE: Composition

DIANE again extends its approach for service discovery to handle requests with multiple effects. Re-
lations between effects can be modelled in DSD. An example provided in [Kuester et al., 2007], repro-
duced here, illustrates this. A company wishes to order screws and have them delivered to Karlsruhe
in Germany. The screws have to be manufactured and shipped - two effects. There are many screw
manufacturers and many shipping companies. The effects are linked as it would not make sense for the
manufacturer to be geographically far removed from shipping company. Additionally, the manufacturer
may be able to satisfy both effects but may have different plants in different countries with one being
cheaper than the other but slower to deliver. An intuitive approach is to take every possible combination
of service offers that satisfy the effects and evaluate the one that best fits. As the authors of DIANE
point out in [Kuester et al., 2007], this leads to a complexity of $O(m_1 \ast m_2 \ast m_3 \cdots \ast m_n)$, where $n$
is the number of effects and $m_i$ is the number of offers for effect $i$, which can be approximated to $O(m^n)$
where $m$ is the maximum value of $M_i$. This is not tractable if $m$ becomes a large value.

The solution of the DIANE middleware is to prioritize effects and to use value propagation. Value
propagation means locally computing the optimal solution for one effect and storing the values of shared
variables in memory before propagating to the next effect. The requester specifies how optimal they
consider effects. As a result, the chosen effect in each phase may constrain the values used when matching
subsequent effects. This is a trade-off that the authors argue is a pragmatic choice. For example, the
shipping effect in the previous example is matched first and then the location of the shipping origination
is used as an input to the effect representing the manufacture of the screws. This approach scales linearly
with the number of effects to evaluate with the trade off that there is no global optimization.
7.2 SWE-ET

The SWE-ET project is a combination of (1) the Glue engine [Valle and Cerizza, 2005] and (2) the WebRatio framework [Ceri et al., 2000; Manolescu et al., 2005; Brambilla et al., 2006b] for the discovery, mediation, composition and invocation of Semantic Web services. The Glue engine was developed as a mediation-centric approach to service discovery using an extension to WSMO as the conceptual model. It is based on the assumption that, when Semantic Web service descriptions are published, the domain ontologies used in the semantic annotations can be expected (for various reasons) to vary across service providers. It deploys a mediator-centric approach to tackle this heterogeneity, aligning with the core WSMO principles (1) strong decoupling and (2) scalable mediation [Roman et al., 2005; Fensel and Bussler, 2002]. To solve the additional mediation and composition problems, presented by the SWS Challenge, SWE-ET combines Glue (for discovery) with the Computer Aided Software Engineering (CASE) techniques of the WebML language and the WebRatio execution environment (for selection, mediation and composition).

7.2.1 SWE-ET: Data Model

Glue extends the WSMO model with the definition of meta-classes for goals and Web services. This enables the definition of pre-defined parametrized goals that represent classes of individual goal instances. For example a goal class could be defined to request shipping services. Goal instances belonging to that class could include a goal to ship from Dublin, Ireland to all EU countries, a goal to ship from Dublin, Ireland to China etc. Clients of Glue are responsible for locating a pre-defined goal, as required, and providing concrete values for its parameters. Thus, a pre-defined, parametrized (meta) goal defines a set of concrete goals. The same relationship is defined between a pre-defined, parametrized (meta) Web service and a corresponding set of concrete services. With this extension in place, Glue models the mediator-links between (1) goals and other goals, (2) goals and Web services, (3) differing ontologies, and (4) Web services and other Web services using the set of mediators defined by WSMO (ooMediator, ggMediator, wgMediator, and wwMediator (see Appendix A.1)). Glue makes the role of WSMO mediators explicit. ggMediators automatically generate goals that semantically equivalent to that expressed by the requester but possibly using different ontologies or in a different form. wgMediators are responsible for matching goal and Web service descriptions. ooMediators are responsible for data mediation.

Glue uses F-Logic as its knowledge representation and ontology language along with the Flora-2\textsuperscript{44} plugin for the XSB reasoning engine\textsuperscript{45}. The model for goals and Web services is shared with the WebML/WebRatio framework. Although this is principally to enable WebRatio to (1) provide a GUI for

\textsuperscript{44}http://flora.sourceforge.net/ (accessed 10 Mar 2011)
\textsuperscript{45}http://xsb.sourceforge.net/ (accessed 10 Mar 2011)
clients to create and dispatch goals to Glue, (2) select the best Web service from candidates matching the goal and (3) enable the invocation of Web services, either to achieve the goal or as a part of discovery process. WebML is a high-level graphical notation for a data- and process-centric language initially designed for defining Web applications using a formal notation that can be used to auto-generate code. The data model used for applications defined in WebML is based on an extended entity-relationship model. Of relevance to this discussion is that this model, in turn, is extended to handle WSMO concepts that is relevant to this discussion. An example of modelling a WSMO mediator directly using WebML can be found in Figure 8 of [Ceri et al., 2007].

### 7.2.2 SWE-ET: Architecture

![Glue Middleware](image)

Figure 7.3: Glue Middleware [Valle and Cerizza, 2005]

The SWE-ET architecture is a combination of the (1) Glue middleware, responsible for Semantic Web service discovery and mediation, and the (2) WebML/WebRatio framework responsible for Semantic Web service selection, invocation and composition. The architecture of Glue is shown in Figure 7.3 (note: the image is reproduced from the original at [Valle and Cerizza, 2005] which has a low resolution). The
Communication Manager provides a public interface for publishing Semantic Web service descriptions, for the submission of client goals, and for returning the results of the matches. The Constructors are responsible for translating the XML messages into a WSMO representation. The Goal Translator mediates user goals into pre-defined goals (using ggMediators) that match the Web services available to Glue. Finally, the Proof Generator uses the ontology repository and the various mediator repositories (ooMediators and wgMediators) to match a concrete goal to Web service, carrying out whatever data mediation is necessary.

[Ceri et al., 2000] defines WebML as a conceptual model for data-intensive Web applications. It was devised as a language that could separate the orthogonal concerns of content, interface logics and presentation logics. Using WebML a Web application developer can model the various aspects of their application separately, link these using the GUI provided by WebML/WebRatio and then use this model to automatically compute a set of Web pages made up of logical components connected by logical links. [Ceri et al., 2007] describe how the deployment technologies for WebML have evolved from unstable to mature stable packages. Consequently WebML provides a rich platform that has been extended to include support for the concepts required for (1) Web service and SOA and (2) ontological data models. It is this support, combined with the Semantic Web service discovery functionality provided by Glue, which allows SWE-ET provide effective solutions to the SWS Challenge problem set.

Figure 7.4: Combined Glue and WebML/WebRatio Architecture [Brambilla et al., 2007]
Figure 7.4 provides a high level view of how the combined Glue and WebML/WebRatio framework work together. A Goal Creation/Composition GUI is used to create a semantic goal which is sent to the Glue discovery engine. If necessary, the engine can negotiate with potential matching Web services by invoking operations on their interfaces and feeding the results back to the Proof Generator of Glue. Once a set of matching services has been discovered, this is sent to the WebRatio Selection UI. Data mediation may be carried out as a WebML process step before the selected Web service is invoked using the Service Invocation component.

SWE-ET uses WebML/WebRatio to provide a graphical user interface (GUI) for the selection, invocation and composition of Web services. The richness of the WebML formalism allows for the various steps required to solve the mediation problem (both data and process) of the SWS Challenge to be encoded using WebML and deployed as an executable process using WebRatio. To support Web service usage in Web applications, WebML was extended to represent different types of Web service data exchange that grounded to Web service WSDL descriptions. Latterly, as part of the activities for the SWS Challenge, WebML was further extended to include primitives for the WSMO model and other primitives for ontology querying.

7.2.3 SWE-ET: Mediation

Both data and process mediation are handled by SWE-ET by (1) first modelling the problem scenario using BPMN, (2) translating BPMN to WebML, (3) fine-tuning the WebML via a rich user interface, and (4) executing the modelled process via the WebRatio framework. [Brambilla et al., 2006b] describe this as an emphasis on the use of Web engineering methods for the high-level specification of applications featuring business processes and remote services invocation. In addition to the data mediation possible during discovery by the Glue middleware, data mediation can also be defined through the modelling of WSMO mediators using WebML and the inclusion of this in an overall WebML process description.

SWE-ET’s approach to process mediation [Brambilla et al., 2006a] is based on adapting the existing WebML software methodology to support Semantic Web service concepts. This includes the integration of Glue to support semantic Web service discovery, and the inclusion of primitives for modelling and querying ontologies inspired by SPARQL\(^46\) and RDF-S\(^47\) e.g. WebML units for SubclassOf, InstanceOf, HasProperty etc. [Ceri et al., 2007]. It also allows for the modelling of WSMO mediators. The result is that a business process designer can use the software engineering methodology and graphical user interface of WebML to design solutions to software integration problems using Semantic Web service technology. If the scenario changes (as is the case in the SWS Challenge), the WebML model has to be

\(^{46}\)http://www.w3.org/TR/rdf-sparql-query/ (accessed 10 Mar 2011)  
\(^{47}\)http://www.w3.org/TR/rdf-schema/ v1.0 (accessed 10 Mar 2011)
updated. However the level of abstraction provided by WebML, its graphical nature and the fact that it can be executed directly using WebRatio help to minimize the required effort.

### 7.2.4 SWE-ET: Discovery

![Figure 7.5: Glue Discovery - Setup Phase [Valle and Cerizza, 2005]](image)

Discovery for the SWE-ET framework is provided by the Glue middleware [Valle and Cerizza, 2005] through the definition and use of WSMO mediators. There are three phases: (1) setup, (2) publishing, and (3) discovery. The setup phase is illustrated in Figure 7.5. It is assumed that service requesters and providers each use different ontologies in the same domain to model their data. In Figure 7.5, the domain is assigned the letter $D$, the different models of the domain are denoted as $D^+$ and $D^-$, and the corresponding ontologies are labelled $Onto_{D^+}$ and $Onto_{D^-}$ respectively. There are two steps to complete during the setup phase. The first is that a Semantic Web services expert creates semantic descriptions for (1) the Web service classes and (2) the goal classes, which use these ontologies. The second step is where a domain expert creates the set of mediators required to link goal classes to Web service classes ($wg$Mediators), goal classes to other goal classes ($gg$Mediators), and semantically equivalent terms in the ontologies for $Onto_{D^+}$ and $Onto_{D^-}$ ($oo$Mediators).

The publishing phase is where Semantic Web service and goal descriptions are defined and published to the Glue middleware. The Communication Manager component of Glue, shown in Figure 7.3, provides the interface which allows Web services descriptions to be published and goals to be submitted. To publish a concrete service, a service provider selects a suitable Web service class and provides values for any required parameters. The resulting Web service instance description is stored in the Glue Web service repository.

At discovery time, a goal instance is created and submitted to the Glue middleware. First, the class
of goals that this instance belongs to is determined. Next, ggMediators are used, if necessary, to translate the goal into other semantically-equivalent goals (for which matching Web services are defined). The matching function is provided through wgMediators, each of which consists of a set of F-Logic rules. When a specific instance of a goal is presented, each rule of the wgMediator is applied on each candidate Web service instance. Depending on which rules are satisfied, an aggregate matching value is produced. This value is used to rank the matched Web services in order. In the course of the matching, ooMediators are used to overcome any data heterogeneity.

7.2.5 SWE-ET: Composition

SWE-ET enables the modelling of composite processes through extending WebML to support elements based on the Business Process Modelling Notation (BPMN). For the SWS Challenge, three types of service composition problem were defined (1) uncorrelated composition, (2) correlated composition and (3) composition with optimization (see Section 6.4. The problems were designed to encourage the Challenge participants build solutions where the minimal amount of change is required when a new composition problem is introduced or an existing problem changed. SWE-ET provided a solution that combined its WebML process modelling elements with the extensions allowing the resolution of semantically-described goals to be included. It demonstrated a solution to the first problem type, uncorrelated composition, based on using a ggMediator to split a composite goal into two subgoals and then using the combination of Glue (for discovery) with WebML (for selection, mediation and invocation) to resolve the two goals independently and then aggregate their results. Solutions for the other problems were not presented but can be achieved through direct WebML modelling for each individual problem scenario.

7.3 Comparison of DIANE, SWE-ET and WSMX

This section compares DIANE, SWE-ET and WSMX in terms of (1) their architectural model with respect to the requirements for an SSOA architecture, (2) their peer-led evaluation at the SWS Challenge, and (3) their approaches to each category of problems published by the SWS Challenge.

7.3.1 Comparison: Architecture

Figure 7.6 summarizes how the architectures of each of the three frameworks relate to the functional and non-functional requirements for an SSOA described in Section 3.2. The meaning of the symbols in the table are as follows: + means the requirement is supported; - means the requirement is not supported.

---

## Architectural Principles

### Functional Requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>WSMX</th>
<th>DIANE</th>
<th>SWE-ET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability-driven discovery</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Service selection</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Dynamic data mediation</td>
<td>+</td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>Choreography interpretation and execution</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic process mediation</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Logical reasoning support</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Compatible with WSDL Web services</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

### Non-functional Requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>WSMX</th>
<th>DIANE</th>
<th>SWE-ET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reusable across problems and domains</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Loosely coupled with standardised interfaces</td>
<td>+/-</td>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>Simple programmatic interface</td>
<td>+</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td>Scalable &amp; extensible</td>
<td>+</td>
<td>++/-</td>
<td>+/-</td>
</tr>
<tr>
<td>Goal oriented</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 7.6: SSOA Requirements vs WSMX, DIANE and SWE-ET

Supported; +/- means the requirement is partially supported and; ++/- means that the requirement is strongly supported for one aspect of the solution. WSMX was designed from the outset to be a reference implementation for SSOA. Consequently, Figure 7.6 shows that it meets all the requirements by having the + symbol on every row.

### Functional Requirements

All three frameworks support (1) capability-driven discovery and selection, (2) evaluation of the semantics of ontological descriptions through the use of logical reasoning engines, and (3) the grounding of the semantic models used to the standard WSDL description language for Web services. However, there are different levels of support for dynamic data mediation, choreography description and execution, and process mediation.

WSMX supports dynamic data mediation through the application of ontology-to-ontology data mappings as needed during any phase of discovery, selection and invocation. The data mappings are created independently of the semantic modelling of individual Web services and goals, and stored in the Semantic Data Mapping Store (see Figure 4.5 in Chapter 4). SWE-ET supports dynamic data mediation in both the Glue middleware and the WebML/WebRatio framework. In Glue, it is possible through the definition of F-Logic rules in ooMediators available for use during discovery. It is also possible to define ooMediators in WebML which can be included as part of Web service invocation. DIANE partially supports dynamic data mediation through the inclusion of rules in Service Grounding descriptions. However, any required data transformation during the invocation of Web services is carried out using syntactic XSLT transformations between XML documents.

