Semantics Enriched Service Environments

Karthik Rajagopal Gomadam

Wright State University

Follow this and additional works at: http://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation

This Dissertation is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu.
Semantics Enriched Service Environments

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

By

KARTHIK GOMADAM
M.Sc (Mathematics), Sri Satya Sai Institute of Higher Learning, 2001

2009
Wright State University
August 25, 2009


Amit P. Sheth, Ph.D.
Dissertation Director

Thomas Sudkamp, Ph.D.
Director, Computer Science and Engineering Ph.D. Program

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Committee on Final Examination

Amit P. Sheth, Ph.D.

Lakshmish Ramaswamy, Ph.D.

Michael L. Raymer, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Guozhu Dong, Ph.D.

Shu Shiller, Ph.D.
ABSTRACT

Karthik Gomadam, Ph.D., Department of Computer Science and Engineering, Wright State University, 2009.
Semantics Enriched Service Environments.

During the past seven years services centric computing has emerged as the preferred approach to architect complex software. Software is increasingly developed by integrating remotely existing components, popularly called services. This architectural paradigm, also called Service Oriented Architecture (SOA), brings with it the benefits of interoperability, agility and flexibility to software design and development. One can easily add or change new features to existing systems, either by the addition of new services or by replacing existing ones. Two popular approaches have emerged for realizing SOA. The first approach is based on the SOAP protocol for communication and the Web Service Description Language (WSDL) for service interface description. SOAP and WSDL are built over XML, thus guaranteeing minimal structural and syntactic interoperability. In addition to SOAP and WSDL, the WS-* (WS-Star) stack or SOAP stack comprises other standards and specification that enable features such as security and services integration. More recently, the RESTful approach has emerged as an alternative to the SOAP stack. This approach advocates the use of the HTTP operations of GET/PUT/POST/DELETE as standard service operations and the REpresentational State Transfer (REST) paradigm for maintaining service states. The RESTful approach leverages on the HTTP protocol and has gained a lot of traction, especially in the context of consumer Web applications such as Maps.

Despite their growing adoption, the stated objectives of interoperability, agility, and flexibility have been hard to achieve using either of the two approaches. This is largely because of the various heterogeneities that exist between different service providers. These heterogeneities are present both at the data and the interaction levels. Fundamental to addressing these heterogeneities are the problems of service Description, Discovery, Data mediation and Dynamic configuration. Currently, service descriptions capture the various operations, the structure of the data, and the in-
vocation protocol. They however, do not capture the semantics of either the data or the interactions. This minimal description impedes the ability to find the right set of services for a given task, thus affecting the important task of service discovery. Data mediation is by far the most arduous task in service integration. This has been a well studied problem in the areas of workflow management, multi-database systems and services computing. Data models that describe real world data, such as enterprise data, often involve hundreds of attributes. Approaches for automatic mediation have not been very successful, while the complexity of the models require considerable human effort. The above mentioned problems in description, discovery and data mediation pose considerable challenge to creating software that can be dynamically configured.

This dissertation is one of the first attempts to address the problems of description, discovery, data mediation and dynamic configuration in the context of both SOAP and RESTful services. This work builds on past research in the areas of Semantic Web, Semantic Web services and Service Oriented Architectures. In addition to addressing these problems, this dissertation also extends the principles of services computing to the emerging area of social and human computation. The core contributions of this work include a mechanism to add semantic metadata to RESTful services and resources on the Web, an algorithm for service discovery and ranking, techniques for aiding data mediation and dynamic configuration. This work also addresses the problem of identifying events during service execution, and data integration in the context of socially powered services.
## Contents

1 Introduction .......................................................... 1

2 Background and Literature Survey .............................. 7
   2.1 Specification .................................................. 7
       2.1.1 OWL-S .................................................... 8
       2.1.2 Web Services Modeling Ontology - WSMO ................. 9
       2.1.3 METEOR-S ................................................. 10
           2.1.3.1 SAWSDL ............................................ 11
           2.1.3.2 Semantic Template .................................. 12
       2.2 Discovery ..................................................... 14
   2.3 Data Mediation .................................................. 17
   2.4 Composition ..................................................... 18

3 Motivation: Web applications, the way you want them .... 22

4 Of Faceted Search and Serviu Rank ......................... 26
   4.1 Overview ..................................................... 27
   4.2 Indexing APIs ................................................ 29
       4.2.1 Creating Term and Tag Vectors ......................... 30
           4.2.1.1 Bootstrapping Term Vectors ....................... 33
       4.3 Classification ................................................. 34
           4.3.0.2 Computing Similarity ............................... 34
   4.4 Searching ....................................................... 35
   4.5 Serviu Rank ..................................................... 36
   4.6 Evaluation ...................................................... 40
       4.6.1 Accuracy of Classification ............................... 41
       4.6.2 Precision and Recall ..................................... 42
       4.6.3 Effectiveness of Ranking ................................. 43

5 SEMRE: A SAWSDL Based Registry ............................. 45
   5.1 Motivation ...................................................... 46
CONTENTS

5.1.1 Motivating Scenario .................................................. 46
5.1.2 Registry as the centerpiece .......................................... 48
5.2 Service Models .......................................................... 49
  5.2.0.1 SAWSDL .............................................................. 49
5.3 Business assertions in WS-Policy ..................................... 49
5.4 System Design ............................................................ 51
  5.4.1 Components in the registry middleware ......................... 53
  5.4.2 Data Model ............................................................. 53
5.5 Publication and Selection .............................................. 55
  5.5.1 Definitions ............................................................ 55
  5.5.2 Publication .............................................................. 58
  5.5.3 Selection ............................................................... 60
    5.5.3.1 Matching Functional Requirements ......................... 61
    5.5.3.2 Matching Non-Functional Requirements ..................... 62
    5.5.3.3 Ranking ............................................................. 64
5.6 Evaluation ............................................................... 65
  5.6.1 Implementation ...................................................... 65
  5.6.2 Evaluation ............................................................. 66
    5.6.2.1 Test case generation ........................................... 67
    5.6.2.2 Experimental Results ......................................... 68
6 Data Mediation, Mediatability and Mediation as a service ...... 72
  6.1 Motivation ............................................................... 75
  6.2 Mediatability: Definition and Computation ....................... 76
    6.2.1 Overview ............................................................. 77
    6.2.2 Computing Element Similarity .................................. 78
    6.2.3 Factoring Structural Similarity ................................ 81
    6.2.4 Computing mediation similarity .................................. 82
    6.2.5 Calculating Mediatability ........................................ 84
    6.2.6 Optimizing Time Complexity ..................................... 86
  6.3 Evaluation ............................................................... 86
    6.3.1 Evaluating Accuracy .............................................. 87
    6.3.2 Impact of Annotation ............................................. 88
    6.3.3 Evaluating the Scalability ....................................... 89
7 Bringing They All Together: Composition and Configuration ... 91
  7.1 Motivating Scenario ................................................... 92
  7.2 Declarative Approach towards Solution ............................ 93
    7.2.1 Formal model of abstract Web services ...................... 94
  7.3 Automatic Web service composition ................................ 97
    7.3.1 Formal definition of Web service composition .............. 97
    7.3.2 Composition of semantic Web services ....................... 100
    7.3.3 Planning For Process Mediation ............................... 101
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>August 25, 2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.4 Extended GraphPlan Algorithm</td>
<td>103</td>
</tr>
<tr>
<td>7.3.5 Pattern-Based Approach For Loop Generation</td>
<td>104</td>
</tr>
<tr>
<td>7.3.5.1 Lifting and Lowering Mechanism of Data Mediation</td>
<td>105</td>
</tr>
</tbody>
</table>

8 Event Identification in SOA | 108

8.1 Overview of the Framework for Event Identification | 112
   8.1.1 Overview of the Functional and Non-Functional Ontologies | 112
   8.1.2 Outline of the Proposed Framework | 113

8.2 Defining the $\rho^{\text{path}}$ Query Operator | 113
   8.2.1 Event Identification using the $\rho^{\text{path}}$ query operator | 117

8.3 Computing event relevance | 120
   8.3.1 Adjusting the relevance based on feedback | 123

8.4 Evaluation | 125

9 Conclusion | 131

Bibliography | 134
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of the various components of the METEOR-S framework</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Semantic template modeling the requirements of the game manufacturer for ordering hard drives</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Steps involved in creating Big Mac finder application</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Illustrating the role of mediation in comparing image search schemas</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Snapshot of the Web API Taxonomy</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Example of Search for Image Search API’s that support XML or GData and use REST protocol</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Serviut Rank Example</td>
<td>37</td>
</tr>
<tr>
<td>4.4</td>
<td>(a) Normalized Popularity Scores for Mashups using API1; (b) Normalized Traffic Scores for Mashups using API1</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>Example of business assertions</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>(a) System Architecture (b) Extending a UDDI based registry</td>
<td>52</td>
</tr>
<tr>
<td>5.3</td>
<td>Hierarchical matching of the SEMRE registry</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Service Interface Publication Performance</td>
<td>68</td>
</tr>
<tr>
<td>5.5</td>
<td>Service Publication Performance</td>
<td>69</td>
</tr>
<tr>
<td>5.6</td>
<td>Impact of Match Criteria on the Accuracy</td>
<td>70</td>
</tr>
<tr>
<td>5.7</td>
<td>Discovery Performance</td>
<td>71</td>
</tr>
<tr>
<td>6.1</td>
<td>Search Services and Search Request Schemas</td>
<td>75</td>
</tr>
<tr>
<td>6.2</td>
<td>(a) Computing Element Similarity (b) Nearest Similar Ancestor</td>
<td>78</td>
</tr>
<tr>
<td>6.3</td>
<td>(a) Computing mediation similarity (b) Mediatability Computation</td>
<td>81</td>
</tr>
<tr>
<td>6.4</td>
<td>Accuracy Based on User Study</td>
<td>88</td>
</tr>
<tr>
<td>6.5</td>
<td>Impact of Semantic Annotation</td>
<td>89</td>
</tr>
<tr>
<td>6.6</td>
<td>Measuring Execution Time</td>
<td>90</td>
</tr>
<tr>
<td>7.1</td>
<td>SWS Challenge Scenario</td>
<td>93</td>
</tr>
<tr>
<td>7.2</td>
<td>Defining Web service composition</td>
<td>98</td>
</tr>
<tr>
<td>7.3</td>
<td>Different Heterogeneities</td>
<td>107</td>
</tr>
<tr>
<td>8.1</td>
<td>Identifying Events from semantic Template</td>
<td>114</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.2</td>
<td>Variation in the percentage of events identified with change in hop count before and after feedback-based adjustment.</td>
<td>127</td>
</tr>
<tr>
<td>8.3</td>
<td>Studying the performance of hybrid and fixed Adjustment schemes with variation in the total number of events.</td>
<td>127</td>
</tr>
<tr>
<td>8.4</td>
<td>Performance of hybrid and fixed adjustment schemes with variation in the percentage of relevant events</td>
<td>128</td>
</tr>
<tr>
<td>8.5</td>
<td>Variation in the accuracy of feedback schemes with increase in the number of events</td>
<td>129</td>
</tr>
<tr>
<td>8.6</td>
<td>Variation in the accuracy of the feedback schemes with increase in the percentage of relevant events</td>
<td>130</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Precision : Apihu, PWeb and Google ................................................. 40
4.2 Precision and Recall of Apihu ......................................................... 41
7.1 Representation of Order Management System Web service .................. 97
7.2 Representation of Order Management System Web service .................. 98
Acknowledgement

I am grateful to the many people who have helped me with this journey. I would first like to thank my advisor, Dr. Amit P. Sheth, for his guidance and support. His tremendous drive and dedication has shown me what it takes to be successful. His willingness to allow his students the freedom to explore various aspects of Computer Science, has played an important role in my growth as a researcher. I thank Dr. Lakshmish Ramaswamy for his valuable technical guidance during the course of my Ph.D. I would also like to thank the members of my dissertation committee, Professors T. K. Prasad, Guozhu Dong, Michael Raymer and Shu Shiller, for their support and insightful comments about my research. Dr. John Miller was one of my earliest collaborators and I will always cherish my interactions with him during my time at LSDIS.

I have had the privilege of working with a very active and energetic group of researchers during my studies, both at the LSDIS Lab and at the Kno.e.sis Center. Dr. Kunal Verma and Kaarthik Sivashanmugam find a special mention for mentoring me into research and for being a sound board for all my research ideas. Kunal has been instrumental in my working in the area of services computing and has been a source of inspiration for this work. I will forever cherish my association with Ajith Ranabahu. With his diligent effort, valuable insights, and his patience as my roommate for two years, Ajith has significantly contributed to my growth, both as a person and as a researcher. I am thankful to Jonathan Lathem, my first mentoree, for helping me translate my ideas to reality and for creating a platform of ideas for my further research. Matthew Evanson, Rob Tarr and Wesley Workman have helped me to explore new technical paradigms that have enabled me to prototype my research in useful and meaningful ways. I also acknowledge the contributions of my other colleagues: Wenbo Wang, Pablo Mendes, Delroy Cameron, Prateek Jain, Ashutosh Jadhav, Raghava Mutharaju, Pramod Anantharam, and Cory Henson for their feedback.

I am thankful for interactions with Drs. Sanjiva Weerawarana, Francisco Curbera, Rania Khalaf, Stefan Tai, Rong Chang, and Biplav Srivastava at IBM Research and for their valuable inputs during my internships. The skills I learned during my internship and through later collaborations have been a valuable asset while completing this dissertation. Also thanks to Michael Maximilien
of IBM Almaden Research for his continued collaboration and for his contributions to our research in the area of RESTful services.

Finally, I would like to thank my friends and family. Thank you to Christopher Thomas, Meenakshi Nagarajan and Dr. Cartic Ramakrishnan for being with me throughout some of the hardest times in this journey. Thank you to my parents, my sister for all their support and encouragement. I cannot thank Sindhuja enough for being my biggest support.
Dedicated to

My parents Usha and Ramanujam, my sister Sudha, and Sindhuja for their support
and to
Ajith, Cartic, Meena and Topher for being with me through the hardest and the best times
Introduction

One of the significant developments in the evolution of the Internet and the World Wide Web, has been the growth of the Web as a platform for application development, deployment, and distribution. The principles of Service Oriented Computing and Service Oriented Architecture (SOA) (Curbera et al. [10]) have played a significant role in this advancement. Software is increasingly developed by integrating remotely existing components, popularly called services. The foundations of SOA lie in distributed computing and earlier approaches to realizing distributed software systems such as CORBA, Remote Method Invocation (RMI) and the Distributed Component Object Model (DCOM). However, these earlier attempts were largely built around proprietary technologies and communication protocols. These limitations impacted the interoperability across software components. The objectives of SOA is to create an approach to agile and flexible software design, while overcoming the limitations of the earlier approaches. To achieve interoperability, SOA is built around standardized interfaces, communication object models and protocols. The resulting interoperability has made it possible to design software systems, whose capabilities can be changed or improved by adding or replacing existing services.

Two popular approaches have been developed for realizing SOA. The first approach is based on the SOAP protocol for communication and the Web Service Description Language (WSDL) for
service interface description. SOAP and WSDL are built over XML, thus guaranteeing minimal structural and syntactic interoperability. In addition to SOAP and WSDL, the WS-* (WS-Star) stack or SOAP stack comprises other standards and specifications that enable features such as security and services integration. More recently, the RESTful approach has emerged as an alternative to the SOAP stack. This advocates the use of the HTTP operations of GET/PUT/POST/DELETE as standard service operations and the REpresentational State Transfer (REST) paradigm approach proposed by Fielding et al. (15) for maintaining service states. The RESTful approach leverages on the HTTP protocol for communication and data formats such as XML and JSON for data representation. This approach has gained a lot of traction, especially in the context of consumer Web applications such as maps and data feeds.

Despite their growing adoption, the stated objectives of interoperability, agility, and flexibility have been hard to achieve using either of the two approaches. This is largely because of the heterogeneities that are present between services. The adoption of standards and open data formats has gone a long way in addressing the syntactic and structural heterogeneities. However, the standards and specifications often fail to capture the meaning or semantics, thus making it hard for automated/semi-automated understanding. This creates considerable challenges for service discovery and data mediation, thus affecting agility and flexibility. Research in the area of Semantic Web Services (SWS) addresses this shortcoming by adopting Semantic Web techniques. Semantic Web Services build upon shared domain models. These models, called ontologies, capture the various concepts and their relationships in a domain. Gruber (21) defined an ontology in computer science as a “specification of a conceptualization”. By annotating service descriptions with concepts from an ontology, one can add Semantic metadata that captures the meaning in a service description.
semantics for services can be classified into four categories: 1) Data semantics that capture the meaning of data elements in a service; 2) Functional semantics that describe the functional capabilities of a service; 3) Non-functional semantics to model the non-functional properties and 4) Execution semantics to capture the various execution states and faults (Sheth (61)).