In contrast to WSMX, neither DIANE or SWE-ET support the description of complex stateful
CHAPTER 7. COMPARATIVE EVALUATION OF WSMX AS AN SSOA TO PEER SYSTEMS - DIANE AND SWE-ET

service choreography interfaces. They assume that the invocation of a service is atomic. Both provide workarounds by modelling multiple interactions, between either DIANE or SWE-ET and individual Web services, using an external process definition language – WSBPEL in the case of DIANE and BPMN-compliant WebML units in the case of SWE-ET. For dynamic process mediation, WSMX uses the rule-based semantic descriptions of goal and Web service choreography interfaces to enable mediation between mismatching process models to be carried out at runtime (subject to the constraints identified by [Cimpian, 2007], reproduced in Figure 4.17. In contrast, process mediation in both DIANE and SWE-ET is carried out through the manual modelling of the overall process for each scenario using WSBPEL and WebML respectively.

Non-Functional Requirements. All three frameworks are (1) reusable – as they are agnostic to the domain of problem scenarios to which they can be applied and (2) support interoperability across programming languages and operating systems – through grounding mechanisms to standard WSDL service descriptions.

Having open, clearly-defined component interfaces allows the functional components that make up an SSOA to be loosely coupled with each other. This promotes scalability and eases the possibility of replacing components with a minimal impact on the rest of the system. In this regard, the SWE-ET framework has an open interface to the Glue discovery engine which is also compatible with the WSMX discovery component interface. The approach to the functionality required for selection, mediation and invocation functionality is different. Each is manually modelled using WebML and the respective components are then automatically generated for execution by WebRatio. The advantage is the powerful user interface which makes it easier for business analysts, who may not have software development skills, to create the process they want in WebML and have this automatically interpreted into an executable form by the WebRatio engine. The potential drawbacks are (1) the need for manual involvement during selection and (2) that there may be limited scope to replace the execution code for individual units with alternative implementations, without revisiting the WebML model. DIANE publishes a clearly defined interface for the acceptance of DSD requests by the DIANE middleware. However, the functionality for discovery, selection, mediation and invocation are aggregated in the DIANE middleware. It may be possible for these subcomponents to be updated without the need for a redesign of the DIANE middleware but this is not clear from the documentation.

WSMX defines a simple interface to access the three SSOA behaviours of (1) service discovery, (2) receiving data and (3) combined service discovery and invocation. The benefit of having a simple interface is that it lessens the knowledge required of client applications to interoperate with the SSOA. The mix in both DIANE and SWE-ET of (1) a dedicated semantic discovery component with (2) a
manual definition of a process description for mediation and invocation, mean that both partly support
a simple programmatic interface to their overall semantic SOA architecture.

Of the three solutions, DIANE uses an optimized matching algorithm for Web service discovery with
value propagation semantics [Kuester et al., 2007] to enable efficient discovery of a composition of services
against a request with multiple related effects. The authors provide experimental evidence that their
approach is scalable and efficient enough to be used for dynamic service matching in real-world scenarios.
The scalability of the other components of the DIANE framework are lower as they require per-scenario
modelling of WSBPEL processes to enact process mediation and the invocation of the Web services
themselves.

Glue offers scalability through the explicit and widespread usage of mediators as part of service
discovery. However there are trade-offs to scalability for SWE-ET overall. The first is that an F-Logic
rule is required for every attribute that is compared as part of a wgMediators internal logic. Depending
on the number of attributes, the set of rules can become very large. Secondly, the rich graphical user
interface offered through WebML for service selection, mediation and invocation means the overall process
of service discovery and invocation is only partially automatable, limiting scalability.

The overall architecture of WSMX, following the SSOA model, was designed with scalability in mind.
This is reflected in (1) the event-driven communication model (Section 4.6), (2) the separation of concerns
into individual components (Section 4.4), (3) the goal-oriented execution model (Section 4.5.3), and (4)
in the inherent support for data and process mediation (Sections 4.4.4 and 4.4.4) – necessary in its
adoption of the WSMO conceptual model as its basis.

The final non-functional characteristic of interest is whether the framework is goal-oriented. Goal-
oriented means that the purpose of the system is to satisfy goals that service requesters have. It leads
to the requirement of a system to allow a service requester model what they want to achieve from their
perspective independently of how service providers model Semantic Web service descriptions. It also leads
to the requirement for mechanisms to mediate any heterogeneity between the semantic descriptions of
the respective data and process models. Both SWE-ET and WSMX use WSMO as the basis of their
conceptual data model. The modelling of goals and mediators are fundamental elements of WSMO which
both SWE-ET and WSMX support. The data model used by DIANE allows for service requests to be
modelled independently of service offers in DSD. It also incorporates a mechanism for the inclusion of
rules, and callouts to external services, to mediate data mismatches during the discovery process. In
this regard, DIANE supports the concept of goal-oriented discovery, but arguably to a lesser extent than
either SWE-ET or WSMX.
### 7.3.2 Comparison: Experimental Evaluation

<table>
<thead>
<tr>
<th>Problem Level</th>
<th>DER/AT &amp; DER/ET (Solution Details)</th>
<th>FSU Jena (Solution Details)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. static mediation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1a. changes data mediation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1b. changes process mediation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1c. Discovery Based on Destination</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2a. Discovery Based on Destination and Weight</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2b. Discovery based on Destination, Weight and Price</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2c. Discovery involving simple Composition</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2d. Discovery including temporal reasoning</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3a. Discovery based on clear defined product specifications - Goal A1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3b. Discovery based on clear defined product specifications - Goal A2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3c. Discovery 3C - Additionally specify preferences - Goal B1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3d. Discovery 3C - Additionally specify preferences - Goal B2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3e. Discovery 3C Composition of services - Goal C1 (unrelated composition)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3f. Discovery 3C Composition of services - Goal C2 (correlated composition)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3g. Discovery 3C Composition of services - Goal C3 (unrelated but global condition)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3h. Discovery 3C Composition of services - Goal C4 (unrelated with global condition and preferences)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 7.7: SWS Challenge Evaluation (from http://sws-challenge.org/ (accessed 10 Mar 2011)

The table of Figure 7.7 shows the results of evaluating the three frameworks against the mediation and discovery problems of the SWS Challenge. A tick indicates the problem could be solved. The table illustrates that all three frameworks were successful in solving the majority of the published mediation, discovery, and combined discovery & composition problems. However, it is not possible to determine an outright best solution, in terms of problems solved, as each participant have their individual trade-offs. These are indicated by the superscripted numbers.

Superscripted numbers above the ticks indicate more specific evaluation results. 1 indicates that only adapters had to be changed for the framework to be able to solve the problem. 2 indicates that there was an aspect of the changed scenario that was not handled correctly in the solution. 5 means that arithmetic calculations had to be carried out outside the framework. 6 indicates a change was required to the abstract code model (WebML in this case). 8 indicates that the algorithm was judged to be correct but incomplete.

### 7.3.3 Comparison: Discovery

Each one of DIANE, SWE-ET and WSMX provide effective solutions to discovery based on semantic annotations of service requests and service descriptions i.e. they each distinctly model the perspective
CHAPTER 7. COMPARATIVE EVALUATION OF WSMX AS AN SSOA TO PEER SYSTEMS - DIANE AND SWE-ET

of the service requester and provider. Discovery is carried out based on what the Web services can do (their postconditions and/or effects) rather than on the data types used in their inputs and outputs. In addition, all three augment static matching through support for the fetching of additional data, required to fully determine success or failure of a match, from Web services at runtime.

The section is divided into two parts. The first compares WSMX and SWE-ET, as both use the same underlying conceptual model. The second part compares WSMX with DIANE. Note that a comprehensive comparison of discovery between DIANE and SWE-ET is available at [Kuester et al., 2008].

WSMX and Glue

Both WSMX and GLUE use a three-stage approach to Semantic Web service discovery. Stage 1 is an abstract level match where only the description of the desired and provided Web service capabilities are taken into account. Stage 2 is instance-based matching where any data that is made available with the goal is mediated if necessary and then applied to variables in each service description. The corresponding reasoner (IRIS\textsuperscript{49} for WSMX, Flora-2 XSB plugin for Glue) is used to determine matches. Stage 3 involves querying the Web service for additional data if instance-level matching is insufficient.

Although, both WSMX and Glue both use WSMO as the conceptual model, there are differences in how each approaches the three stages. Abstract-level matching (stage 1) in Glue is achieved through the identification of wgMediators that link the given goal to a set of Web services, without yet taking any input data into account. WSMX can be configured to enable or disable the use of wgMediators. In the scenarios for the SWS Challenge, the number of Web services is small and using the IRIS reasoner for abstract-level matching proved sufficient. The advantage lies in reducing the set-up time overhead by removing the need to define wgMediators in addition to ontologies, goals and Web services. Data mediation in WSMX is available internally during discovery through the implementation of the SSOA event-based communication model (see Section 4.6).

Instance-level matching in Glue (stage 2) applies the input data to the rules defined in the wgMediators. Where data mediation is necessary, it is achieved through an ooMediator modelled in WebML and executed via WebRatio. In WSMX, instance data is added to the knowledge base used by the reasoning engine (it already contains the goal and Web services descriptions). The matching algorithm is applied over this knowledge base. The conditions, used by WSMX to define whether or not a goal can be evaluated as resolved, are included in the goal descriptions. In Glue, these rules are defined in the wgMediator. On the one hand, the WSMX approach obviates the need for wgMediators for small sets of Web services. However, including these conditions in a wgMediator allows more flexibility in how the rules are expressed and evaluated e.g. the rules may not necessarily be in a formal language requiring

\textsuperscript{49}http://www.iris-reasoner.org/ (accessed 10 Mar 2011)
Extending the instance-level matching by fetching additional data from the Web service at runtime is the third stage. For WSMX, the data-fetch interface has a defined rule-based choreography, which can be used to efficiently invoke the Web service to fetch only data which is necessary for a match. This is derived from the variables defined in the Web service capability, and the rules defining the choreography of the data-fetch interface (see Section 5.1.2). The actual invocation is carried out by the WSMX Communication Manager in response to an event raised by the Discovery component. Glue identifies the Web service calls that are required to negotiate with the candidate Web service by fetching additional data, and then delegates the invocation of the Web service to the WebRatio engine. Once the data is received by WebRatio it is passed back to Glue where it is evaluated in the rules defined in the relevant wgMediator.

Selection in both SWE-ET and WSMX is regarded as a separate task to discovery. Both support ranking discovered Web services based on the specification of preferences as non-functional properties in the goal descriptions. For simple conditions such as lowest price and/or available in Europe the modelling language used by WSMX provides sufficient power for automated selection – including where the conditions are aggregated. SWE-ET, in contrast, makes no assumption on how selection is to take place and relies on expert human input if multiple matching Web services are found.

WSMX and DIANE

DIANE uses a customized formalism, DSD (described in Section 7.1.1) for describing service offers and requests. Both are defined in terms of sets of effects. For service offers, the values used to determine membership of a set can be either fixed or within a range of values. Variables can be used to mark where data is either required as input or provided as output. Such variables can be annotated as corresponding to Web service requests that do not have an effect in the real world so that DIANE can invoke the Web service at run-time to fetch the required value. As with WSMX, data is only fetched in this way as needed. Where service offers require the inclusion of more complex arithmetic formula or logical axioms, DIANE may have to resort to the use of an external service.

WSMX uses rule-based goal and Web service capabilities (defined using WSML-Rule) with a compatible reasoner (IRIS) to enact discovery. Axioms and arithmetic formula included in the capability description can be directly evaluated by the reasoner. The same language is used to define the choreography of the Web service data-fetch interface. This means that there is greater flexibility in the types of conditions, relations and axioms that can be defined directly in the goal and service descriptions used by WSMX. Also the Discovery component has full event-driven access to the Data mediation component which uses the same language and logical reasoning engine. A potential drawback is that such an
open-ended mechanism prevents reasoning tasks from being optimized in advance. In contrast, DIANE provides a very efficient matching algorithm based on synchronously comparing the graphs of service offers and requests. A trade-off is that the matching algorithm for DIANE depends on the request and offers having very similar ontological structures and data mediation is assumed to have already taken place – through the use of a lifting adapter.

The basic discovery implementation for WSMX allows preferences, which can influence selection, to be included as non-functional parameters. DIANE has a more sophisticated model for specifying preferences using fuzzy conditions for set memberships and allowing selection strategies to be incorporated as part of service offers. Overall, the primary advantage of DIANE is its optimization of the discovery algorithm itself. Balancing against this is less support for rich condition and axiom definitions in comparison to WSMX.

### 7.3.4 Comparison: Mediation

SWE-ET’s approach to mediation is described in Section 7.2.3. It relies on a mix of Semantic Web and Software Engineering techniques supported in a rich graphical user interface. The approach is to model the data models of service requesters and providers in WebML. The overall interaction between the two parties is modelled using Business Process Modelling Notation (BPMN) which is then translated to WebML. To overcome data mismatches, different types of WebML units can be deployed to (1) lift/lower between the SOAP messages required by the various Web services and (2) mediate between ontological terms. WebML supports WSMO $\text{ooMediators}$ as well as XSLT transformation units for this purpose. The completed WebML model of the interaction between the service requester and provider represents a SWE-ET mediator and can be executed via WebRatio. The changes to the SWS Challenge mediation scenario required modifications to the WebML data model and to the BPMN and subsequent WebML models. However, the rich user-interface support means that these modifications can be made quickly without the need to manually change any programming code. The advantages of this hybrid approach stem from the richness of the combined WebML graphical language with strong model-driven user-focused tooling. A corresponding potential drawback is that each scenario must be independently manually modelled as a process using BPMN. This only becomes relevant where large numbers of, potentially changeable, processes must be managed.

WSMX takes a different approach. In contrast to SWE-ET, it does not model individual scenarios as processes that can be executed by the framework. The goal-driven approach depends on (1) the rich ASM-based choreography interface definitions of service-requester goals and service-provider Web services and (2) the defined semantic mappings between ontologies, to carry out all actions to resolve the
given goal. Catering for the changes to the scenario involved changes to (1) the ontologies used, (2) the lifting/lowering adapters and (3) the choreography interface description of the service provider. Thus the changes were limited to updating the semantic definitions. Advantages of the WSMX approach are (1) service requester and service provider are strongly decoupled (2) the requirement for data mediation and/or process mediation (between goal and service choreography) can be automatically determined by WSMX at run-time, and (3) data and process mediation components are core to the framework making use of the WSML-Rule language and corresponding reasoner. Disadvantages are (1) the level of knowledge required for a modeller to create the necessary Web service, goal and ontology descriptions and (2) dependence on the power of the logical reasoning engine. For the former, tool support exists (e.g. WSMT\textsuperscript{50} but is not as rich as that provided by the combined SWE-ET solution. The full description of the WSMX solution to mediation is in Section 6.2.