This dissertation builds on past research in the area of Semantic Web, Semantic Web services and Service Oriented Architectures, and is one of the first attempts to address the problems of description, discovery, and data mediation in the context of RESTful services. We also address the problems of discovery and dynamic configuration of Semantic Web services and have developed techniques for systematic integration of RESTful services. In doing so, this work makes the following contributions:

1. **Description**: Service description plays an important role in discovery, data mediation and configuration. Our earlier work on WSDL-S (Sivashanmugam *et al.* (67)) matured into the W3C standard for Semantic annotation of WSDL and XML schema (SAWSDL) (Verma and Sheth (78)). RESTful services however, do not have a formal WSDL description and are often described in X/HTML documents. The two shortcomings we address are: 1) lack of Semantic information and 2) difficulty of extracting service information such as operations and data types from these descriptions. We address both of these issues by adopting a microformat based approach. Microformats are a lightweight way of adding additional metadata to Web documents using existing X/HTML tags. The SA-REST microformat, first proposed in Lathem *et al.* (28), is a mechanism for adding Semantic metadata to RESTful service descriptions. In addition to RESTful services, SA-REST can also be used as an annotation framework for any resource on the Web. The hRESTs microformat allows a user to add
markups to describe the service information and is discussed in Kopecký et al. (26).

2. **Search and Ranking:** Web service discovery is one of the most studied problems in the area of Semantic Web services. However, most of the past and current research address service discovery in the context of SOAP / WSDL services. In this dissertation, we present a search engine for searching and ranking RESTful APIs (Gomadam et al. (17)). Our work borrows ideas from faceted search discussed in Vickery (79) and Spiteri (69), document classification and indexing. We have also developed an algorithm called “Serviut rank” for ranking Web apis based on their utilization. We also discuss our registry framework for discovering and ranking SOAP services (Gomadam et al. (18)). While much of the prior work in this area (Benatallah et al. (3); Paolucci et al. (51); Li and Horrocks (30); Sivashanmugam et al. (66)) has focused on inputs and outputs, the work presented here factors in data, functional and non-functional requirements. Chapter 4 discusses the contributions in the area of RESTful API search in detail while Semantic Web service discovery in the context of SAWSDL services is presented in Chapter 5.

3. **Data mediation:** Data mediation is one of the most studied problems in computer science research. This problem has been studied in the context of multi-database and heterogeneous database systems (Kashyap and Sheth (25)), workflows and Service Oriented Architectures. Much of this work involves automatic mediation. More recent research such as Nagarajan et al. (46), explores techniques for reusable approaches to schema mediation using Semantic annotations. In this work, we discuss an approach that calculates the difficulty of mediation between two schemas for a human. We have defined a quantifiable metric called mediatability in Gomadam et al. (19) and have developed scalable and efficient algorithms for
calculating the same. We have also investigated approaches for a service oriented approach to mediation or data mediation as a service. This would allow the task of mediation to be seamlessly integrated into dynamic compositions (Wu et al. (81)). Definition and computation of mediatability is presented in Chapter 5.

4. **Declarative Approaches for Service Composition**: The ability to dynamically configure service oriented software is one of the significant outcomes of our work in the area of description, discovery and data mediation. We have developed a Semantic Web service middleware that supports deployment and execution of abstract, descriptive applications (Gomadam et al. (20)). These applications are created by integrating templates that capture requirements. The actual services for these requirements are discovered and the application is configured during its execution. We define mashups created using this approach as Smart Mashups or SMashups. We have developed a domain specific language for specifying these templates and the data flow within the application (Maximilien et al. (36)). A declarative approach for service composition is presented in Chapter 7.

5. **Event Identification in SOA**: In Chapter 8 we propose a framework for automatically identifying events as a step towards developing an adaptive middleware for Service Oriented Architecture (SOA). Identifying events that can impact the non-functional objectives of a service request is a key challenge towards realizing a more adaptive services environment. These events can either be user triggered in interactive applications such as mashups or can be trigged by providers themselves. Our approach allows users to capture their requirements in a descriptive manner and uses this description for identifying events of importance. This model is extended to adjust the relevance of the events based on feedback from the under-
lying adaptation framework. We present an algorithm that utilizes multiple ontologies for identifying relevant events and present our evaluations that measure the efficiency of both the event identification and the subsequent adaptation scheme.
Background and Literature Survey

This dissertation addresses research problems in the areas of service description, search, and integration. Central to the rapid adoption of Service Oriented Architecture has been the development of standards and specifications. Building on top of the interoperability of XML, Web service standards allow users to understand service descriptions and create software by integrating services in a standardized manner.

2.1 Specification

Web Service Description Language (WSDL) is the W3C standard for service description. The WSDL 2.0, written by Chinnici et al. [7] is a standard describing an interface element to capture service operations, their inputs, outputs and exceptions. WSDL 2.0 recommends the document literal approach for describing data elements. Data elements are exchanged as XML documents, thus facilitating data definition reuse. WSDL also separates the interface from the implementation. A specific implementation of an interface is captured in the service element. The binding element in WSDL describes a specific interaction mechanism, such as protocol, to be used when invoking a service.
2.1. SPECIFICATION

While WSDL and XML do address the issues of structural and syntactic variations in descriptions, they do not capture the semantic or the meaning of various service elements. For example, currency units amongst different providers can be different. The interaction protocols for complex transactions can vary from provider to provider. While standardization is an option, it is not realistic to expect adopters to deviate from their current approaches. Semantic Web Services is an effort to address this limitation, by creating richer service descriptions. The three notable efforts in this area are OWL-S, WSMO and SAWSDL.

2.1.1 OWL-S

The OWL-S coalition consists of a group of researchers from Stanford, SRI, Maryland, College Park, Carnegie Mellon and other institutes involved in Semantic Web research. The purpose of the OWL-S coalition, as discussed in Burstein et al. (5) and Martin et al. (33) was to define an upper ontology of Web services for semantically describing Web services. The motivation of OWL-S was having the ability to express services in a machine interpretable language, so that various aspects such as discovery, invocation and composition can be automated. The Web Ontology Language (OWL) has been chosen as the language for representing the ontology. OWL has theoretical underpinnings in description logics, which are a decidable subset of first order logic.

The ontology had three core parts: profile (what a service does), process (how to interact with the service) and grounding (how to invoke the service) (Martin et al. (34)). The profile describes the inputs, outputs, preconditions and results (previously called effects) of the service. The process model is used to specify ordering between various operations (called atomic processes in OWL-S) using standard workflow constructs such as sequence, split, join and choice. Finally, the grounding
associated the profile and the process model to WSDL file so that the services can actually be invoked.

### 2.1.2 Web Services Modeling Ontology - WSMO

The WSMO project on Semantic Web services includes a number of institutions primarily in Europe with DERI (Digital Enterprise Research Institute) being one of the lead institutions in the project (Zaremba et al. (83); Anicic et al. (1); Roman et al. (57)). It differs from OWL-S with regard to scope and underlying formalism. Unlike OWL-S, the WSMO project plans to create not only a specification, but also an architecture and a comprehensive set of tools to support the specification.

WSMO defines four main components: ontologies, Web services, goals and mediators. WSMO defines its own Web ontology language called Web Service Modeling Language (de Bruijn et al. (11)), which is based on F-logic. There are five different variants of WSML based on expressivity and scope of the language. They are - WSML-Core (intersection of Description Logic and Horn Logic), WSML-DL (extends WSML-Core to an expressive Description Logic), WSML-Flight (extends WSML-Core in the direction of Logic Programming), WSML-Rule (extends WSML-Flight to a fully-fledged Logic Programming language) and WSML-Full (unifies all WSML variants under a common First-Order umbrella).

Web services in WSMO are defined in terms of their capabilities using preconditions, postconditions, assumptions and effects (Stollberg and Norton (72); Feier et al. (13)). The distinction is that preconditions and postconditions represent conditions on the information space (e.g., electronically placing an order) before and after the services are executed, whereas the assumptions

---

1 http://deri.org
and effects are conditions on the state of world (e.g., item actually being shipped). In addition, a
Web service may have an interface and non-functional attributes. Goals represent users request for
services and are also defined in terms of desired capabilities using preconditions, postconditions,
asumptions and effects. Mediators allow linking heterogeneous components. There are four types
of mediators: ontology to ontology mediators (OOMediators), goal to goal mediators (GGMedi-
ators), Web service to goal mediators (WGMediators) and Web service to Web service mediators
(WWMediators).

2.1.3 METEOR-S

The METEOR-S research project is a follow-on to the METEOR (for “Managing End To End
OpeRations”) system presented in Krishnakumar and Sheth (27) focused on workflow manage-
ment and addressing issues of formal modeling, centralized as well as distributed scheduling and
execution (including exception handling, security, survivability, scalability and adaptation). The
work yielded two notable frameworks: 1) WebWork discussed by Miller et al. (42), a Web based
implementation and 2) ORBWork, a CORBA based implementation. The METEOR project ini-
tially started at BellCore in 1990 and was continued at the LSDIS lab until 1998. A commercial
spinoff, Infocosm, Inc. and the product METEOR EAppS (for Enterprise Application Suite) are
other notable accomplishments.

Adapting to the SOA and semantic Web evolution, METEOR evolved into METEOR-S where
S stands for services (or Service oriented Architecture) and semantics. It was largely carried out
at LSDIS Lab during later 1990s and 2006. One of the significant contributions of METEOR-
S research is the submission of WSDL-S specification as a W3C member submission, along with
IBM. In 2006, the W3C created a charter for the Semantic Annotation of Web Services (SAWSDL; www.w3.org/2002/ws/sawsdl), which used WSDL-S as its primary input. SAWSDL became a W3C candidate recommendation in January 2007.

Defining an environment that extends the principles of Semantic Web Services to the emerging notions of Web2.0 and the People Web is the research discussed in this dissertation. One of the key initial outcome is a microformat for annotating service descriptions in HTML called hREST and a faceted extension called SA-REST. Both hREST and SA-REST are in their early stages of research discussed in Sheth et al. (62). Figure 2.1 illustrates the various conceptual parts of the METEOR-S project.

![Figure 2.1: Overview of the various components of the METEOR-S framework.](image)

### 2.1.3.1 SAWSDL

Verma and Sheth (78) discuss the Semantic Annotation of WSDL and XML Schema (SAWDL), the W3C recommendation for adding semantics to service descriptions. SAWSDL leverages on the extensibility of WSDL to add semantic metadata. This is done using the modelreference extensibility attribute of WSDL. SAWSDL supports both data type annotation as well as interface annotations. Data type annotations in SAWSDL can be done either at the level of the document root (top-level)
or at the level of the document elements (bottom-level). The annotated XML schema can then
be mapped to a higher level semantic description. The transformation from XML to a semantic
model is defined as *lifting* and the transformation from a model to XML is defined as *lowering*.
The definitions of lifting and lowering, along with a systematic approach to data mediation is a
significant impact of SAWSDL.

The `modelreference` attribute can be also be attached to interface and operation elements in a
service. The functional semantics of what a service does, along with execution semantics related
to faults, are captured by annotating the interface, operation and fault elements.

### 2.1.3.2 Semantic Template

A semantic template is a model to capture the requirements of a requestor. This was first discussed
in Verma (75). Formally semantic templates are defined by:

**Definition 1** A semantic template $\psi$ is a collection of template terms $= \{\theta | \theta$ is a template term}\.

A template term $\theta = \{\omega, M_{\omega}, I_\omega, O_\omega, \pi_\omega, p_\omega, e_\omega\}$ is a 7-tuple with:

- $\omega$: the operation
- $M_{\omega}$: set of operation model references
- $I_\omega$: operation inputs and their model references
- $O_\omega$: operation outputs and their model references
- $\pi_\omega$: operation level term policy and the non-functional semantics
• $p_\omega$: operation precondition

• $e_\omega$: operation effect

The template term $\theta_s = \{\epsilon, \epsilon, \epsilon, \pi_s, \epsilon, \epsilon\}$ defining just the term policy defines semantic template wide term policies.

In the example described in Figure 2.2, $\omega$ is the operation PO_Order_HDD. The operation model reference $M^\omega_r$, is the concept PurchaseOrder in the functional ontology. $I_\omega$ is the input Order_HDD_Input along with the model reference PO_Input. $O_\omega$ is the output Order_HDD_Output along with the the model reference, PO_Output. This models the data and the functional requirements of the manufacturer.

Term policies can be specified either for individual operations as part of their template term or for a complete semantic template through the template term $\theta_s$. The term policy with assertions on SupplyTime and Security in Figure 2.2 is an example of a semantic template level policy. When a term policy is associated with an operation, the scope of the policy is limited to that operation. Such a term policy is called operation level term policy ($\pi_\omega$). In Figure 2.2, the term policy with an assertion on the UnitPrice is an example of operation level term policy. Together, the semantic template level and and the operation level term policy form the effective policy of an operation. Formally this is defined by:

**Definition 2** Given a semantic template $\psi = \{\theta_s, \theta_1, ..., \theta_n\}$ the effective policy $\pi_{eff}(\omega_1)$ of an operation $\omega_i (i = 1, ..., n)$ is defined as

$$\pi_{eff}(\omega_1) = \pi_s \land \pi_{\omega_1}$$
A term policy is defined as a set of assertions. Each assertion consists of a policy constraint and a model reference, which describes the assertion semantically. A policy constraint, finally, is a key-value pair describing the non-functional constraint. The policy constraint can be either a logical constraint specified in the assertion or imported from an external policy specification. The constraints on the $SupplyTime$ and $UnitPrice$ illustrated in Figure 2.2 are examples of policy constraints specified in the assertion.

**Definition 3** A term policy $\pi = \{\alpha | \alpha \text{ is an assertion} \}$ where $\alpha = (M_\alpha, C_\alpha)$.

A given policy constraint can be a quantitative constraint or a qualitative constraint. The constraint on the supply time is an example of quantitative constraint. The constraint on security is an example of qualitative constraint. The constraint on security also demonstrates how external policies can be imported. The effective policy for the PO_Order_HDD operation then includes the constraint on the unit price modeled on the operation level as well as the security and supply time constraint modeled on the semantic template level.

### 2.2 Discovery

One of the central contributions of this research is in the area of Semantic Web services discovery and policy matching. There has been a lot of work in the area of Semantic Web service discovery. Much of the earlier work in this area such as work by Paolucci *et al.* (51), Li and Horrocks (30) and Benatallah *et al.* (3), has primarily focussed on discovery based on the service inputs and outputs. A matching algorithm between services and requests is described in DAML-S (Paolucci *et al.* (51)). A match is determined by comparing all the outputs of a query with the outputs of
2.2. DISCOVERY

Figure 2.2: Semantic template modeling the requirements of the game manufacturer for ordering hard drives.

A service advertisement and the inputs of the advertisement with the inputs of the query. Various degrees of match are identified by computing the minimal distance between the concepts in a concept taxonomy. Li and Horrocks \((30)\) present a description logic based approach for service matching using DAML-S ontologies. A match is obtained by comparing the inputs and outputs of a service request and a service advertisement. In addition to the exact match, plugin and subsumption matches were discussed by Benatallah \textit{et al.} \((3)\). This is an approach to rewrite discovery requests into a form that can be expressed as a conjunction of Web services in a given DAML-S ontology.

The work mentioned in this dissertation takes a more holistic approach to service selection by comparing the data, operation, domain and non-functional semantics captured in service descriptions and policies. The above mentioned work assumes services to contain only one operation whereas in this work services are assumed to have multiple operations. Allowing requesters to specify their level of expected match in the request brings flexibility to the selection process that
is lacking in previous work. Sivashanmugam et al. (66) and Paolucci et al. (51) discuss techniques to represent semantic descriptions in UDDI registries. Paolucci et al. (51) first proposed an approach to importing the Semantic Web into UDDI by mapping DAML-S service profiles to UDDI records. Sivashanmugam et al. (66) proposed an approach that maps a semantically annotated service description to objects in the UDDI data model. Rather than mapping the semantic information to objects in the UDDI data model, this work actually creates objects that describe the semantic information and associate them with the objects in the UDDI data model. In addition to semantic information obtained from service description, the relationship between service interfaces along the dimensions of domain, operation and data is computed and stored. This allows seamless integration of semantic reasoning capabilities during discovery, without changing the objects in the UDDI data model. Zaremba et al. (83) presents a goal based approach for service discovery for collaboration establishment. When an agent wants to collaborate with someone, the objectives of the agent are captured as a goal. Other agents that can fulfill this goal are then discovered. This work again assumes services to contain just one operation. This is the key difference from the research described here.

There have been a handful of Web applications that facilitate the categorization and searching of Web APIs, of which ProgrammableWeb is the most popular. ProgrammableWeb allows users to add, tag, and describe APIs and mashups. ProgrammableWeb provides category-based API browsing and searching. Two other categorizations worth mentioning here are Tech-News and TechFreaks. Both of them offer very good API categorization, but do not provide search or other capabilities provided by ProgrammableWeb. The applications mentioned above support limited faceted search and do not have a ranking mechanism. The faceted search technique discussed in
2.3 Data Mediation

The third contribution of this work is a computable metric for measuring the ease of mediating between two schemas. Our research is inspired by and builds upon the past work in the areas of database, XML and ontology schema matching. There has not been any previous research to estimate the degree of human involvement in XML schema mediation.

Since the early work on federated databases by Sheth and Larson (64), interoperability among databases with heterogeneous schemas has been a well researched issue. Miller et al. (43) and Madhavan et al. (32) have discussed approaches to matching that transform heterogeneous models into a common model. Patil et al. (52) discusses an approach for automatic annotation by converting XML descriptions to schema graphs to facilitate better matching. Melnik (38) abstracts the mappings between models as high level operations independent of the underlying data model and the applications of interest. Melnik et al. (39) discusses an approach to computing the matching between two schemas based on similarity flooding. The approach presented by Melnik et al. (39) computes the similarity of an element, based on the similarity of the neighboring elements in a graph.