DIANE solves the overall SWS Challenge mediation problem through modelling the interaction between service requester and provider as a WSBPEL process. This process is then modelled as a DSD service offer that can be matched to a service request corresponding to the RosettaNet PIP 3A4. Lifting/lowering adapters are provided to translate data between the RosettaNet data model and that expected by the Moon Web services. Data mediation is also possible. Additionally, the grounding definition of DSD service offer descriptions allows the inclusion of data mediation rule. The disadvantages of the DIANE approach to mediation is that, like SWE-ET, it each new scenario must be modelled separately as a WSBPEL process, coupling service requester and provider. For example, changes specified in the Challenge scenario required a remodelling of the WSBPEL process and lifting/lowering adapters. Unlike SWE-ET, DIANE does not have a rich user interface to integrate process modelling with Semantic Web service techniques. However, DIANE’s solution to the mediation problems proves sufficient to solve the problems and the remodelling effort was not excessive. The advantage is that DIANE uses arguably just enough semantics coupled with standard process modelling to keep the complexity of the solution to a minimum.

7.4 Summary

In this Chapter, I have described in detail the two most active participants, along with the WSMX, in the Semantic Web Services Challenge. Their architectures have been compared and contrasted with that proposed in SSOA model of Chapter 4. The respective approaches to mediation and discovery (including combined discovery and composition), which each deployed to solve the relevant problems, were reviewed. In each case, the fundamentals of each framework were identified, compared and contrasted. Each of the

\textsuperscript{50}http://wsmt.sourceforge.net (accessed 10 Mar 2011)
three approaches has strengths and weaknesses. All three are compatible with the conceptual model of
WSMO used as the data model for the SSOA reference architecture.

For DIANE, this is through a mapping to the DSD language. WSMX and SWE-ET are explicitly goal-
driven. In particular the Glue component of SWE-ET fully uses the different types of mediators defined
by WSMO and relies strongly on wgMediators for Semantic Web service discovery. WSMX and DIANE
treat composition as an extension to the goal-based solution for Semantic Web service discovery. They
use the rich formal descriptions of the Web service offers and the requests (or goals) to match services
that are required. In both cases the composition is only determined at run-time, albeit completely
based on how the service and goal descriptions are modelled. While DIANE and WSMX use different
implementations and algorithms, their fundamental basis is the matching of the state transition models
of services and requests, extended for composition to include relationships between multiple required
effects. DIANE’s model for prioritizing effects is more formal than that of WSMX. However, the rich
rule-based descriptions of the WSML-Rule language operating in tandem with the IRIS reasoner arguably
give WSMX an edge in terms of reasoning power and flexibility. Both DIANE and WSMX support the
invocation of Web services during run-time, based on declarative interfaces included in the semantic
description, to fetch contracting data that may be necessary for one of the goals in the composition to
be resolved. On the other hand, WebML take a software engineering approach to composition aligning
with standards such as BPMN and proving that the expressive power of their processing primitives, in
the large, covers the patterns laid out by van der Aalst in [van der Aalst et al., 2003]. SWE-ET is a
family of technologies, each focused on a specific area, that have been configured to cooperate together
to tackle the problems presented by the Semantic Web Services Challenge.

Service discovery happens as a distinct separate step to composition. The process defined for a
composition is intended to be built by a business analyst using the comprehensive and robust set of tools
provided within the SWE-ET framework. Where changes to scenarios happen, this defined composition
must be modified by the analyst to reflect the new situation. WSMX and DIANE aspire to a more
automated reaction to change. With the assumption that ontologically, mappings are available between
all ontologies required for a scenario, then only the goal need be modified and the framework will react.
This reflects the aspiration in Chapter 1 of applications that are constructed on the basis of what capability
is needed rather than on how that capability will be achieved.

WSMX has, at its core, algorithms to employ process and data mediation at run-time as necessary.
Handling heterogeneity on-the-fly is still the greatest barrier to the vision of virtual systems coming
together on the fly. The “magic” is that even though, as ever, data mappings have to be created before
either data or process mediation is possible, these mappings are created at the ontological level and
applied as needed. This is the heart of the process mediation algorithm in Section 5.3.1. DIANE does
not approach this directly, rather assuming that interfaces will be normalized in advance of execution so that mediation at run-time is not required. The SWE-ET approach is to engineer the solution at design time including the creating of any XSL instance-based mappings that may be required.
8. Related Work

The OWL-S Broker [Paolucci et al., 2003, 2004a], Internet Reasoning Service (IRS) [Motta et al., 2003], METEOR-S [Verma et al., 2005a] and WSMX (see Chapter 6) represent the initial set of frameworks for discovery, mediation and invocation of Semantic Web services based on the conceptual models of OWL-S, WSMO and SAWSDL. This Chapter describes and compares the architectures of these three frameworks. The criteria used for the comparison is based on the data, component, behavioural and communication models of the reference SSOA, described in Chapter 4. Each of the frameworks is evaluated against the functional and non-functional requirements for a Semantic Service Oriented Architecture developed in Section 3.2. Figure 8.1 reproduces the summary table of these requirements.

8.1 The Internet Reasoning Service (IRS)

[Motta et al., 2003] present IRS-II as a framework to support the publication, location, selection, composition and execution of software applications (denoted as services) based on their semantic descriptions.
Its motivation was to make possible run-time application creation through the automatic configuration of reusable knowledge components, available from distributed libraries on the Internet [Motta et al., 2003]. IRS-II uses the UPML problem-solving framework of [Fensel et al., 1999] to distinguish between classes of components for domain models, task models, problem solving methods (PSM) and bridges (to resolve heterogeneities between different model components). [Domingue et al., 2008] describes the original IRS (IRS-I) as an initial prototype supporting the creation of knowledge intensive systems structured according to the UPML framework.

[Cabral et al., 2006], and later, [Domingue et al., 2008] explain how IRS-III evolved to support the Semantic Web service conceptual model defined by WSMO. They provide a high-level description of the IRS-III design and an example of its implementation in an e-government scenario. While, in principle, the feature set of IRS-III remains unchanged from IRS-II, the WSMO model is adopted and more detail is provided on the design of specific elements for handling choreography and orchestration. Both the WSMO and IRS projects have a common research ancestor in the UPML framework facilitating the mapping from one conceptual model to the other. The concepts of task and PSM used in IRS-II map directly to the WSMO concepts of Goal and Web service respectively. Likewise the concept of bridge in IRS-II maps to that of Mediator in WSMO.

8.1.1 Global View

The IRS-III framework has three components: (1) IRS server, (2) IRS publisher and (3) IRS client. The IRS Server is a broker for Semantic Web Services. The IRS Publisher is responsible for generating semantic wrappers for Web services, Java, or LISP application code to make them available as IRS semantic services. The IRS Client is a goal-based API for interacting with the IRS Server. The relevant component for this thesis is the IRS Server. Figure 8.2 reproduces its architectural diagram. The architecture is related to the model for the SSOA reference architecture in the following sections.

8.1.2 Data Model

All internal data in IRS-III is represented in OCML [Motta, 1998] including the concepts from the conceptual model provided by WSMO. IRS-III provides specific modifications to the WSMO conceptual model to facilitate operations within the OCML Reasoner. These modifications are described in [Domingue et al., 2008] and include: (1) definition of meta-classes for the top level WSMO concepts of Web Service, goal and mediator; (2) allowing user definitions of goals, Web services and mediators as classes from the corresponding WSMO concepts (rather than as instances); (3) extension to the model for defining choreographies and orchestrations. In addition to OCML, the IRS Invoker can send and receive XML
Figure 8.2: The IRS-III Server [Domingue et al., 2008]

data types to Web Services.

8.1.3 Component Model

The SWS Library holds the OCML semantic descriptions of goals, service descriptions to solve them, mediators and ontologies representing domain knowledge models. This maps to the SSOA architecture Service Registry. The SSOA architecture does not specify if, or how, user-goal descriptions or domain model ontologies should be stored.

The OCML Reasoner represents the Logical Reasoning Engine of the SSOA model.

The Mediation Handler consists of the subcomponents of Goal Mediator, Process Mediator, and Data Mediator. The Goal Mediator searches for WSMO wgMediators [Roman et al., 2005] whose source component matches a received goal and whose Web service matches the capabilities required in the goal description. This corresponds to the responsibility of the Discovery and Selection components of the SSOA component model. The design of the IRS Goal Mediator allows for Goal refinement, as a discovered wgMediator may be theoretically linked to one (or a chain) of WSMO ggMediators, which enable a user’s goal to be mapped to a goal whose capabilities are known to be satisfied by a Web service in the IRS SWS Library. Similarly a wgMediator can include a reference to a WSMO ooMediator, providing the data mediation mappings (or implemented service), which can be resolved by the IRS
CHAPTER 8. RELATED WORK

Data Mediator component. The explicit use of wgMediators conforms to the WSMO conceptual model and is consistent with the declarative programming style of IRS.

There is a difference in design-approach here between IRS and the SSOA reference model. The responsibilities of the IRS Goal Mediator are equivalent to an aggregation of responsibilities of several components of the SSOA model. On the one hand, this reduces the number of components of the IRS framework, possibly enabling greater code efficiency. On the other hand it detracts from the scalability and comprehensibility of the architecture as the different functions of the components are not easily separated from each other.

Additionally, in practice\(^{51}\), IRS works best in a system where (1) all goals are predefined, (2) all candidate Web services are known in advance, and (3) goals and Web services are explicitly linked through wgMediators. If multiple wgMediators exist for a given goal, selection is based on the used of defined \textit{kappa} LISP-functions included in the mediator descriptions. In many of the examples for IRS-III, the first stage of Web service discovery effectively takes place during the modelling phase when goals are linked to Web services through wgMediators. At run-time, if multiple wgMediators have a given goal as their source and different Web services as their target, then all that remains is to select the service that best matches, using non-functional parameters in a LISP kappa function. Advantages of this declarative-programming approach include (1) speed and (2) the flexibility that new Web services can be linked to existing goals by the creation of new wgMediators. The disadvantage is that this places the burden on the system modeller of knowing which Web services match which goals. It’s important to note that run-time discovery can, in theory, be facilitated in IRS by using a wgMediator to link all goals to a Web service that implements a discovery algorithm.

The Process Mediator is responsible for keeping the state of the communication throughout the sequence of operations defined in either a service choreography or a service orchestration. It acts as an intermediary between either the Choreography Interpreter or the Orchestration Interpreter and the Invoker. It is also responsible for locating WSMO ggMediators, in the context of executing an orchestration, to link the sub-goals defined to represent individual steps in that orchestration. The mediation of mismatches in the process models of two cooperating choreographies is common with the feature of the Process Mediation engine in the SSOA model. However, the IRS Process Mediator differs from that of the SSOA model in that (1) the cooperating choreographies are typically two sub-goals in an orchestration, rather than the choreographies of a goal and matching Web service; (2) it has responsibility for managing choreography and/or orchestration state, as well as driving the Invoker element. In the reference SSOA, the latter function is the responsibility of the Choreography component. As with

\(^{51}\)I co-authored and co-presented five joint Semantic Web Service tutorials using a combination of WSMO, WSMX and IRS-III, including a hands-on session using IRS-III.
the comments for the Goal Mediator, the difference between IRS and the reference architecture comes
down to design and implementation choices for IRS-III as the overarching principle of handling process
mismatches and coordinating interface interactions through declarative choreographies are common to
both it and the SSOA model.

The **Data Mediator** is responsible for executing mapping rules between ontologies as declared
within WSMO ooMediator descriptions. The mappings can either be executed directly (where the rules
are declared using OCML) or indirectly (where the ooMediator points to a Web service which implements
the mappings). This is exactly the same responsibility as the Data Mediation engine of the SSOA model.

The **Choreography Handler** is responsible for accepting declarations of the choreography of a Web
service interface and initiating the messages which should be sent to that Web service, based on the
execution of the rules defined by the choreography, using the methodology of Abstract State Machines
(ASM). The Choreography Handler partially maps to the SSOA Choreography component. In both cases,
the components initiate messages which are sent to either the Web service or Goal owner. However,
choreography state is maintained in the SSOA Choreography component, while in IRS-III, state is
managed by the Process Mediator. This is a design choice which is not significant operationally, but
which results in a lack of clarity in the responsibilities of individual components.

The **Orchestration Handler** has similar responsibility as the Choreography Handler but, in this case,
handles Abstract-State-Machine based declarations of Web service orchestrations. IRS-III extends the
WSMO model for orchestration considerably to make the implementation of orchestration possible. The
extension includes the definition of a set of control-flow primitives which are implemented as part of IRS-
III. Subgoals making up an orchestration are linked using WSMO **ggMediators** (goal-to-goal mediators)
which may in turn use ooMediators (ontology-to-ontology mediators) to handle data mismatches.

In IRS-III, goals are linked to Web services using wgMediators during domain analysis. A Web
service that has a defined orchestration is linked to a suitable goal. The input and output roles (named
data types) defined for the goal are inherited by the Web service. At run-time, the implementation of
IRS ensures that data provided with the goal, corresponding to input roles, is made available to the
orchestration. Further, IRS-III is designed so that when a goal is matched to a Web service with an
orchestration, (1) the data provided with the goal, corresponding to input roles, is made available to the
orchestration, (2) the Orchestration Handler takes control of the execution of the orchestration, and (3)
the output data of the orchestration (if any) is bound to the output role of the Web service, which in
turn is bound to the output role of the original goal.

This design for orchestration has been demonstrated as part of a use-case for Essex County Council in
the context of emergency services response during severe weather episodes [Gugliotta et al., 2006]. While it proves the usefulness of defining Web service orchestrations, it does not solve the problem of linking the choreography (external process model) and orchestration (internal process model) in a declarative way. The design depends on the specific LISP and OCML implementation of the IRS-III framework itself.

**The Invoker** is responsible for generating the required Web service messages in SOAP format, triggered by the execution of rules in a choreography or orchestration execution, which can be sent to the appropriate service-endpoint URIs.

The aspect of interacting with the implemented Web services is common between IRS-III and the SSOA model. In the SSOA model, the lifting and lowering translation of messages between XML and the semantic representation language being used is carried out by a dedicated component. In IRS-III, it is carried out through the extension of the WSMO choreography model to include explicit lift and lower primitives, and their corresponding implementation in the IRS framework. The essential difference is that the SSOA model is more loosely coupled in that it allows any implementation of lifting and lowering to be used, as long as it complies with the defined interface.

### 8.1.4 Behavioural Model

Capability-based service invocation is the primary behaviour offered by IRS-III. A client provides a goal to IRS which then identifies matching Web services, selects the service that best matches the non-functional requirements, and uses the Web service choreography description to exchange messages with the Web service. Mediators are used to handle heterogeneity at the data and process levels. If the selected Web service has an orchestration, the Orchestration Handler is used to coordinate its execution.

Capability-based service invocation aligns with the combined goal-discovery and invocation behaviour defined in Section 4.5. A second IRS-III behaviour is to publish stand-alone code (Java and LISP), or existing WSDL-described Web services as Semantic Web services in its SWS Library. The publishing behaviour, which is not specified as part of the SSOA architecture, provides a convenient means to create Semantic Web service descriptions from existing artifacts for use in IRS-III. A drawback is that the resultant descriptions are in OCML tailored for the IRS implementation.

In Section 4.5 two additional behaviours are defined for the SSOA: (1) service discovery and (2) receiving data. These are not available in IRS-III. The service discovery behaviour allows for standalone discovery of Web services matching a given goal. The receiving data behaviour allows for asynchronous invocation of Web services where the thread\(^{52}\) on which the Web service is invoked does not block

\(^{52}\)A thread is a unit of processing of an operating system.
waiting for the response message. Instead, the Web service uses the receive data behaviour of the SSOA framework to send back the response data when it is available.

8.1.5 Communication Model

In normal operation, the components of IRS communicate with each other through direct LISP calls and the exchange of data represented in OCML. It is possible for the components themselves to be published as Semantic Web services to the IRS SWS Library so that the operational semantics of the IRS framework could be defined in terms of the following orchestration of sub-goals (not strictly in order) e.g. (1) discover and select a matching service, (2) choreograph the service message invocation, (3) mediate data, (4) mediate process, (5) orchestrate the sub-goal defined in a Web service orchestration description. Unlike the model for SSOA, communication between the components is not event-driven which reduces the opportunity for scalability. On the other hand, if the IRS is configured where all components themselves are published as Semantic Web services, then the flexibility of using a goal-driven invocation model can be achieved, albeit with some further customization of the IRS.

8.1.6 Summary

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Architectural Principles</th>
<th>IRS-III</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional Requirements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capability-driven discovery</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Service selection</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Dynamic data mediation</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Choreography interpretation and execution</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Dynamic process mediation</td>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>Logical reasoning support</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Compatible with WSDL Web services</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td><strong>Non-functional Requirements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reusable across problems and domains</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Loosely coupled with standardised interfaces</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Simple programmatic interface</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Scalable &amp; extensible</td>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>Goal oriented</td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.3: SSOA Requirements vs IRS-III

IRS-III provides a framework for capability-driven service invocation based foundational research on knowledge modelling and the adaptation of the original IRS problem-solving platform to use an
extended form of the WSMO conceptual model for Semantic Web services. It is interoperable with WSMX through a common agreed application program interface (API). It has been successfully applied in a real-world use-case scenario (emergency planning) and demonstrated in more than ten Semantic Web service tutorials at various international conferences. On the whole, IRS-III is compatible with the model for an SSOA reference architecture presented in this thesis. There is a consistent focus on maintaining the property of being an open declarative system. All elements—data, processing, and connector—are declarative structures, and are available for inspection and customization. Service requester and provider are clearly decoupled. The execution of the system is based on formal semantic descriptions and powered by a logical reasoning engine for OCML. Figure 8.3 summarizes how IRS-III satisfies the requirements for an SSOA described in Section 3.2.

All functional requirements are met. Components are not separated as they are in the SSOA model. For example, discovery and selection are part of the same component which has possible impacts for scalability and extensibility. The approach taken by IRS is to declaratively model all Web services, goals and mediators required for a given scenario during the problem analysis. Also, at this time goals are linked to specific Web service descriptions through the use of specified mediators. This means that all candidate Web services for capability-based invocation must be known in advance and their descriptions linked to existing goals via existing mediators. While this may limit the non-functional aspects of IRS, it is a design choice and does not detract from the fundamentals of the approach. Likewise the choice of OCML and LISP as implementation languages mean a specific skillset is necessary for semantic modellers wishing to work with IRS-III.

8.2 DAML-S Virtual Machine & OWL-S Broker

8.2.1 Global View

The DAML-S Virtual Machine (DAML-S VM) [Paolucci et al., 2003] is a software component which can be embedded into an application which uses the DAML-S Process Model to control the interaction between Web service process models. This first version, of what evolved into the OWL-S Broker [Paolucci et al., 2004a], provided a general purpose Web service client for the invocation of Web services based on the process model of their respective DAML-S descriptions. The focus was on the mapping of rules used by the DAML-S VM to the operational semantics of the DAML-S process model. There are two principle elements to the DAML-S VM (illustrated in Figure 8.4): the DAML-S Processor and the DAML-S Web Service Invoker. The former, supported by a DAML logical reasoning engine, drives the interaction between a client and a Web service having a DAML-S description. The latter element is
Figure 8.4: Architecture of the DAML-S VM [Paolucci et al., 2003]

responsible for invoking the Web services based on their WSDL description and handling any required data transformation using XSLT.

Figure 8.5: Architecture of the OWL-S Broker [Paolucci et al., 2004a]

The OWL-S Broker [Paolucci et al., 2004a] is a broker-based architecture for Semantic Web services based on the conceptual model defined by OWL-S [Martin et al., 2007]. It is a successor to DAML-VM,
targeting the requirements of mediation, discovery, selection, coordination and invocation of Semantic Web services described using the OWL-S ontology. When a service requester uses the OWL-S Broker to discover and interact with a Web service, the broker takes on the role of a proxy for the service provider. Much of the focus of the OWL-S Broker is toward solving what [Paolucci et al., 2004a] term the Broker’s Paradox. This problem arises as the service requester expects the broker to provide a public declaration of the desired Web service’s process model so that all required information can be provided by the requester as needed. However this process-model description is not available until the service has been discovered. Instead the requester must communicate with the Broker itself through its OWL-S process model. OWL-S has no mechanism to allow the process model of a service to be changed at runtime (from the generic Broker process model to the specific process model of the discovered service). A solution to this problem is provided by the OWL-S broker by extending the OWL-S process model with a new exec statement to allow a service’s process model to be changed dynamically. This differs from the approach to handling process heterogeneity of the SSOA model as the latter allows for mismatching public process descriptions of Semantic Web services to be resolved using process mediation. [Paolucci et al., 2004a] identify the following specific reasoning tasks for the OWL-S Broker.

1. Interpretation of capability advertisements
2. Interpretation of requester queries
3. Finding the best provider for the requester query
4. Invocation of the selected provider on behalf of the requester
5. Returning the query results to the provider.

There are two significant modelling difference between OWL-S and WSMO that are reflected in the framework of the OWL-S broker in comparison to both WSMX and IRS (as implementations of the SSOA model). The first is that goals (as the perspective of the service requester) are not explicitly modelled in OWL-S. The second is that mediators are not explicitly modelled. OWL-S takes the approach that both goals and mediators can be modelled using the OWL-S concepts of service profile, process model, and grounding and that further explicit concepts are unnecessary [Paolucci et al., 2004b]. In the absence of explicitly modelled goals, the OWL-S broker expects requests for services to be based on queries specified in OWL-QL [Fikes et al., 2005]. These queries are translated and then matched to service capabilities presented as service profiles. The potential drawback is subtle – a service requester must specify a query for the Web service that they wish to invoke rather than specifying, in declarative statements, the capability that they are looking for. More details on the architectural elements of the OWL-S Broker are provided in the following sections.
8.2.2 Data Model

All Web services descriptions used by the OWL-S Broker must be described using concepts from the three ontologies that define the OWL-S conceptual model: Profile, Process Model and Grounding. (See Section 4.3.2 for a listing of the primary models in this area and Appendix A for further details.) Domain ontologies, defining the semantics of terms used in the service descriptions, are described using the Web Ontology Language, OWL. All data within the OWL-S broker has well-defined semantics that can be reasoned over by the Query Processor component. As with IRS and WSMX, the OWL-S broker communicates with WSDL-described web services using XML – syntactically translated (lifted/lowered) from the corresponding OWL ontology fragment.

8.2.3 Component Model

The Discovery Engine is responsible for discovery, mediation and selection. It accepts an OWL-QL query from a service requester as input, and returns the OWL-S service description of a Web service that matches this query, if any can be found. Service discovery requires the initial step of translating the OWL-QL query into an OWL-S Service Profile with the appropriate required service inputs and outputs that (1) reflect the semantic content of the query and (2) reflect the requirements of the generated service request [Paolucci et al., 2004a]. Next the broker matches the capability description generated from the requester query with capability descriptions of service providers in the Advertisement Database (equivalent responsibility to the SSOA Service Registry). The final step of the discovery phase is selection where, if multiple matching services are found, the most appropriate is determined based on the level of similarity of the match and possibly other information from the candidate Service Profiles. [Paolucci et al., 2002] and [Li and Horrocks, 2003] provide detailed descriptions of two independent alternatives for service-matching based on the OWL-S conceptual model which are possible designs for this element.

In comparison with the SSOA component model, the Discovery Engine of the OWL-S broker has an aggregated responsibility for discovery and selection. Mediation during discovery is limited to pruning the generated OWL-S Service Profile of the discovered matching Web service so that data already provided by the service requester is not requested again. A form of implicit data mediation is possible, during discovery and selection, using the inferencing capability of the Query Processor component to determine relationships between ontologically-defined terms.

The OWL-S Processor is responsible for coordinating the interaction between the OWL-S Broker and individual Web services based on the specification of their Service Process Models and Groundings. It inherits this functionality from the earlier work on the DAML-S VM. The Process Model can specify a

single atomic process that grounds to a single WSDL service operation, or it can specify a composition of atomic processes with control constructs such as sequence, split, split-join, if-then-else, or choice. There is no restriction that all atomic processes in a composition must ground to WSDL operations on the same Web service. When all atomic processes ground to the same service, this could be considered as a form of implicit service choreography. Where the atomic processes ground to operations on different services, this could be considered as similar to service orchestration.

The OWL-S Processor uses the Query Processor to derive inferences from the information (1) provided to it by a service requester, and (2) received from a service provider. [Paolucci et al., 2003] provides an example illustrating that these inferences can form the basis of (1) data mapping between terms in different ontologies, (2) a mapping following a parent-child relationship between terms, or (3) a form of process mediation – determining that specific information required by a service Process model has not been provided and triggering the request to a service requester for it.

The Service Grounding specifies how operations in the Service Process Model map to specific WSDL operations. The execution of these mappings is driven by the OWL-S Processor using the Query Processor to interpret and execute the process model ruleset. Additionally XSLT mappings are generated to lift and lower data from OWL to the XML expected at the WSDL service interface.

As with the OWL-S Discovery Engine, the OWL-S Processor presents an aggregation of several distinct roles identified in the SSOA component model including aspects of the Choreography engine, Data Mediation, Communication Manager, Lifting and Lowering Adapters. This is understandable as (1) the OWL-S Broker acts as a reference implementation for OWL-S and (2) it acts as a prototype to show how Semantic Web service discovery combined with invocation is possible.

The OWL-S Query Processor operates at the heart of the OWL-S Broker implementing the operational semantics of the OWL-S Process Model. It is used to execute the Process Model descriptions of individual services and to provide a logical inferencing service to the the different phases of the Matching Engine. It maps to the SSOA Logical Reasoning component.

Knowledge Base is used by the Query Processor to store information as needed during the execution of a service requester query. It is responsible for holding the OWL-S descriptions of Web services, the OWL-S descriptions of requester queries (after translation from OWL-DL) and the OWL definitions of any domain models required by the OWL-S Broker to carry out its tasks. It maps to the SSOA Ontology Store.

The Web Service Invocation module uses off-the-shelf open-source implementations of Web service engines to make the actual Web service calls and handle the response. The Invocation module is driven
by the OWL-S Processor which grounds OWL-S operations to corresponding WSDL operations and provides the Invocation module with the XML expected at the relevant Web service endpoints.

**Implicit Data and Process Mediation.** Data mediation does not have its own distinct OWL-S Broker component element but is an important aspect of functionality for both the Discovery Engine and the OWL-S Processor. In both cases the inferencing capability of the Query Processor is used to enable mediation between ontologically defined terms. Additionally, [Paolucci et al., 2004a] suggest that the OWL-S Processor is capable of some of the process mediation tasks described in Section 4.4.4. While no detail is provided on how this is achieved, it indicates the importance of this type of mediation in the context of a Semantic Web service framework.

### 8.2.4 Behaviour Model

![Figure 8.6: Behaviour Model for the OWL-S Broker [Paolucci et al., 2004a]](image)

The OWL-S Broker offers the behaviour of query-driven Semantic Web service discovery and invocation. Although not explicitly stated, it also offers a discovery-only behaviour (inherited from the DAML-S VM). As with IRS-III, the receive-data behaviour, which provides support for asynchronous message interaction between the service requester and provider is not present. One consequence is a potential for reduced scalability as it is assumed that the service requester process thread blocks waiting for the broker to discover and get a response from a Web service which can answer the requester query. A second consequence is that all data, required for the discovered Web service to be invoked, must be included along with the service requester’s request upfront.
8.2.5 Communication Model

This mechanism for communicating between the OWL-S Broker components is not explicitly defined. It is likely that each component exposes their interface as a programming language API and that communication between components is based on direct method-invocation from a main control thread of the OWL-S Broker itself.

8.2.6 Summary

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Architectural Principles</th>
<th>OWL-S Broker</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional Requirements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capability-driven discovery</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Service selection</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Dynamic data mediation</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Choreography interpretation and execution</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Dynamic process mediation</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td>Logical reasoning support</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Compatible with WSDL Web services</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Non-functional Requirements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reusable across problems and domains</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Loosely coupled with standardised interfaces</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Simple programmatic interface</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Scalable &amp; extensible</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Goal oriented</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 8.7: SSOA Requirements vs OWL-S Broker

The OWL-S Broker provides a purpose-built framework for the discovery, mediation and invocation of Semantic Web services described using the OWL-S ontology. It extends earlier work on the DAML-S Virtual Machine, which provided an infrastructure for the invocation of WSDL-based Web services that had been annotated using OWL-S. The primary behaviour of the OWL-S Broker is to facilitate the interaction between two autonomous and independent parties – service requesters and service providers. A constraint of using the OWL-S broker is that an instance of it must be present as part of the client service requester application as well as part of the OWL-S Broker server deployment. Its not clear if OWL-S Brokers can be federated.