The various heterogeneities that can exist between two schemas is discussed by Kahsyap and Sheth (23). Nagarajan et al. (47) further extended this in the context of Web services, where message level heterogeneities between two interoperating Web services are studied in detail.
In the area of semantic Web services, the WSMO project (Zaremba et al. (83)) which coined the term Data Mediation, is most relevant to our work. Much of the focus of WSMO research has been in ontology mapping. Cimpian et al. (8) discusses a mediator based approach to address data and process mediation. Mocan et al. (44) present a formal model for ontology mapping. Mocan et al. (44) further discusses the role of the formal model in creating and expressing mappings in WSML, based on semantic relationships. Stollberg et al. (71) discusses an integrated model based on data level, functional level and process mediation for the Semantic Web with the main focus on services created using WSMO. Ontology matching and mapping is a vast area of research. In addition to the WSMO approach to ontology mediation, Calvanese et al. (6) and Mena et al. (40) among others also address this problem in different contexts. However, as discussed before, the measure of difficulty in data mediation (as captured by mediatability) and comprehensive evaluation with real world data as presented in this work was previously missing.

2.4 Composition

Automated Web service composition has received much attention from the academic community. The work in composition can be divided into two groups - 1) automated composition using AI planning, 2) finding services for predefined abstract processes. Our work falls in the second group and we now present an overview of previous research in these two areas.

One of the earliest works that proposed using AI planning for Web services was presented in Ponnekanti and Fox (54). It was based on the assumption that all services were data providing services and all the services were represented as horn rules of the form (inputs \(\rightarrow\) outputs). Based
2.4. COMPOSITION

on initial inputs and desired outputs the system would generate a composition of services by using a rule engine and forward chaining. A fundamental flaw with this work was that the authors neglected considering the preconditions and effects of the services, since it is possible that services with the same inputs and outputs may have exactly opposite effects (addition and subtraction services may have identically typed inputs and outputs but opposite effects). Another work McIlraith and Son (37) proposed using Golog to represent high level plans and then use a prolog reasoner to come with a concrete plan. There were constraints defined for suitable action in each state based on user preferences. Since Golog is a high level programming language, it was unclear how much automation was achieved beyond the selection of Web services based on the user defined constraints. Composting services using Hierarchical Task Network (HTN) planning was proposed in Wu et al. (80). HTN divides plans into sub-plans and recursively solves the sub-plans. The ordering has to be specified between sub-plans and it is hard to measure the amount of automation achieved. An approach that uses semantic relationships between preconditions and effects of services for automated composition is presented in Lin and Arpınar (31). Thus, most of the prominent work in Web service composition that uses planning techniques is either overly simplistic or the high complexity of representing the input to the planner (also called goal) makes the level, quality and value of the automation achieved unclear. In addition, the efforts to date have not presented a notion of global optimality or global constraints. In our opinion, while using AI planning for Web service composition represents an interesting research problem, due to the aforementioned problems, it may be quite a few years before it is applied in real world settings.

The second technique for Web service composition involves creating executable processes from abstract processes by using user defined constraints for finding the services for the processes.
In this technique, processes of any complexity can be created manually and then they can be configured by selecting the services on the basis of the constraints. Sivashanmugam et al. (65) presented an approach for representing the functionality of each partner service of Web processes (represented in BPEL) and using Semantic Web service discovery based on functional (what the service does) and non-functional (cost, time, etc.) requirements to find services for the process. This work only allowed static binding and did not have a notion of global optimality. An approach for representing inter-service dependencies in a Web process using OWL ontologies and accommodating them using a description logics reasoner was discussed in Verma et al. (77). Early results of combining the notion of optimality and generalizing inter-service dependencies to logical constraints were presented in van der Aalst and Basten (73).

Rao et al. (55) discuss the use of the GraphPlan algorithm to successfully generate a process. The notion of considering the interaction with the users improved the planning efficiency, but the approach discussed in Rao et al. (55) approach suffers from the extent of automation. Also this work, unlike ours does not consider the input/output message schema when generating the plan, though their system does give alert of missing message to the users. This is important because an operation’s precondition may be satisfied even when there is no suitable data for its input message. Another limitation of their work is that the only workflow pattern their system can generate is sequence, although the composite process may contain other patterns. As the reader may observe from the described in Chapter 7, other patterns such as loops are also frequently used.

Duan et al. (12) discuss using the pre and post-conditions of actions to do automatic synthesis of Web services. This is initiated by finding a backbone path. One weakness of their work is the assumption that task predicates are associated with ranks (positive integers). Their algorithm
gives priority to the tasks with higher rank. However, this is clearly invalid if the Web services are developed by independent organizations, which is the common case and the main reason leading to heterogeneities.

Pistore et al. (53) propose an approach to planning using model checking. They encode OWL-S process models as state transition systems and claim their approach can handle non-determinism, partial observability, and complex goals. However, their approach relies on the specification of OWL-S process models, i.e., the users need to specify the interaction between the operations. This may not be a realistic requirement in a real world scenario where multiple processes are implemented by different vendors.
Motivation: Web applications, the way you want them

The Web has developed into a platform for users to create and share content. Technologies such as blogging and tagging have made it possible for most people to create new content that is multi-modal (such as videos and pictures) and share it with other users. This idea of “read-write Web”, however is still to be realized in creation of Web applications. On one hand, creating Web applications has become as easy as integrating data from two services, while on the other, the complex details that one must understand before developing them makes it almost impossible for people to create their own application. These primary motivation for this dissertation is to explore the idea of read-write Web in the context of Web applications.

We will now describe an example of a personalized Web application. Services such as Google Maps\footnote{http://maps.google.com} and mobile applications such as AroundMe\footnote{http://www.tweakersoft.com/mobile/aroundme.html} allow users to search and find information such as fast food restaurants, near a particular location. However, they often involve a series of actions that a user has to perform, such as search and selection. Customizing these applications to perform certain actions, when they are loaded, can be an arduous task. This motivating scenario is inspired
by our personal experience when going on road trips.

John Doe is going on a road trip from Athens, GA to Hawthorne, NY via Dayton, OH and would like an application that gives him a list of McDonald’s near him at any point. Applications such as AroundMe can do this, but also require inputs from him. Giving these inputs would be difficult when he is driving. John would like to have his own personalized application that shows him a list of restaurants, when opened. In this dissertation, we identify the series of research issues involved in creating a platform for John, that allows him to create simple applications, with limited or no programming. We identify and address four important research problems in creating such a platform. We also mention the kind of semantics involved in addressing each problem. Figure 3.1 illustrates the motivating scenario and the contributions of this dissertation are highlighted.

Figure 3.1: Steps involved in creating Big Mac finder application
1. **Finding the right service** The first important challenge that we address is the searching and ranking services. In our example, we will have to first find services that will give us a list of restaurants near a location, map the locations on a map and convert physical addresses to latitude longitude pairs. Further, services use different modeling and representation approaches to describe data. Examples of these include JSON and XML. It will be easier to integrate the services that have the same data model. For example, if all our services used JSON, integration will be easier than if one of the services used XML and the others used JSON. An example search query for our scenario would be “*Find a mapping service that uses JSON*”. We discuss a two fold approach. In Chapter 4, we propose an algorithm for searching and ranking descriptive API documents on the Web. Discovery of conventional services described more formally in WSDL / SAWSDL is discussed in Chapter 5. Our algorithms and tooling will enable John Doe to find a service that gives a list of McDonald’s restaurants around a location, and a service that will geocode and reverse geocode. Finding the right service involves exploiting the data, functional, non-functional and execution semantics.

2. **It’s all about making the data flow:** Having found a service that will help him geocode and reverse geocode, John can now identify his location in coordinates and also in a physical location. The restaurant finder service takes in a latitude and longitude pair and returns a set of addresses. Finally, John needs to use a mapping service for getting directions. The mapping service requires a collection of lat-longs. How does he mediate between each and every data element in this composition? Does the ability to mediate have a bearing on his choice of a service? These questions are answered in Chapter 6, where we define a metric called *mediatability* that determines the ease of mediation between two services.
Mediatability exploits data semantics.

3. **Bringing services together:** The most important part of creating an application is the *wiring* of services. Wiring is the task of combining services in a manner that when the composition is executed, the desired output is created. Often one has to employ languages like javascript, WS-BPEL to create the wiring, making it hard for most Web users. In Chapter 7 we discuss a declarative approach for creating the wiring. More recently, we have developed a simple script that interacts with our middleware on the IPhone that can be used for wiring. Wiring and integration exploits data and functional semantics.