The OWL-S broker consists primarily of five components: Discovery Engine, Knowledge Base, OWL-S Processor, Query Processor, and the Web Service Invocation Engine. The preceding sections examined how the roles played by these elements can be mapped to elements in the SSOA architecture. In many
cases, roles are aggregated, possibly due to the prototype-nature of the OWL-S Broker as (1) a framework to prove the feasibility of brokering Web services based on OWL-S descriptions and (2) a platform for further research.

There are two significant differences in the design of the OWL-S Broker to the WSMO-based approaches of IRS and WSMX. Both stem from the respective choice of Semantic Web service ontologies (OWL-S vs. WSMO). The first is that queries must be generated in an external language - OWL-QL - rather than specified as a goal in the same language as used by the elements of the framework. This adds to the complexity of the role required of the service requester as (1) the query is specified in terms of the OWL-S Web service description that is required, rather than simply what a service requester wants to achieve; (2) an additional language to represent queries must be used; and (3) all data required for the invocation of the discovered service must be provided along with the query. The second difference is that mediation is implicit and based on the use of run-time inference by the Query Processor. This is in contrast to the SSOA model which allows for explicit declarative mediation which can be reused and managed externally to the execution environment.

Finally, in terms of the non-functional requirements, a detailed description of the subcomponents of the design is not publicly available and so the openness and extensibility properties of the architecture are difficult to judge. Architecturally, there is also a constraint that the OWL-S Broker is required at both the client and in the OWL-S broker. Its presence at the client is required as the OWL-S Broker itself is presented as Semantic Web service with a corresponding OWL-S description.

### 8.3 METEOR-S

METEOR-S [Verma et al., 2005a] is a project focusing on how the creation of business processes using Web services can be made more scalable and dynamic through the application of Semantic Web technology. What contrasts the METEOR-S approach from other Semantic Web service frameworks, is its intentionally bottom-up approach, focusing on adding semantics to existing Web service technology and specifications, particularly UDDI [Clement et al., 2004] and WSDL, using extension points in those technologies. This contrasts to the OWL-S and WSMO approaches which develop Semantic Web service ontologies that are then mapped to WSDL operations. METEOR-S was initially developed at the LSDIS Lab at the University of Georgia 54. Its research is carried on by the Knoesis project at the Wright State University, Ohio 55.

54http://lsdis.cs.uga.edu
55http://knoesis.wright.edu/ (accessed 10 Mar 2011)
8.3.1 Global Overview

Overall METEOR-S uses a collection of techniques from Semantic Web and workflow technology for service discovery and publication, and to enhance current Web process composition techniques by using Semantic Process Templates to capture the semantic requirements of the process [Sivashanmugam et al., 2005], enabling the creation of flexible and scalable Web service processes. These processes are constructed as templates where each step in the template corresponds to a process activity. There are different ways in which each activity can be bound to a concrete Web service. Once a process template has been constructed, through interaction with a human expert, the process can be transformed into an executable form represented in WSBPEL [Alves et al., 2006], and executable by any compliant WSBPEL execution engine.

Figure 8.8: Architecture of METEOR-S [Verma et al., 2005a]

An overview of the architecture of METEOR-S is shown in Figure 8.8. There are three sub-areas: (1) service annotation, (2) service publication and discovery, and (3) a composition and service execution framework. The first of these – service annotation – is a bottom-up approach to the semantic annotation of WSDL service descriptions. This started as the WSDL-S joint approach with IBM [Akkiraju et al., 2005], later evolving into the W3C recommendation, SAWSDL [Kopecký et al., 2007b] 56. The problem of semi-automatic mapping of WSDL elements to domain ontology concepts is described in [Patil

56http://www.w3.org/TR/sawsdl/ (accessed 10 Mar 2011)
et al., 2004]. It describes a graph-based algorithm for the comparison of a WSDL document to one or more domain ontologies to determine potential mappings. In the context of this thesis, the latter two components are most relevant and are described briefly in the following sections.

8.3.2 Data Model

Web services in the METEOR-S framework are described using the SAWSDL conceptual model (originally using WSDL-S – see Section 4.3.2). SAWSDL is a W3C Recommendation specifying extensions to WSDL to allow the mapping of WSDL elements to concepts defined in an ontology. It is agnostic to both the model and language used by the ontology (e.g. RDF, OWL and WSMO can be supported), and requires only that concepts can be referenced by URI. Domain ontologies in METEOR-S provide the formal definition of the semantics of the concepts used in the Web service descriptions. There is an implicit assumption that any domain ontologies used in METEOR-S are represented in XML.

Web service requests are formed by the creation of service templates which specify the inputs and outputs of the desired service from one or more of the domain specific ontologies. Templates may also include service preconditions and effects. The template is used as the basis for a query against semantically enhanced Web service descriptions published in one or more UDDI registries using the METEOR-S service annotation and publication component.

METEOR-S supports the semi-automatic creation of semantically-enriched Web service processes. These processes are modelled and represented as executable service-based business processes, using WS-BPEL, which can be executed by any compatible engine. The messages exchanged between such a WSBPEL process and individual Web services may include terms from ontologies defined using conceptual models described by OWL, RDFS or WSMO, but must be represented using XML.

8.3.3 Component Model

Semantic Publishing & Discovery. The METEOR-S Web Services Discovery Infrastructure (MWSDI) [Verma et al., 2005b] provides a layered architecture over UDDI [Clement et al., 2004], which enables (1) the publication of Semantic Web service descriptions, and (2) a scalable P2P-based infrastructure for the discovery of service descriptions across a potentially large collection of UDDI registries. It facilitates service providers in publishing semantic service descriptions through a graph-matching algorithm which matches elements in WSDL descriptions with concepts from one or more domain ontologies. Both semantic publication and discovery are based on the creation of semantic templates that contain semantic descriptions of the inputs and outputs of the data that is expected at a service interface. UDDI metadata structures (tModels & CategoryBags) are used to store semantic descriptions of the other aspects of Web
services. The model is agnostic to the Semantic Web service conceptual model used.

MWSDI has a layered architecture that is required at each node in a MWSDI federation of UDDI registries. The top-most layer is called the Operator Services Layer and is responsible for maintaining the services for publishing and discovering Web services in that registry. [Sivashanmugam et al., 2005] describe that the discovery process is based on template matching followed by ranking and selection. These operations are based on (1) semantic annotations of service inputs and outputs and (2) service functional semantics, and (3) service preconditions and effect, but specific details on the design for these elements is not provided. In terms of the SSOA architecture, the MWSDI component represents an aggregation of the discovery, selection, service registry and goal owners elements. Template-based discovery differs from goal-oriented discovery in that the definition of a template is focused on defining the types and semantics of inputs and outputs a desired Web service should have. This is in contrast to the SSOA approach which allows a goal with the desired capability to be provided and (semi-)automatically resolved by the infrastructure.

**Composition and Execution Environment (MWSCF).** The component is responsible for providing a user-driven framework for the composition of Web processes using semantically annotated Web services. MWSCF provides an application which enables the semi-automatic composition of Semantic Web processes. *Semantic Process Templates* (SPT) are used to store the semantic requirements of processes. An SPT is a collection of activities, and possibly intermediate calculations, that are connected by control-flow constructs. Each activity in a process can be specified using one of (1) a Web service implementation, (2) a Web service interface (WSDL), or (3) semantic activity template. Semantic activity templates are used as input for service discovery to the MWSDI, which provides the necessary Web service discovery infrastructure.

MWSCF uses the Semantic Publishing & Discovery module provided by MWSDI. Additional sub-components to MWSCF are shown in Figure 8.8: Invoker, Process Manager, Constraint Analyser, and Optimizer. The Process Manager is exposed to users through the Semantic Web Process Designer (SWPD), a graphical user interface. The SWPD is used to design or modify Semantic Process Templates and pass these to the Process Manager which, through its subcomponents, generates an executable process description in WSBPEL.

The Process Manager subcomponents consist of a Constraint Analyser and a Constraint Optimizer described in [Aggarwal et al., 2004]. All criteria that affect the selection of the services are treated as constraints. They are represented as ontological rules and stored in UDDI. An example of a constraint is that a particular supplier is a favoured supplier of electronic parts for a manufacturing company. A mechanism for storing the cost of specific constraints is also proposed. During the design of the
Web service process using METEOR-S, candidate services, their constraints, and costs are retrieved from the respective UDDI registries. The constraints and costs are provided as input to the Constraint Optimizer subcomponent (implemented as an integer linear programming solver, LINDO\textsuperscript{57}). This outputs the optimized set of Web services that can fulfill the required process. The Invoker is responsible for making the Web service invocations and is driven by the execution of the WSBPEL process model. It is implemented using the open-source Apache Axis framework\textsuperscript{58}.

8.3.4 Behavioural Model

The behaviours offered by METEOR-S are (1) discovery and publishing of Semantic Web services (using MWSDI) and (2) the creation of executable Semantic Web service compositions (using MWSCF). Both behaviours are provided via graphical user interfaces as tools to assist Semantic Web service system designers.

8.3.5 Communication Model

Components relating to the publishing and discovery of services in MWSDI are connected through a peer-to-peer (P2P) network as outlined in [Verma et al., 2005b]. Four types of peer (Gateway, Operator, Auxiliary and Client) are identified along with high-level protocols for their interaction. Each type of peer has a different level of responsibility regarding the provision or the updating of the possibly multiple UDDI service registries used by the MWSDI. Client queries are created as semantic templates by client peers using the MWSDI graphical user interface. There is no explicit description of data transformation or mediation elements during discovery or service composition. The invocation of the Web services is delegated to the Apache Axis SOAP engine. Likewise the execution of the WSBPEL process description created using the MWSCF is delegated to a suitable execution engine.

METEOR-S differs to IRS, WSMX and OWL-S Broker in that it provides a scalable, user-driven design-time tool for the publication, discovery and composition of Web services. The graphical user interface provides a dashboard that enables a human expert to drive the publication, discovery and composition of Web services. The semantics of the run-time connections and interaction between the Web services is defined by the selected WSBPEL engine. This is consistent with the aim of METEOR-S to provide a framework to enable bottom-up semantic annotation of Web services to maximise the re-use of the existing Web service technology i.e WSDL, UDDI and WSBPEL.

\textsuperscript{57}http://www.lindo.com/ (accessed 10 Mar 2011)
\textsuperscript{58}http://ws.apache.org/axis/ (accessed 10 Mar 2011)
CHAPTER 8. RELATED WORK

8.3.6 Summary

METEOR-S is a framework designed to tackle the problems of Web service discovery and composition through the combination of Semantic Web technology with existing Web services specifications, particularly WSDL and UDDI. The W3C SAWSDL recommendation is a direct consequence of this research path. METEOR-S reflects the opinion that acceptance of Semantic Web technologies is best achieved by incorporating semantic annotations in existing Web service specifications as extensions. This is in contrast to the OWL-S and WSMO approach which each define conceptual models for Web services and then map elements in those models to existing WSDL descriptions.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Architectural Principles</th>
<th>METEOR-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capability-driven discovery</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Service selection</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Dynamic data mediation</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Choreography interpretation and execution</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Dynamic process mediation</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Logical reasoning support</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Compatible with WSDL Web services</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Non-functional Requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reusable across problems and domains</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Loosely coupled with standardised interfaces</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Simple programmatic interface</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Scalable &amp; extensible</td>
<td></td>
<td>+/-</td>
</tr>
<tr>
<td>Goal oriented</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 8.9: SSOA Requirements vs METEOR-S

Like the other Semantic Web service frameworks described earlier, METEOR-S use semantics to improve the accuracy of Web service discovery, composition and execution. The strategy is to incrementally include semantic annotations as needed while supporting Web services that have no semantic descriptions. It is up to the human user creating Web service processes to decide which services to include or not. METEOR-S uses semantic descriptions, if available, to make suggestions. Architecturally, the fundamental differences between METEOR-S and IRS, WSMX and the OWL-S broker is that the principal architectural style of the latter systems is broker-based while that of METEOR-S is expert-system based. This is reflected in the property that the requirements behind the creation of a METEOR-S process are defined interactively by a human user using a GUI. The other systems allow the specification of a goal or required capability that is submitted to the respective systems for resolution.
Figure 8.9 summarizes how METEOR-S evaluates against the SSOA requirements. Semantic Web service discovery is possible using a *Semantic Activity Template (SAT)* which is similar in intent to the concept of a WSMO goal. A service requester can define an SAT having semantic definitions of the required inputs and outputs as well as semantic definitions of the activity's preconditions and effects. There is no explicit support for automating data or process mediation. The former is solved using XSLT transformations while the latter is at the discretion of the human process designer when building a WSBPEL process definition, supported by METEOR-S. *Semantic Process Templates (SPT)* allow SATs to be composed into a process that can be resolved by the MWSCF component at run-time. This matches the intent of the Orchestration component of the SSOA component model. In terms of non-functional properties, the main constraints are due to the fact that METEOR-S is intended as an aid to Semantic Web service process designers. There is a focus on scalability and extensibility in the P2P design of the MWSDI component for service publishing and discovery. This is based on the assumption that these operations require access to federations of UDDI registries. With respect to automated Semantic Web service discovery, mediation, and invocation, however, METEOR-S is less scalable as the system requires the coordination of these tasks to be carried out by a designer. The execution of selection (constraint analysis) and process execution are dedicated to software which is external to METEOR-S itself.
Part V

Conclusion
9. Concluding Remarks

The research for this thesis was motivated by a gap in the body of knowledge for Semantic Web services. The gap existed between (1) conceptual models and languages for Semantic Web services on the one hand (e.g. WSMF [Fensel and Bussler, 2002], DAML-S/OWL-S & SWSF/SWSL (see Appendix A)), and (2) prototype implementations using those models and languages, primarily for service discovery, on the other (e.g. IRS-III (Section 8.1), the DAML-S Virtual Machine (Section 8.2), and METEOR-S (Section 8.3)). The absence of a reference software architecture model, as distinct from a conceptual model, for Semantic Web services is significant.

In Section 2.2, I described how software architecture provides an abstraction of a system that allows specific characteristics of that system to be easily identified, and allows the system to be communicated clearly. Without a reference architecture for Semantic Web service systems, it is difficult to (1) understand the relationship of Semantic Web services with respect to the timeline of technology that preceded them, (2) determine the core set of elements common to this type of architecture, (3) compare the architectures for Semantic Web services emerging from various research groups, and (4) experimentally and qualitatively evaluate different Semantic Web service framework implementations against independently published problems.

The goal of this thesis was to fill this gap in the body of knowledge for Semantic Web services by proposing a reference Semantic Service Oriented Architecture, evaluating this architecture through a concrete design & implementation and qualitative comparison, and contribute to the body of knowledge by providing a point of reference by which existing and future research, in this area, can be compared and contrasted.

The research was carried out using a combination of top-down and bottom-up approaches. The top-down approach is demonstrated in Chapters 2, 3 and 4. An analysis of the timeline of distributed and agent systems technologies that precede and directly lead to Semantic Web services is presented in Chapter 2. The motivation and requirements for a Semantic Service Oriented Architecture are laid out in Chapter 3. Chapter 4 proposed a set of models to describe the reference architecture derived from requirements analysis and software architecture modelling. The bottom-up approach is illustrated
CHAPTER 9. CONCLUDING REMARKS

in Chapters 5 and 6. Chapter 5 describes novel algorithms for Semantic Web service discovery & composition, and process mediation. A reference implementation of the architecture, which has been experimentally and qualitatively peer-evaluated against a set of integration problems in the context of the Semantic Web Services Challenge, is presented in Chapter 6. The research is also evaluated from both top-down and bottom-up perspectives. The former is carried out through a qualitative comparison of the reference architecture against three contemporary Semantic Web service architectures in Chapter 8. The latter is shown through an experimental evaluation of the reference implementation of SSOA in Chapter 6 and comparative evaluation with peer systems in the context of the Semantic Web Services Challenge in Chapter 7.

The following sections list and revisit the contributions made in this thesis and highlight open research questions suitable for possible future work.

9.1 Summary of Contributions

There are three categories of contributions in this thesis. These are described in the following three subsections. Section 9.1.1 describes the contributions with respect to the development and publication of a SSOA component model and reference architecture. Section 9.1.2 contains the contributions made in the definition of algorithms for specific problems regarding Semantic Web service discovery, mediation and composition. Finally, in Section 9.1.3, I describe the contribution related to the qualitative comparison of the SSOA architecture to peer Semantic Web service frameworks.

9.1.1 SSOA Component Model & Reference Architecture

Although there are many fixed points in the body of knowledge for Semantic Web services, no reference architecture for Semantic Web services exists which (1) explicitly sets out a minimal set of essential architectural elements, (2) validates the architecture against existing models and frameworks, and (3) verifies the architecture’s feasibility against both a requirements matrix and independently posed problems. Other Semantic Web services architectures exist but they do not meet each of these criteria (e.g. SWSA – see Section 2.1.3; DIANE and SWE-ET – see Chapter 7; and IRS, OWL-S VM, & METEOR-S – see Chapter 8).

This thesis contributes to filling this gap in the knowledge of Semantic Web services by presenting a SSOA reference architecture both from (1) a modelling perspective and (2) a design & implementation perspective, to enable the architecture’s feasibility to be evaluated against independently-set problem scenarios. Section 2.1 positions Semantic Web service systems in the context of a timeline of distributed, agent and Web service systems. Based on a requirements matrix (see Section 3.2) and guiding principles
(see Section 4.1), a global view, coupled with data, component, behavioural and communication models, are presented to define the SSOA reference architecture from five different perspectives in Chapter 4.

At its core, the SSOA is premised on giving equal weight to the complimentary, but independent, perspectives of Web service requesters and providers. The driving motivation is to enable Web services to be discovered, composed and invoked, as needed at runtime, based on a service requester’s description of what is to be achieved, in terms of the ontologies they use. In the presented architecture and prototype implementation, both requesters and providers are free to define the goal and Web service descriptions which represent them, using whichever ontologies and process models they choose. The SSOA shares its choice of ontology for its data model with both SWE-ET and IRS. The differences between these frameworks and the SSOA is described in Sections 7.2 and 8.1.6. One example of this is that, in contrast with IRS and SWE-ET, the SSOA does not mandate that specific mediators between goal and Web service descriptions be established prior to discovery or invocation operations. The experimental validation of the SSOA is provided in Section 6.5. In Chapter 7, the WSMX reference implementation of the SSOA is evaluated both experimentally and qualitatively, with respect to two other frameworks in the context of the SWS Challenge, identifying both the strengths and weaknesses of the application of the SSOA model from a bottom-up perspective.

9.1.2 Novel Algorithms for Discovery and Process Mediation

In Chapter 6, I described in detail, how an open-source implementation of the SSOA solved a set of integration problems independently formulated and published by the SWS Challenge. Challenges arose in the course of this work which led to the need for additional specific investigation and experimentation on individual functional components of the architecture, driven by bottom-up, problem-based requirements. Two specific challenges were in the areas of (1) Semantic Web service discovery where incomplete data available in the service description must be augmented by fetching additional data at the time the service is required, and (2) the enactment of process mediation between the choreography interface description of a service requester goal and a matching offered Web service. Devising solutions to these problems, applying them to real-world problems and evaluating their outcomes both experimentally (in Chapter 6.5) and qualitatively (in Chapter 7) represent the second contribution of this thesis.

The first challenge, described in Section 5.1, is the scenario where not all data necessary for Web service discovery is published in the static descriptions of Web services. An algorithm is presented, using the conceptual data model of WSMO, specification of a dedicated data-fetch interface, and the design of the SSOA choreography and reasoner components, where data is efficiently fetched, as needed at run-time, and provided as input to the discovery algorithm. The technique is the subject of joint work
CHAPTER 9. CONCLUDING REMARKS

published at [Zaremba et al., 2007a]. Its key contribution is the demonstration of an efficient solution to the pre-contracting step of fetching dynamic service information at run-time, achieved by analysing the rule definitions which comprise WSMO goal and Web service choreography interface descriptions. The use of the choreography interface description as a means to determine what data to fetch, and when it should be fetched, is a distinct novel aspect of the approach. The results of the algorithm’s experimental evaluation are described in Section 6.3.

Additionally, in Section 5.2, I describe how the discovery algorithm can be extended to allow for simple combined discovery & composition of Semantic Web services, which can cater for constraints to be specified on individual sub-clauses of the service requester goal. The contribution is a lightweight approach to service composition which allows constraints to be applied both on (1) local clauses of a goal, and (2) over all clauses of the goal as a whole. The results of the algorithm’s experimental evaluation, described in Section 6.4, demonstrate that the technique was sufficient to solve the combined discovery & composition problems of the SWS Challenge. The algorithm extending service discovery for composition is the result of joint work published at [Moran et al., 2007].

The second challenge – enacting process mediation – involved two parts. The first was the design of a behaviour to receive incoming data, use ontology-driven data mediation to ensure the information is made available in terms of both ontologies of the goal and Web service respectively, and then make the data available to the SSOA choreography components. This is described in Section 4.5.2. The second part was the definition of an algorithm for the SSOA choreography component, which uses the rule-based choreography interface descriptions of both the goal and Web service to coordinate what data, and when, is dispatched to the endpoints defined in the goal and Web service grounding descriptions. This is defined in Section 5.3. The application of the overall solution to process mediation is demonstrated in detail in Section 6.2, which describes how this approach was successfully able to (1) solve the initial mediation problem of the SWS Challenge, and (2) solve the modified mediation problem by only changing the semantic descriptions of the goal and Web services. The development of the design and algorithm for process mediation is published as joint work in [Vitvar et al., 2008].

9.1.3 Qualitative Comparison of the SSOA to Peer Semantic Web Service Frameworks

A consequence of the absence of a reference architecture for SSOA is the lack of a frame-of-reference to qualitatively compare the models of contemporary Semantic Web service frameworks. The third contribution of this thesis is to provide a set of reference criteria against which such frameworks can be compared so as to better understand the strengths and weaknesses of the respective models. Based on
the matrix of functional and non-functional requirements derived in Section 3.2, and (2) the set of five models for the SSOA described in Chapter 4, a comparative analysis of the three initial most prominent Semantic Web service frameworks is presented in Chapter 8. The chapter identifies the strengths and weaknesses of these frameworks with respect to the SSOA reference architecture. Additionally, in Section 7.3.1, I use the requirements matrix to critically review the SSOA with respect to two peer participant frameworks in the SWS Challenge (DIANE and SWE-ET).

9.2 Open Questions and Future Research

This section lists a number of open research questions, some of which are published as part of my contribution to the joint publication at [Kashyap et al., 2008].

**Integration Problem.** Regardless of how services find each other, the processes at the interfaces of services wishing to interact need to overcome their data and process heterogeneity and have to match for a successful communication. If a solvable mismatch exists, the communication between the services needs to be mediated. A lot of the focus remains on discovery and composition for services to find each other, but data and process mediation remains largely an unsolved problem, notwithstanding the work presented in Sections 4.4.4, 5.3, and 6.2.

**Efficiency, Speed, Security.** Semantic Web service expectations are high but there are some elephants lurking in the corners. Speed in the case of the SOAP/HTTP model; lack of support for process models in the REST HTTP/XML model (and possibly reliability and security at the message level rather than the transport protocol level); the focus on grounding is to WSDL but the focus of the WSDL binding is more or less based on Remote Procedure Calls (see Section 2.1.1). Add to this the persisting reality that logical reasoning engines for the major Semantic Web service languages are not yet demonstrated to scale to industry levels. These are all infrastructure issues and so questions needs to be resolved on where the improvement in efficiency will come from what has been promised. In other words there is still no solution to the Proposed Challenge for Measuring the Success of SWS [Han and Roman, 2006].

**SSOA Model Contains Only Essential Components.** I deliberately kept non-functional components such as security, auditing, and monitoring of service-level agreements out of scope of the reference SSOA, as I believe that these problems become relevant only after the problems associated with the mediation and discovery phases of Semantic Web service execution are resolved. However, a blueprint for research challenges in this area is presented by the Semantic Web Service Architecture (SWSA) [Burstein
et al., 2005b] (briefly described in Section 2.1.3), and serves as the basis for further investigation on these topics.

**Humans Are Still Needed** This thesis presents an SSOA architecture and implementation to enable independent heterogeneous software agents to discover and interact with each other. Achieving this requires a reasonable amount of expertise and knowledge in modelling ontologies, Semantic Web services, and service requester goals. The vision that software agents can find each other dynamically and figure out any mediation needed for interoperation, is still not achieved. In particular, for fuzzy problems humans are still in the loop and probably will for some time. It is worth noting, however, that positive steps, with regard to modelling and handling fuzzy matching of service requests and offers, are underway in the DIANE project (see Section 7.1).
Part VI

Appendix
A. Semantic Web Service Ontologies

To realize the formal, explicit aspect of Gruber’s definition [Gruber, 1993], ontologies are represented using a language with well-defined logical properties that can be validated and enforced by a logical reasoning engine. Consequently, there are two aspects of each of the models to consider, (1) the conceptual model defining the terms that go to make up the ontology, and (2) a language syntax that defines the formal explicit semantics.

A.1 WSMO

A.1.1 Conceptual Model

WSMO was initially derived from the Web Services Modeling Framework (WSMF) [Fensel and Bussler, 2002], refining its model and defining a language. WSMF proposed two fundamental principles:

- Strong decoupling of elements in the model
- Scalable mediation to overcome heterogeneity

WSMO is a meta-ontology in terms of the Object Management Group (OMG) Meta Object Facility (MOF) [OMG, 2010] specification for an abstract language and framework to represent its meta-models. MOF provides the constructs of classes and their generalisation through sub-classes as well as attributes with declarations of type and multiplicity. The four layers of MOF and how they relate to WSMO are described in detail in [Roman et al., 2005]. The relationship to MOF is intentional, as MOF is an ISO standard (ISO/IEC 19502:2005) that enables exchange of interoperable metamodels.

Figure A.1 shows the four top level elements defined by WSMO. These are Ontologies, Web Services, Goals and Mediators. Each of these elements is represented as a WSMO class with various attributes. Attributes have their multiplicity set to multi-valued by default. If an attribute is single-valued, this is explicitly stated. All WSMO elements have the attribute hasNonFunctionalProperty. This allows for the assignment of any non-functional properties (e.g. related to quality-of-service, price, meta-data regarding
the owner of the element etc.) to any element. WSMO recommends most elements of the Dublin Core metadata initiative\textsuperscript{59}. The following sections provide a brief description of each of the top level WSMO elements. For complete descriptions refer to the WSMO specification.

**Ontologies**

They are used to define the information model for all aspects of WSMO. Compared to structural languages used to define taxonomies such as XML, ontologies allow for the formal definition of concepts and attributes in addition to restrictions and rules constraining them as well as functions and relations that range over them. Two key distinguishing features of ontologies are the principle of a shared conceptualization and a formal semantics. Ontologies are only useful if the meaning the express corresponds to a shared understanding of its users. Likewise, the strength of an ontology is that the semantics of its elements are machine understandable, made possible through the provision of a mathematical base for the language used to express the ontology. Ontologies defined in WSMO are part of the MOF model layer.

The WSMO ontologies has attributes of the following types: ontologies (enable import of other ontologies), ooMediator (to deal with data heterogeneity), concept (define the basic elements of the problem domain), relation (map interdependencies between concepts), function (a special type of relation with a unary range), instance (instances of concepts defined for the ontology or an imported ontology), and axiom (logical expressions over terms defined by the ontology).

**Web Services**

From a simplified perspective, WSMO Web services are defined by the functional capability they offer and one or more interfaces that describe the communication model of the service. Capability is one example

\textsuperscript{59}http://dublincore.org/ (accessed 10 March 2011)
of an attribute of a WSMO class (i.e. Web service) that is single-valued. In WSMO a Web service is defined as offering exactly one capability. The Web Service class also has attributes for mediators (align to another ontology or link to a goal), non-functional properties (as described above) and ontologies that are imported (providing domain models for some part of the description). We will focus on the capability and interface descriptions as this is one area where the similarities and differences between WSMO and OWL-S are apparent.

**Capability**  The capability of a Web Service in the WSMO model defines the functionality that the service can provide when invoked by a service requester. It is defined using a state transition model. Prior to a Web service invocation, preconditions define the required state of the information space available to the Web Service and assumptions define the state of the world outside that information space. An example of a precondition when using a Web Service to purchase goods is that a credit-card number is valid or that a post-code is valid for the delivery scope of the service. An example of an assumption is that the address provided actually exists. Preconditions and assumptions are defined using sentences in a logical language known as axioms. Depending on the language used, the axioms can be more or less expressive.

Correspondingly, where a service executes successfully, postconditions are used to define the state of the information space and effects describe the state of the world outside the information space. For example, a postcondition might be that an shipment confirmation message is sent to the service requester, and an effect might be that the goods are physically put into a container and shipped.

All four types of condition are optional in the capability description. The service can be considered as one or more state transitions which lead from the state, defined by the preconditions and assumptions, to the state defined by the postconditions and effects. An application, wishing to locate a service for a specific task, uses the capability description of a WSMO service to determine if it offers the required functionality. Universally quantified shared variables are used to allow information to be shared between the four conditions supported by capability descriptions.

**Interface - Choreography and Orchestration**  Where the capability defines what a service offers, the WSMO Web service interface elements describe views of external parties on how they can interact with the service. This is sub-divided into two further elements, choreography and orchestration. The interface choreography element describes how a service requester can interact with the service to achieve their goal including message exchange patterns, the process model supported and the definition of the information types exchanged at the interface.