4. **Event handling:** Handling events, such as alerting the user when he is near an exit or when the service fails to return a result, is a critical part of the application. We elucidate an approach for identifying various events based on the declarative specification in Chapter 8. While this discussion is centered around SOAP and the Web services stack, one can easily adapt the methods to our example.
Of Faceted Search and Serviut Rank

Web application hybrids, popularly known as mashups, are created by integrating services on the Web using their APIs. Support for finding an API is currently provided by generic search engines or domain specific solutions such as Google and ProgrammableWeb. Shortcomings of both these solutions in terms of reliance on user tags make the task of identifying an API challenging. Since these APIs are described in HTML documents, it is essential to look beyond the boundaries of current approaches to Web service discovery that rely on formal descriptions. Given the descriptive nature of Web APIs, one can adopt the principles that drive search on the Web. Fundamental to search is document classification and vector based approaches form the backbone of this. Further, API search is rather a search on a specific type of resource. Thus identifying the features or facets of this resource, would make it possible to extend keyword based search into more specific, faceted search.

The second important problem this dissertation addresses in the context of API search is ranking. As demonstrated by Google earlier in the decade, ranking and relevance play an important role in determining the quality of search itself. A key metric in ranking resources on the Web, is the popularity of the resource itself. The Page Rank algorithm discussed by Page et. al in (49), uses the link structure found on the Web to determine the relative importance of a particular Web
resource. Borrowing this idea and further customizing it to the domain of APIs and mashups lays the foundation for servviut rank. The evaluation discussed at the end of this chapter compares faceted search with existing approaches for API search and demonstrates the usefulness of this approach.

4.1 Overview

The twin tasks of information retrieval, classification and indexing, form the core of the faceted search algorithm. APIs are classified and indexed based on the terms in the API and available user tags. Indexing, search and ranking are built on top of well known document classification algorithms. This Section presents a brief description of techniques used in classification, searching and ranking of APIs.

1. Defining Facets for Web API search: The ability to classify APIs based on their facets is key to faceted classification. Identifying and modeling the different facets for Web API search is the first step, accomplished by adopting the seven-step procedure for building a faceted classification, based on the work of Vickery (79) and Spiteri (69). The first step is to collect representative API samples to define the scope of the domain and the facets. This set of Web APIs was selected from a wide variety of domains, chosen for the richness of their description, which were manually inspected to isolate the concepts that they described.

We found that all APIs described the functionality provided, the messaging formats supported, the protocols and the programming languages they support (known as programming language bindings). Using this information, one can create four facets for Web API search:
4.1. OVERVIEW

1) Functionality, 2) Messaging formats, 3) Programming language bindings and 4) Protocol. Further, each API also had information about the domain (mapping, image search). The principles of citation order of facets and foci, described in Spiteri (69), can be applied towards organizing these facets. For example, the domain information found in APIs can be categorized under the functionality facet. Current domain based classification of APIs found at ProgrammableWeb, TechFreaks and TechMagazine were used as additional inputs in this step.

The end product is a taxonomy, snapshot in Figure 4.1, that models 62 different domains, 11 messaging types, 2 protocols and 7 programming language bindings and is 4 levels deep. Information about the APIs that were used to define categories is also preserved. The rest of the system uses this taxonomy and sample APIs in order to classify unseen APIs.

2. **Classification of APIs using Facets**: Traditional term vector based approaches were used for classifying and indexing APIs. Each category (node in the taxonomy) has two initial sets of term vectors created by considering a small set of representative APIs (manually classified in the categories) using bayesian techniques. One is a term vector built from terms spotted in an API and the other is built using user tags assigned to the APIs. Subsequently, when users add a new API, a term vector for the API is created by spotting entries that are in the API and in the category term vectors using a domain specific entity spotter. The API is then classified into relevant categories, based on the cosine similarity of the API and category term vectors. An API is classified into only those categories that pass a tuneable similarity threshold. Classification of APIs is discussed in detail in Section 4.2.

3. **Searching**: The system currently allows users to search on the following facets: 1) The
functionality of the API, 2) Messaging formats, 3) Protocol, and 4) Programming language bindings. The functionality is a mandatory facet for every query, while the others are optional. The search query is parsed to identify the services that match desired functionality using term vector similarity methods. Matched services in each category are grouped according to their facets before being passed on to a service ranking module.

4. **Ranking**: Services in each category are then ranked based on a utilization factor. The system calculates a service utilization score or serviut score for each API that is used to rank APIs. The serviut score for an API is calculated by the number of mashups that use a given API, the number of Mashups that are classified into the same functional categories as the API, the popularity of the Mashups based on user score and the Alexa traffic rank. Computation of the serviut score and ranking of APIs is discussed in Section 4.5. While services like ProgrammableWeb do offer a way to search for APIs, we believe that this work is one of the earliest to define a quantifiable metric for ranking APIs.

The core components of the system and the details of the faceted search algorithm are discussed in the subsequent sections of this chapter.

### 4.2 Indexing APIs

Much like traditional classification procedures, we first create the weighted term vectors for the categories under each of the primary facets. A term vector for a facet is the union of all term vectors of APIs classified in categories grouped under the facets. Facet tag vectors are simply the

---

1[^1]: http://alexa.com
union of tags that users have assigned to the APIs in categories grouped under the facets. For any new API that needs to be classified, we build a term vector for the API that consists of terms in the API that overlap with terms in facet term vectors and a tag vector that consists of tags assigned by the users. We use a term spotting technique that borrows from basic dictionary and edit-distance based spotting techniques \(^{(68)}\). Using term vectors of facets as dictionary entries, we use a variable window to spot an entity and their lexical variants (Levenshtein with a string similarity >0.9) in an API description. Finally, to decide which category an API is assigned to, we compute the vector cosine similarities between the API and category term and tag vectors. A tuneable threshold is used to pick the most relevant categories for classifying the API.

### 4.2.1 Creating Term and Tag Vectors

Typically, terms in a term vector have a weight assigned to them that is indicative of their discriminatory nature to a document or to a category. Variations of TF-IDF and the Naive Bayes method are the most commonly used term weights in document classification \(^{(59)}\). Here we explain how we assign weights to terms and tags in their vectors.
**Weighted Term Vectors:** A weighted term vector for a category is a collection of tuples where each tuple contains a term and a relevance factor. The relevance factor is a weight that is indicative of the association between the term and the category. The relevance of a term $t_i$ to a category $c_r$ is measured by computing the conditional probability $p(c_r|t_i)$. The conditional probability can be interpreted as the probability that an API containing the term $t_i$ belongs to the category $c_r$. We start by finding term frequencies of different terms across the APIs. Let $f_{t_i}$ be the frequency of a term $t_i$ across all the APIs in a given category. We can estimate the probability that any API in this category will contain this term as

$$p(t_i|c_r) = \frac{f_{t_i}}{\sum_j f_{t_j}}$$  \hspace{1cm} (4.1)

The probability of a category can be estimated as

$$p(c_r) = \frac{|A_r|}{\sum_j |A_j|}$$  \hspace{1cm} (4.2)

where, $|A_r|$ is the number of APIs in $c_r$ and $\sum_j |A_j|$ is the total number of APIs across all categories. Using Equations 4.1 and 4.2 in Bayes theorem would yield $P(c_r|t_i)$. The weighted term vector ($WT(c_r)$) for a category $c_r$ then is

$$WT(c_r) = \{ (t_i, p(c_r|t_i)) \}$$  \hspace{1cm} (4.3)

The term vector for a primary facet is created by computing the union of the term vectors of the categories classified under the primary facet in the taxonomy. The weight of a term in the
4.2. INDEXING APIs

The facet term vector is determined by the number of categories that are relevant to a term. A term that has fewer relevant categories has a higher weight than a term that has a large number of relevant categories. This is because, fewer categories indicate a stronger relevance.

\[
TT = \{(t_i, w_i) : t_i \in WT(c_r) \text{ and } w_i = \frac{1}{|C_S(t_i)|}\}
\]  
(4.4)

where \(C_S(t_i)\) is the set of relevant categories for a term \(t_i\), defined as

\[
C_S(t_i) = \{c_r : p(c_r | t_i) > 0\}
\]  
(4.5)

**Weighted Tag Vectors:** A weighted tag vector for a category is a collection of tuples where each tuple contains a tag and a relevance factor. The relevance factor is a weight that is indicative of the association between the tag and the category. Computing the relevance of a tag is similar to computing the weight of tags in a tag cloud. The relevance of a tag \(u_f\) to a domain \(c_r\) is computed as

\[
R(u_f, c_r) = \frac{f_{u_f}}{\sum f_{u_g}}
\]

The weighted tag vector \((WU(c_r))\) for a category \(c_r\) is defined as

\[
WU(c_r) = \{(u_f, R(u_f, c_r))\}
\]  
(4.6)

The approach to creating a the facet tag vector is similar to the one described for creating facet
term vectors. The tag vector of a facet is defined as,

\[ TU = \{ (u_i, w_i) : u_i \in WU(c_r) \text{ and } w_i = \frac{\sum R(u_i, c_r)}{m} \} \] (4.7)

, where \( m \) is the total number of categories.