The interface orchestration element allows for the definition of a Web service as an orchestration
of other co-operating services (or goals which are described later). The idea is not that all (or indeed any of the) details of how a service achieves its capability must be made public but rather an explicitly described orchestration, including control flow, data flow, and data definitions facilitate the separation of the description of (1) how the Web Service achieves it aims from (2) its implementation.

Both choreography and orchestration elements of WSMO Web Services are modelled using Abstract State Machine (ASMs) [Yuri Gurevich, 2003; Börger, 1999]. ASMs were chosen as a general model as they (1) provide a minimal set of modelling primitives (no ad-hoc elements), (2) were judged to be sufficiently expressive, and (3) provide a rigid mathematical model for expressing dynamics.

Goals

WSMO Goals are used to describe, from their own perspective, the aims service requesters have when they wish to interact with Web Services. The separation of Goal and Web Service descriptions in WSMO is the realisation of the objective to separate concerns. Service requesters are free to specify the services that they require in their own terms. Like Web Services, Goals are defined with attributes for non-functional properties, imported ontologies, mediators, capabilities and interfaces. All of these attributes are defined from the perspective of what a service requester would like to get from a Web Service. The matching of Goal and Web Service descriptions (service discovery) depends on the matching of these descriptions, which require logical reasoning (for machine understanding) and the use of one or more of the mediator types, defined by WSMO, to cater for interoperability issues.

Mediators

The last of the four top level elements of the WSMO conceptual model are mediators. They are used to bridge interoperability between any two WSMO elements. A number of distinctions are drawn in the WSMO Mediator model. The first is between the description of a mediator and its implementation. Where WSMO Web Service descriptions say nothing about how the services are implemented (they ground to WSDL for this), the same holds true for Mediators (they can be optionally grounded to a Goal, Web Service or another Mediator). They describe the bridge that is required between any two elements. A second distinction is between the kind of mediation that is necessary for Semantic Web Services and the types of Mediator that are defined by the WSMO model. The former breaks down to three varieties of mediation:

- **Data mediation.** Handle mismatches at the data definition level.

- **Protocol mediation.** Handle mismatches between message exchange protocols. This relates to the choreography descriptions of Web Services.
• **Process mediation.** Handle mismatches between heterogeneous business processes such as those defined by the RosettaNet or ebXML standards.

Other varieties of mediation may also become necessary over time. The list above is not considered exhaustive. The latter distinction is represented by the four types of mediator defined by WSMO:

• **OOMediators.** Cater for differences in the descriptions of data models defined by ontologies.

• **WGMediators.** Handle mismatches between the definition of a service request as expressed in a Goal and the definition of an offered service as expressed in a Web Service.

• **GGMediators.** Where a repository of Goals is already available, GGMediators allow Goals to be linked together where there are differences in their descriptions. For example, say a Goal is already known to match to a given Web Service, a match of a weaker Goal to the same Web Service may be facilitated through a GGMediator.

• **WWMediator.** This is analogous to the GGMediator. Where a given Web Service already is known to match a specific Goal, a weaker or stronger Web Service could also be matched to the same Goal through the use of a bridging WWMediator.

### A.1.2 Language

The formal syntax and semantics for WSMO is provided by the Web Service Modeling Language (WSML) [Bruijn et al., 2006] which is actually a layered family of formal description languages. Each member of the family corresponds to a different type of logics which allows for different expressivity and reasoning capability. The various logical formalisms are Description Logics, First-Order Logics and Logic Programming, all of which are useful for the modelling of Semantic Web Services. There are five variants, illustrated in Figure A.2 and described briefly below.

• **WSML-Core.** Defined by the intersection of Description Logic and Horn Logic based on Description Logic Programs. It is the least expressive member of the family and consequently, has the best computational characteristics.

• **WSML-DL.** An extension of WSML-Core which captures Description Logics SHIQ(D) and is equivalent to OWL-DL.

• **WSML-Flight.** This is based on a logic programming variant of F-Logic which provides convenient object-oriented and frame-based constructs. It extends WSML-Core with support for meta-modelling, constraints and non-monotonic negation.
- **WSML-Rule.** This extends WSML-Flight in the direction of Logic Programming to provide a powerful rule language.

- **WSML-Full.** Unifies WSML-DL and WSML-Rule under First-Order logic with extensions for the support of non-monotonic negation. This language is the least specified of the WSML family as it is not yet clear which formalisms are necessary to achieve it.

In Figure A.3, the layering of the WSML languages is illustrated with WSML-Core (least expressive) at the bottom and WSML-Full (most expressive) at the top. There are two possible layered combinations:

- WSML-Core + WSML-Flight + WSML-Flight + WSML-Full

- WSML-Core + WSML-DL + WSML-Full + WSML-Full

The two layerings are disjoint to the extent that only the WSML-Core subset provides a common basis. With that restriction, the semantics in the language remain relatively clean in contrast to SWSL-FOL and SWSL-Rule which, as described earlier, share common syntax but not semantics and consequently can not be used together. The rationale, provided by the authors of WSML, for not using OWL as the basis for the language is that OWL was designed as a language rich enough to describe the process models that are part-and-parcel of the Semantic Web service conceptual model. This is reflected directly in the OWL-S work where there is no alignment with a framework like MOF and which relies on OWL combined with different notations and semantics for expressing conditions. In some cases this leads to open semantics and decidability issues which are a hindrance to practical usage of the language.
A.2 OWL-S

A.2.1 Conceptual Model

OWL-S [Martin et al., 2004] is an ontology described using the W3C Web Ontology Language (OWL)\textsuperscript{60} for describing different aspects of Semantic Web Services. It is structured into three sub-ontologies. The first models the functionality a Web service offers, including the constraints and non-functional properties that influence it. This is described using the ServiceProfile. Web services enact their functionality through a behavioral model. Describing this is the aim of the ServiceModel. Finally, OWL-S seeks to build on top of WSDL and SOAP by mapping elements in the ServiceModel to elements in the WSDL description. This part of the OWL-S ontology is called the ServiceGrounding. Each of the three parts is looked at in the next paragraphs.

In OWL-S, the ServiceProfile describes what a Web service does and provides the means by which the service can be advertised. As there is no distinction in the conceptual model of OWL-S between service requests and service provisions, the ServiceProfile is aimed equally at advertising services offered by providers and services sought by requesters. Owing to its genesis in the research area of artificial intelligence (AI), OWL-S defines the capability a service offers in terms of a state transition. It is possible to specify the inputs and outputs expected to be sent to and received from a service along with preconditions that must hold before the service can execute and the effects of the service executing. The intent is that along with arbitrary non-functional properties, this should be sufficient information for a

\textsuperscript{60}http://www.w3.org/TR/owl-features/ (accessed 10 March 2011)
discovery agent to be able to decide if a desired ServiceProfile matches any of the ServiceProfiles in the set of candidate OWL-S Web service descriptions available to it.

The ServiceModel is used to define the behavioral aspect of the Web service. This part of the service is modelled as a process in the sense that a service requester can view the process description and understand how to interact with the service in order to access its functionality. In some ways, this process model can be considered as a partial workflow where the service requester provides the missing parts. The ServiceModel allows for the description of different types of services, atomic, abstract and composite. Atomic processes correspond to a single interaction with the service, e.g. a single operation in a WSDL document. Composite processes have multiple steps, each of which is an atomic process, connected by control and data flow. Simple processes are abstractions to allow multiple views on the same process. These can be used for the purposes of planning or reasoning. Simple processes are not invocable but are described as being conceived as representing single-step interactions. A simple process can be realized by an atomic process or expanded to a composite process. The final part of the conceptual model is the ServiceGrounding, providing a link between the ServiceModel and the description of the concrete realization for a Web service provided by WSDL. Atomic processes are mapped to WSDL operations, where the process inputs and outputs, described using OWL, are mapped to the operation inputs and outputs, described using XML Schema. It is possible that a single OWL-S Atomic Process can be mapped to many WSDL operations. Composite processes, being composed of atomic processes, are grounded in the same way with the additional requirement of an OWL-S process engine to interpret the defined control and data flow.

In many ways OWL-S was the first consensus-based ontology for describing Semantic Web services. It is the product of merging earlier research from two separate languages, DAML [Hendler and McGuinness, 2001] and OIL [Fensel et al., 2000], resulting in an ontology initially called DAML-S but later renamed to OWL-S to emphasize the perceived layering of the ontology on OWL (a W3C Recommendation). The actual use of the description logic variant, OWL-DL, as the ontology language for OWL-S led to some unwanted side effects noted in detail in [Balzer et al., 2004]. In particular, OWL-S does not comply with the OWL-DL specification, which places constraints on how OWL-S ontologies can be reasoned over. A second problem is that variables are not supported within OWL-DL but are necessary when combining data from multiple co-operating processes in OWL-S.

A.2.2 Language

Although primarily the OWL-S ontology is defined using the Web Ontology Language (OWL), OWL-S is actually a mixture of a number of languages. This breaks to some extent the claim for OWL-S that
APPENDIX A. SEMANTIC WEB SERVICE ONTOLOGIES

it is layered on top of OWL. The reason for the language mixture is that Web services are inherently associated with distributed computing on the Web through process definition and execution. OWL was not originally designed with this purpose in mind. Rather, it provides an upper ontology for defining conceptual models. In particular, to take advantage of the most commonly available implemented logical reasoners – those for OWL-DL – the language used to define the domain models used in the Semantic Web service descriptions.

When describing logical expressions for the preconditions and results of ServiceProfiles or Service-Models, the modeler has a choice. The Semantic Web Rules Language (SWRL) [Horrocks et al., 2004] and Resource Description Framework (RDF) [W3C, 2008] treat expressions as XML literals while the Knowledge Interchange Format (KIF) [Genesereth et al., 1992] or the Planning Domain Description Language (PDDL) [McDermott, 1998] can be used for treating expressions as string literals.

A.3 SWSF

The establishment of the Semantic Web Services Framework (SWSF) [Battle et al., 2005] was motivated by the recognition of some shortcomings of OWL-S as a conceptual model for Semantic Web services. At the time OWL-S was developed, attention was focussed on how an ontology for Web services could be described using OWL. OWL itself is layered on top of the Resource Description Framework (RDF), and it was considered an elegant solution to add OWL-S as a further layer. A significant problem, as indicated in Section A.2, is that OWL (or more precisely OWL-DL) is not well suited to describing processes. This situation is unsatisfactory as the functionality offered by Web services can be considered as a partial process involving the operations that the Web service makes available to a client application. The process description is partial as the client itself provides the complimentary activities when it interacts with the service.

SWSF was devised to provide a full conceptual model and language expressive enough to describe the process model of Web services. There are two parts to the SWSF. The first is a conceptual model called the Semantic Web Services Ontology (SWSO) axiomatised using first order logic, and the second is a language called the Semantic Web Services Language (SWSL).

A.3.1 Conceptual Model

SWSO defines a conceptual model for Semantic Web services with a deliberate focus on extending the work of OWL-S to interoperate with and provide semantics for industry process modeling formalisms like the Business Process Execution Language (BPEL). The first-order logic axiomatisation of SWSO is called FLOWS (First-Order Logic Ontology for Web Services) and is based on the Process Specification
Language (PSL) [Michel and Cutting-Decelle, 2004], an international standard ontology for describing processes in domains of business, engineering and manufacturing. One of the intentions of PSL was to provide a common interlingua for the many existing process languages, allowing interoperability to be established between them. As the number of conceptual models and languages for Semantic Web services grows, there is a perceived need for such an umbrella formalism to facilitate interoperability in this area.

As mentioned, FLOWS is axiomatised in first-order logic and is expressed in a language called SWSL-FOL (Semantic Web Services Language for First-Order Logic). To enable logic-programming-based implementations and reasoning for SWSO, there is a second ontology available called ROWS (Rules Ontology for Web Services) and this is expressed in SWSL-Rules. ROWS is derived from FLOWS by a partial translation. The intent of the axiomatisation of ROWS is the same as that of FLOWS but in some cases it is weakened because of the lower expressiveness of the SWSL-Rules language compared to SWSL-FOL.

Service is the primary concept in SWSO with three top-level elements, derived from the three parts of the OWL-S ontology. These are Service Descriptors, Process Model and Grounding.

**Service Descriptors.** They provide a set of non-functional properties that a service may have. The FLOWS specification includes examples of simple properties such as the name, author, and textual description. The set is freely extensible to include the properties identified in other conceptual models such as WSMO non-functional properties or OWL-S service profile elements. Metadata specifications for online documents including Dublin Core are also easily incorporated. Each property is modelled as a relation linking the property to the service. For example, Listing A.1 shows FLOWS relations for service name, version and reliability. Note that Web service reliability is a subjective notion in the context of the quality-of-service (QoS) attributes a service may have. For it to be effective, a formal description of the meaning of reliability in Web services is required. Some ongoing work in modelling this type of attribute using WSMO ontologies is described in [Vu et al., 2006a].

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name(service, service_name)</td>
</tr>
<tr>
<td>2</td>
<td>version(service, service_version)</td>
</tr>
<tr>
<td>3</td>
<td>reliability(service, service_reliability)</td>
</tr>
</tbody>
</table>

Listing A.1: FLOWS Service Descriptor Properties

**Process Model.** The underlying objective of PSL is to provide a language and ontology that is expressive enough that all other process languages can be represented in it. If this is achieved then the integration of independent processes described with heterogeneous models becomes possible. FLOWS extends the PSL generic ontology for processes with two fundamental elements, especially to cater to
Web services:

- The structured notion of atomic processes as found in OWL-S
- Infrastructure for allowing various forms of data flow

The Process Model of FLOWS is organized as a layered extension of the PSL-OuterCore ontology. The primary layer is called FLOWS-Core and contains the two extensions just mentioned for Web services. On top of this, five additional ontology modules are defined that are used to express different constraints on the occurrences of services and their subactivities. A simplified diagram of this layering is provided in Figure A.4.

![Figure A.4: FLOWS Layered Process Model [Battle et al., 2005]](image)

As defined in the SWSF submission to the W3C, the layer has five additional ontologies:

- **Control Constraints** axiomatize the basic constructs common to workflow-style process models. In particular, the control constraints in FLOWS include the concepts from the process model of OWL-S.
- **Ordering Constraints** are used to specify activities defined by sequencing properties of atomic processes.
- Occurrence Constraints support the specification of nondeterministic activities within services.
- State Constraints support the specification of activities triggered by states (of an overall system) that satisfy a given condition.
- **Exception Constraints** provide some basic infrastructure for modeling exceptions.

Four key terms defined by the FLOWS ontology are listed below:
• **Service.** A service is an object that can have an associated number of service descriptors as described above, and an activity that specifies the process model of the service.