4.2.1.1 Bootstrapping Term Vectors

The initial term vectors for facets were created using the representative APIs from programmableweb that were manually classified into categories; see Section 2.1. The APIs were chosen based on the richness of their description and their user popularity in programmableweb. The representative set consisted of 215 APIs across all categories. As in programmableweb, popular categories like Search and Mapping had more APIs than categories like Database and Weather. The method for creating the initial term vectors is determined by the number of distinct terms that can be used to describe a category. For the categories under the messaging formats, programming language bindings and protocol facets, the term vectors were created by manual inspection of the representative set of APIs. This was possible because the set of terms that can be used to describe them is rather sparse. Term vectors for the categories in the functionality facet were obtained from the initial set of APIs using Apache Lucene.
4.3 Classification

In this section, we discuss the classification of new APIs into categories defined in the taxonomy. To classify an API, we compute the similarity between the API and the categories in the taxonomy, using their weighted tag and term vectors.

4.3.0.2 Computing Similarity

To refresh, an API-Term vector \(API_T\) is the collection of the spotted terms in an API while an API-Tag vector \(API_U\) is created using user assigned tags for the API. To compute the similarity between an API and a category, we use the popular cosine similarity approach, although other techniques may well be applicable. We compute two similarity measures, one over term vectors of APIs and categories and other over tag vectors of the APIs and the categories.

\[
\alpha_T(API, c_r) = \frac{WT(c_r).API_T}{|WT(c_r)||API_T|} \tag{4.8}
\]

\[
\alpha_U(API, c_r) = \frac{WU(c_r).API_U}{|WU(c_r)||API_U|}
\]

Using the cosine similarity values, the overall similarity between an API and a category is calculated as the weighted sum of the similarity over terms and tags.

\[
\alpha(API, c_r) = w_t \alpha_T(API, c_r) + w_u \alpha_U(API, c_r) \tag{4.9}
\]

The similarity set of an API \(\psi(API)\) is the set of the similarity values between the API and all the categories. To eliminate the categories with weak similarity, we normalize using
\[
\alpha^N(c_r) = \alpha(c_r) - (AVG(\psi(API)) - \sigma(\psi(API)))
\] (4.10)

where \(AVG(\psi(API))\) is the average of the similarity values and \(\sigma(\psi(API))\) is the standard deviation. The set of similar categories is then

\[
sim_{cat}(API) = \{c_r : \alpha^N(c_r) > 0\}
\] (4.11)

**Example:** We illustrate our method for classification with an example. Consider the categories of Mapping, Geo and Photo and a mapping API. The \(\alpha_T\) and the \(\alpha_U\) values are shown in the table below. Taking \(w_t = 0.75\) and \(w_u = 0.25\) and using Equation 4.9, we get \(\alpha(API, Mapping) = 0.73\), \(\alpha(API, Geo) = 0.45\) and \(\alpha(API, Photo) = 0.075\). Using Equation 4.10, we get \(\alpha^N(Mapping) = 0.64\), \(\alpha^N(Geo) = 0.36\) and \(\alpha^N(Photo) = -0.01\). From Equation 4.11, \(sim_{cat}(API) = \{Mapping, Geo\}\).

### 4.4 Searching

Here, we describe our method for a faceted search for Web APIs. In addition to providing a search based on the functionality, the flexible faceted search also allows users to optionally specify requirements related to the other facets. To allow the specification of faceted queries, we adopt a command line approach to search. *Image Search; MType: XML,GData; Protocol: REST* is an example of a search command to search for *image search* services that use the GData or XML
messaging formats and the REST protocol. Each facet is identified by a facet operator, which if used has to be accompanied by a facet search value (called search facets). When the search command is executed, the search query (functional facet) and the search facets are identified by parsing the search command.

APIs for a given search query are identified by first identifying the categories that are relevant to the search query. To do this, we find the term that is the most similar lexical variant (Levenshtein with a string similarity >0.9) of the search query in the functional facet term vector. The terms for other search facets are identified in a similar manner. Using the lexical variants allows us to accommodate for typographical errors in the search command. Once the terms are identified for all facets, the categories belonging to the set of relevant categories for the term identified in the functional facet term vector are ranked in descending order of their similarity. The set of relevant categories is defined in Equation 4.5. APIs that are classified under each of the categories are selected and grouped. The APIs within each functional facet category are then grouped according to their fulfillment of the search facets. Serviut ranking, discussed in the next section, is used to rank the APIs according to their service utilization. Figure 4.2 illustrates an example execution for the image search command described above.

4.5 Serviut Rank

Once matched, APIs are ranked according to their relevance strengths. Here, we introduce service utilization (serviut) Rank, a method for rating APIs objectively, based on their utilization. In computing the serviut rank of an API, we adopt the widely accepted notion that traffic and re-use
are reliable indicators of the quality of a Web resource.

The serviut rank measure is inspired by the PageRank (Page et al. (49)) approach for ranking Web pages. Central to the PageRank approach are the incoming links to a page and the PageRank of the source of the links. The Mashups that use a given API are analogous to the incoming links and their rating is analogous to the PageRank of the source of the links. The greater the number of highly rated Mashups that use a given API, the higher the serviut rank of the API.

To compute the serviut rank, we first compute the serviut score for each API. The serviut score of an API depends on the following five factors: 1) The set of mashups that use the API \(M_a\), 2)
The set of mashups in the category \((M_c)\), 3) User assigned popularity of the mashups in \(M_a\) \((P(M_a))\) and \(M_c\) \((P(M_c))\), 4) User assigned popularity of the mashups in \(M_c\) and 5) Popularity of the mashups based on Web traffic. The serviut score has two components. The first component, user popularity score, is the derived using the number of Mashups and their user assigned popularity. The second component, traffic popularity, is derived using the Alexa rankings.

To calculate the user popularity score, we consider the set of mashups, their user assigned popularity scores and the number of user votes. For each Mashup that use a given API, we first calculate the normalized popularity score \((P_N(M_{ai}))\) using Equation \(4.12\).

\[
P_N(M_{ai}) = (P(M_{ai}) - \sigma(P(M_c)))
\]  

(4.12)

where, \(P(M_{ai})\) is the average user assigned popularity for this mashup and \(\sigma(P(M_c))\) is the standard deviation of the user assigned popularity values for all mashups in this category. The user popularity score for an API is calculated using the normalized popularity scores of the mashups that use the API.

\[
U_P(a) = \frac{V_{M_a}}{V_{M_c}} \sum_i P_N(M_{ai})
\]  

(4.13)

where, \(V(M_a)\) is the total number user votes for the mashups that use this API and \(V(M_c)\) is the total number votes for all mashups in this category.

To calculate the relative traffic popularity of mashups, we first obtain the rank of all the mashups in a given category using Alexa Web service. Since the Alexa rank is calculated for the Web in general, and we are interested only in the traffic popularity of a mashup relative to other mashups in the same category, we first normalize the Alexa rank. The normalized traffic popularity
of a mashup $M_{ai} \in M_a$ is given by

$$NT(M_{ai}) = \frac{T_H(M)}{T_R(M_{ai})}$$

(4.14)

where $T_R(M_{ai})$ is the traffic rank of the mashup $M_{ai}$ and $T_H(M)$ is the highest traffic rank for any mashup in $M$.

Using the normalized traffic popularity defined above and the user popularity score defined in 4.13, we define the serviut score of an API as,

$$serviut(a) = w_t \sum_i NT(M_{ai}) n w_u U_P(a)$$

(4.15)

Serviut rank is a ranking of the APIs based on their serviut scores.

**Example** We illustrate the serviut rank method with an example. Consider API1 and API2 illustrated in Figure 4.3. For the purposes of this example, we assume that both of them are the only APIs in a category $c_r$. From Figure 4.3 $M_c = \{Mashup1, Mashup2, ..., Mashup7\}$, $M_{API1} = \{Mashup1, Mashup2, Mashup3, Mashup4\}$ and $M_{API1} = \{Mashup5, Mashup6, Mashup7\}$. The normalized popularity score, calculated using Equation 4.12 is illustrated in Figure 4.4(a). The normalized traffic score, computed using 4.14 is illustrated in Figure 4.4(b).