• **Atomic Process.** An atomic process is generally a subactivity of the activity associated with a service. It is directly invocable, has no sub-processes and can be executed in a single step.

• **Message.** Messages have an associated message type and payload.

• **Channel.** A channel is an abstraction for an object that holds messages that have been sent but may not yet have been received. There is no restriction that all messages sent be associated with channels, but where this is the case there are additional axioms that must hold for the message.

FLOWS allows the modelling of predicates or terms whose values may change in the course of an activity. The modelling elements are called *fluents* and can be imagined as providing a behaviour similar to that of variables in a programming language, in that they allow processes to be chained together where a value from one process may be required by another. The absence of this was one of the observed drawbacks of the OWL-S process model.

**Grounding.** The SWSO approach to grounding follows very closely the grounding of OWL-S v1.1 to WSDL. The SWSO specification defines how the grounding must provide four things. These are:

• Mappings between the SWSO and WSDL messages patterns

• Mappings between message types as defined in SWSO and WSDL respectively

• Serialization from SWSO message types to the concrete message types defined by WSDL

• Deserialization from the concrete WSDL message types to the SWSO messages types

### A.3.2 Language

The Semantic Web Services Language (SWSL) comes in two variants: SWSL-FOL and SWSL-Rules. The starting point is SWSL-FOL which acts as a foundational ontology language with PSL as its foundation. SWS-Rules is derived as a partial translation to facilitate implementation and reasoning based on logic programming techniques.

Both variants share syntax but not the semantics of that syntax. In fact, neither language is a subset of the other, which means the two language variants are mutually incompatible (cannot be used together), which may somewhat complicate the understanding of how to use of SWSO/L. The modeler must decide which language best suits the purpose at hand. The decision is made simpler as each of the variants
APPENDIX A. SEMANTIC WEB SERVICE ONTOLOGIES

Figure A.5: SWSL-Rules Layers [Battle et al., 2005]

has a differing focus. SWSL-FOL is most useful for process-related descriptions while SWSL-Rules is geared toward the description of programming-like tasks such as discovery and contracting. Both variants comply with Web principles such as the use of URIs, integration with XML types and XML-compatible namespaces. Additionally both are layered languages where new features are incorporated at each layer.

A concise review of SWL-Rules is provided by the authors of [Brodie et al., 2005]. As described in this report, SWSL-Rules is a logic programming language including features from Courteous logic programs [Grosof, 1999], HiLog [Chen et al., 1993] and F-Logic [Kifer et al., 1995] and can be seen as both a specification and an implementation language. The SWSL-Rules language provides support for service-related tasks such as discovery, contacting, and policy specification. It is a layered language as illustrated in Figure A.5. The core of the SWSL Rules language is represented by a pure Horn subset. This subset is extended by adding features such as disjunction in the body and conjunction and implication in the head as described by Lloyd in [Lloyd, 1987], or negation in the rule body interpreted as negation as failure (called NAF). Other extensions are (1) Courteous rules (Courteous), (2) HiLog, and (3) Frames.

On the other hand, SWSL-FOL, intended to describe the dynamic (process) aspect of services, is also layered. The bottom layer of Figure 11.6 shows the layers of SWSL-Rules that have monotonic semantics and therefore can be extended to full first-order logic. The most basic extension is SWSL-FOL but Figure A.6 also shows three other possible layered variants that can be achieved by the relevant extension. These are SWSL-FOL+Equality, SWSL-FOL+HiLog and SWSL-FOL+Frame.
A.4 WSDL-S

WSDL-S [Akkiraju et al., 2005] is a lightweight approach for augmenting WSDL descriptions of Web services with semantic annotations. It is a refinement of the work carried on by the METEOR-S group at the LSDIS Lab, University of Georgia, Athens, Georgia, to enable semantic descriptions of inputs, outputs, preconditions and effects of Web service operations by taking advantage of the extension mechanism of WSDL. WSDL-S is agnostic to the ontology language and model used for the annotations of WSDL. The following paragraphs briefly describe (1) the approach of WSDL-S, (2) the conceptual model representing the approach, and (3) the extensions to the WSDL language that realize the semantic annotations.

Approach. In contrast to the OWL-S, SWSO and WSMO, WSDL-S does not specify an ontology for the definition of Semantic Web services. Rather, it takes a bottom-up approach with the appeal that potentially only a little additional effort on the part of service producers will provide a service description where the description of the data and operations of the service are bound to ontological concepts. WSDL-S intentionally builds directly on the existing Web service technology stack.

WSDL v1.1 allows for the definition of extension to its language. This is taken advantage of to provide an in-document link of certain WSDL elements to concepts in one or more ontologies (assuming that the concepts can be identified uniquely and that the links can be specified in legal XML). Figure A.7 provides a high-level overview.
Embedding annotations into WSDL through legal language extensions does not affect the usage by the service provider of any other WS-* specifications or the usage of WSDL in the context of process description languages such as the Business Process Execution Language for Web services (WSBPEL) [Alves et al., 2006]. Another feature is that where XML Schema is used as the data definition language for WSDL, it can be enhanced by linking XML Schema types to domain concepts either by a one-to-one mapping or through a transformation defined in a domain ontology.

**Conceptual Model.** WSDL-S defines its conceptual model using a simple XML Schema, shown in Listing A.2, introducing five elements that extend WSDL. These are:

- **modelReference.** This is used for annotating both simpleTypes and complexTypes in XML Schema where there is a one-to-one mapping between the schema type and the ontological concept. For simpleTypes, it is a direct mapping. For complexTypes, it can be used in two ways, bottom-up and top-down. Bottom-level annotation involves describing every leaf element of the complexType with the modelReference attribute. Top-level annotation means that the complexType element itself is associated with a concept in the ontology. The assumption is that the subelements of the complexType will map directly to the sub-concepts and attributes of the domain concept.

- **schemaMapping.** Where there is no one-to-one mapping this attribute points to a transformation that links the XML Schema element to the ontology concept. For example, the value of the schemaMapping attribute might be a URI that identifies an XSLT transformation.

- **precondition.** At the level of a WSDL operation it is possible to point to a definition of the precon-
dition that must hold before that operation can be executed. For simplicity only one precondition may be included and this may point to a set of logical expressions in the ontology language of choice.

- **effect.** Similar to preconditions, effects point to logical expressions that should hold after the execution of the service. In contrast to preconditions, WSDL-S allows for the definition of multiple-effect subelements of operations.

- **category.** This is adopted from OWL-S and is an extension to the WSDL Interface element of WSDL 2.0 (portType in WSDL 1.1). The intent is that category information can be included here that may be picked up by a Web service registry implementation such as the one for UDDI.
A.5 SAWSDL

The W3C Semantic Annotations for WSDL (SAWSDL) [Kopecký et al., 2007b] working group provides a W3C Candidate Recommendation for Semantic Web services based on a simplified form of WSDL-S. As with WSDL-S, the approach is agnostic to the ontological model used to define the semantics of annotated WSDL elements. From SAWSDLs perspective, the annotations are connected to the descriptions using URIs. SAWSDL is targeted at WSDL v2.0 but it is also possible to use with WSDL v1.1 with an additional non-standard extension.

While WSDL-S specifies the attributes for modelReference, schemaMapping, precondition, effect and category, SAWSDL confines itself to attributes of modelReference and two specializations of schemaMapping, namely, liftingSchemaMapping and loweringSchemaMapping. The modelReference attribute can be used to annotate XSD complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. The liftingSchemaMapping can be applied to XML Schema element declaration, complexType definitions and simpleType definitions. All attributes defined by SAWSDL are defined by the XML Schema, reproduced in Listing A.3, to take a list of URIs as value.

```
<Listing A.2: XML Schema for WSDL-S

A.5 SAWSDL

The W3C Semantic Annotations for WSDL (SAWSDL) [Kopecký et al., 2007b] working group provides a W3C Candidate Recommendation for Semantic Web services based on a simplified form of WSDL-S. As with WSDL-S, the approach is agnostic to the ontological model used to define the semantics of annotated WSDL elements. From SAWSDLs perspective, the annotations are connected to the descriptions using URIs. SAWSDL is targeted at WSDL v2.0 but it is also possible to use with WSDL v1.1 with an additional non-standard extension.

While WSDL-S specifies the attributes for modelReference, schemaMapping, precondition, effect and category, SAWSDL confines itself to attributes of modelReference and two specializations of schemaMapping, namely, liftingSchemaMapping and loweringSchemaMapping. The modelReference attribute can be used to annotate XSD complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. The liftingSchemaMapping can be applied to XML Schema element declaration, complexType definitions and simpleType definitions. All attributes defined by SAWSDL are defined by the XML Schema, reproduced in Listing A.3, to take a list of URIs as value.

```

```
<Listing A.3: XML Schema for SAWSDL

```

```
```
Bibliography


192


193
BIBLIOGRAPHY


194
BIBLIOGRAPHY


195


198


# List of Figures

3.1 Traditional Web Services Architecture ........................................... 28  
3.2 Semantic Web Services System ...................................................... 31  
3.3 Running Example ................................................................. 32  
   
4.1 Summary of SSOA Requirements .................................................. 44  
4.2 SSOA Overview (Adapted from [Vitvar et al., 2007a]) ....................... 46  
4.3 Comparison of SWS Conceptual Models .......................................... 52  
4.4 Choreography and Orchestration of a WSMO Web Service [Roman et al., 2005] ................... 54  
4.5 SSOA Component Model .......................................................... 55  
4.6 Logical Reasoning Interface ...................................................... 56  
4.7 Service Registry Interface ....................................................... 57  
4.8 Data Mappings Store Interface ................................................... 57  
4.9 Communication Manager Interface .............................................. 58  
4.10 Lifting Lowering Adapter Interface ............................................. 59  
4.11 Event Broker Interface ........................................................... 60  
4.12 Execution Controller Interface .................................................. 60  
4.13 Discovery Interface ............................................................... 61  
4.14 Selection Interface ............................................................... 61  
4.15 Data Mediation Interface ........................................................ 62  
4.16 Choreography Interface .......................................................... 62  
4.17 Mismatch Patterns handled by Process Mediator [Cimpian, 2007] ................. 63  
4.18 Process Mediation Interface ..................................................... 64  
4.19 Goal Driven Service Discovery Behaviour ..................................... 67  
4.20 Receiving Incoming Data Behaviour ........................................... 68  
4.21 Event-Driven Communication Mechanism ..................................... 70  
   
5.1 Makeup of the Knowledge Base, $B_{gw}$ [Vitvar et al., 2007b] ............... 75  
   
6.1 SWS Challenge Scenario Overview (reproduced from SWS Challenge Website) ................... 91  
6.2 SWS Challenge Evaluation Criteria: Success Levels .......................... 92  
6.3 Mediation Requirements from Blue’s Perspective ............................... 93  
6.4 Mediation Requirements from Moon’s Perspective ............................... 94  
6.5 Runtime Activity for the Mediation Problem .................................. 97  
6.6 Architecture for the Scenario .................................................... 104  
6.7 Rummage Supplier Products ..................................................... 110
LIST OF FIGURES

6.8 Bargainer Supplier Products ........................................ 111
6.9 Evaluation of Challenge Mediation Problem ........................ 115
6.10 Evaluation of Discovery with Simple Constraints ................ 115
6.11 Evaluation of Discovery with Constraints Requiring Calculation 116
6.12 Evaluation of Discovery with Constraints and Invocation ........ 116
6.13 Evaluation of Discovery with Detailed Product Specification ... 119
6.14 Discovery with Detailed Specification including Preferences .... 119
6.15 Composition Types Specified by the SWS Challenge .............. 120
6.16 Discovery with Composition of Services .......................... 120

7.1 DSD Service Description for a Printing Service [Kuester et al., 2007] 124
7.2 DSD Middleware [Kuester and Koenig-Ries, 2006] .................... 126
7.3 Glue Middleware [Valle and Cerizza, 2005] .......................... 130
7.4 Combined Glue and Webml/WebRatio Architecture [Brambilla et al., 2007] 131
7.5 Glue Discovery - Setup Phase [Valle and Cerizza, 2005] ............ 133
7.6 SSOA Requirements vs WSMX, DIANE and SSW-ET .................. 135

8.1 Summary of SSOA Requirements ..................................... 145
8.2 The IRS-III Server [Domingue et al., 2008] .......................... 147
8.3 SSOA Requirements vs IRS-III ....................................... 151
8.4 Architecture of the DAML-S VM [Paolucci et al., 2003] .......... 153
8.5 Architecture of the OWL-S Broker [Paolucci et al., 2004a] ...... 153
8.6 Behaviour Model for the OWL-S Broker [Paolucci et al., 2004a] 157
8.7 SSOA Requirements vs OWL-S Broker ............................... 158
8.8 Architecture of METEOR-S [Verma et al., 2005a] .................. 160
8.9 SSOA Requirements vs METEOR-S .................................. 164

A.1 WSMO Elements [Roman et al., 2005] ............................... 175
A.2 WSML Family [Bruijn et al., 2006] .................................. 179
A.3 WSML Layering [Bruijn et al., 2006] ................................ 180
A.4 FLOWS Layered Process Model [Battle et al., 2005] .............. 184
A.5 SWSL-Rules Layers [Battle et al., 2005] .......................... 186
A.6 Layers of SWSL-FOL and relationship to SWSL-Rules [Battle et al., 2005] 187
A.7 Associating semantics with WSDL [Akkiraju et al., 2005] ...... 188
Listings

5.1 Sample WSML Ontology Fragment for Data Fetch 78
5.2 Sample Data-Fetch Interface [Vitvar et al., 2007b] 78
5.3 Sample Data-Fetch Interface [Vitvar et al., 2007b] 78
5.4 WSML Capability Postcondition for a Composite Goal 79
6.1 Lifting in XSLT and resulting WSML message 96
6.2 Requester’s Service Choreography 100
6.3 Muller Shipper Service Description 103
6.4 ShipmentOrderRequest relation 105
6.5 isShipped relation 106
6.6 Goal B1: Discovery Based on Destination and Weight 107
6.7 Goal B1 Capability 107
6.8 Goal C1 Capability 108
6.9 Mueller Data Fetching Interface 109
6.10 Goal requiring Correlated Composition 111
6.11 isCompatible relation 112
6.12 User Goal in WSMO 113
6.13 WSML Postcondition for Goal C3 117
6.14 WSML Axiom for Price Calculation – Racer Web Service 117
6.15 WSML Transition Rules and Axiom for Mueller Service 118
6.16 WSML Goal Postcondition for Challenge Goal C3 121
A.1 FLOWS Service Descriptor Properties 183
A.2 XML Schema for WSDL-S 189
A.3 XML Schema for SAWSDL 190