Assuming the weight for the traffic rank to be 0.75 and user popularity to be 0.25 in Equation 4.15 the serviut(API1)= 2.51. Similarly, serviut(API2)= 1.14. Using serviut rank, API1 would be ranked ahead of API2. Even though Mashups creating using API2 attract higher Web traffic, the fewer number of user votes and the poorer user popularity attributed to the lower serviut score of API2. This example also illustrates the importance and significance of the social process in serviut.
4.6 Evaluation

In this Section we present the empirical evaluations of our method for classifying and ranking Web APIs. The data set for the evaluation was obtained by crawling the APIs in programmableWeb. The APIs were then classified using the method discussed in 4.3. The objective of our empirical evaluations is three fold: 1. Evaluate the accuracy of classification through a user study; 2. Evaluate the accuracy of our approach using conventional precision and recall measures; and 3. Evaluate the effectiveness of serviut rank using user evaluation. For the first and third experiments we use the widely accepted Cohen’s Kappa statistical measure of inter-rater reliability Cohen (9).
### Table 4.2: Precision and Recall of ApiHut

<table>
<thead>
<tr>
<th>Query</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query1</td>
<td>0.89</td>
<td>0.75</td>
</tr>
<tr>
<td>Query2</td>
<td>0.83</td>
<td>0.69</td>
</tr>
<tr>
<td>Query3</td>
<td>0.54</td>
<td>0.71</td>
</tr>
<tr>
<td>Query4</td>
<td>0.82</td>
<td>0.21</td>
</tr>
</tbody>
</table>

#### 4.6.1 Accuracy of Classification

To study the accuracy of classification, we presented fifteen users with eleven Web APIs and a set of six categories. The users were selected across different levels of technical capabilities ranging from occasional programmers to expert developers. Users were asked to rate the categories as most similar, moderately similar and negligibly similar for each API. Categories (in the similarity set of an API obtained by our classification method) were classified based on a threshold defined using the average of the similarity values and their standard deviation. Cohen’s measure was then used to calculate the level of agreement between ratings assigned by users and those calculated by our method. The agreement for an API is the average of the agreement between the user rating and the rating calculated by our method for that API. The overall agreement between the system and the set of users is the average agreement across all APIs. Using this measure, the average agreement between the system and the set of users was 0.627. Upon further inspection of the agreement score, we found that when the system classified a category as most-similar, 40% of the users agreed with the system. For moderately-similar classification, the agreement was 47%. However, nearly 87% of the users agreed with the system when a category was classified as negligibly-similar, thereby demonstrating the effectiveness of our normalization approach, defined in Equation 4.10.
4.6. EVALUATION

4.6.2 Precision and Recall

Our second experiment has two parts: 1) Comparing the precision of our system (ApiHut) with ProgrammableWeb and Google. 2) Measuring the precision and recall metrics of our system.

To compare the precision of the results returned by ApiHut, ProgrammableWeb and Google, we used the following queries: 1) Map; Protocol: REST, 2) Video Search; message Type: XML, 3) Photo Editing; Protocol: REST and 4) Geocoding; message Type: XML. Since Google is a general purpose search engine, it is not reasonable to expect it to support facets. Hence, we appended the functional facet (Map, Video Search, Photo Editing and Geocoding) of each query with web service api to create the queries for google. We used the advanced search feature of ProgrammableWeb that allows for searching based on additional parameters. However, the message format and protocol facets are collectively called protocol in ProgrammableWeb. This limits the search option to either a messaging format or a protocol. The results of the first part of the experiment are illustrated in table 4.1. In this experiment, we only considered the top 50 results returned by Google. A closer inspection of Google’s result revealed that pages belonging same API’s description occurred multiple times. For example, Google Maps API appears nearly 15 times in the 30 results, because of the Pagerank. This skew in results validates our claim that a domain specific ranking approach is needed to rank Web APIs.

The second part of this experiment measures the precision and recall metrics. Since there is no way to determine the actual number of services that should be returned for ProgrammableWeb, we do not estimate its recall. To measure the recall of our system, users classified 100 services into 4 categories. The user classification was used as the gold standard. The results obtained by using the same set of queries described above were compared with the gold standard. The result of
4.6. EVALUATION

August 25, 2009

Our experiment is illustrated in Table 4.2. The average recall values were around 70%. The recall value for the geo-domain, however, was very low (21%). Upon further analysis, it was found that a large number of APIs were either poorly described or the vocabulary was inconsistent, leading to poor quality of facet term vectors. One potential approach to alleviate this problem is by manual inspection and correction of the term vectors.

4.6.3 Effectiveness of Ranking

In this experiment we study the effectiveness of our ranking methodology and the adequacy of the serviu rank as an approach to rank API’s. Since ranking is very personal and subjective, to study the effectiveness of the ranking methodology, users were asked to rank the results of seven search queries. The Cohen’s kappa measure was then calculated between the ranks assigned by users and the results of the serviu rank. The average kappa score of 0.83 indicates a strong agreement in the ranks assigned by the users and the serviu rank.

To measure the adequacy of serviu rank as an approach to rank APIs, we asked 40 users to answer a short questionnaire. The questionnaire is available online at ApiHut survey. The users were asked to respond to the following questions:

1. Is user popularity a sufficient measure for ranking an API?

2. Are the metrics used in serviu rank representative of the popularity of an API?

3. Which one of user popularity and traffic popularity is more indicative of the service utilization?

2http://apihut.com/survey
4.6. EVALUATION

August 25, 2009

4. Is serviut rank, by itself a sufficient measure for ranking APIs?

Almost 93% of the users said that user popularity in itself cannot be used for ranking APIs. This vindicates our belief that while user participation is a very important factor in ranking, it cannot be the only factor. 98% of the respondents agreed that the metrics used by serviut rank are representative of the popularity of an API. Asked to choose between user popularity and the traffic popularity metrics, all of the respondents said that the traffic popularity is a more important metric. To the last question, 40% of the felt that serviut rank was sufficient to rank APIs, while the rest said that other metrics such as facet fulfillment must be considered into ranking. This evaluation demonstrates that serviut rank as a measure is very useful in ranking APIs.
Dynamism and autonomy to business process development and deployment is one of the significant promises of Services Oriented Architecture. Many applications that support operations in today’s global organizations can benefit from autonomous processes that can react to the changes in the environments in which they operate in. A registry that can facilitate the matching of the requirements of a process with services that can satisfy them, is a key enabler. To do this, the registry must be capable of understanding both the needs of the process as well as the capabilities of the services that may potentially satisfy them. Creating such a registry involves the following challenges

- To capture the semantics that would enable the understanding of requirements and guarantees

- Addressing scalability concerns involving semantic data processing in the context of Web service registries and,

- To avoid the concerns related to prior registry implementations such as UDDI. that adopts relevant W3C standards for service description (SAWSDL and WS-Policy)

In this chapter, we discuss the architecture and implementation of such a registry framework. Evaluation of the framework is discussed and this approach is compared to existing approaches for
5.1 Motivation

In this section, we illustrate the importance of the registry in realizing self-configuration and adaptation of business processes. Our scenario is a simplified representation of the production line of a computer manufacturer. We have based our example on real-world scenarios described in Kapuscinski et al. (24), and (IBM). Dell outlined the importance of dynamism in inventory management to the overall optimality of a supply chain process in Kapuscinski et al. (24). The second scenario showed the role of the registry in achieving dynamism in a supply chain process is illustrated in (IBM).

5.1.1 Motivating Scenario

Consider the production line in the supply chain of a computer manufacturer. When an order is placed, the manufacturer procures the components needed to fulfill the order from the inventory. The product is then assembled, tested, and shipped. Two activities that exhibit the need for self-configuration and adaptation are:

- **Adding a new product line**: The manufacturer would like to add a new line of less expensive laptops. Once the various components of product are identified, the manufacturer has to select the suppliers from its internal registry. The manufacturer requires the suppliers to provide two operations: one for ordering the components and the other for canceling the
5.1. MOTIVATION

order, should the necessity arise. In addition to the two functional requirements, the manufacturer also has non-functional requirements on cost of the component and on the penalty incurred for canceling an order. The selection of partners that meet these requirements is critical to achieving the desired levels of optimality of this process. It is therefore highly desirable that the registry framework be able to select suppliers that meet the functional and non-functional requirements of the manufacturer. This scenario illustrates the importance of a registry framework in realizing self-configuration.

- **Inventory Management**: As illustrated in the Kapuscinski et al. (24), manufacturers prefer not to hold large volume in inventory because technology changes rapidly. However, an unexpected rise in demand coupled with a delay in receiving required components can create a shortage of inventory. In such circumstances, the manufacturer must find an alternative supplier to fulfill its requirements. Ericsson’s well-chronicled failure to react quickly when a fire disrupted its supply chain clearly shows the importance of being able to find alternative suppliers (Sheffi (60)). This ability, again, is affected by the ability of the registry framework to select suppliers that fulfill the manufacturer’s requirement. The inability of the registry in this regard hampers the ability of the manufacturer to adapt.

These scenarios underline the importance of the registry to achieving self-configuration and adaptation. We now outline the challenges that need to be addressed in order to create such a registry infrastructure.
5.1.2 Registry as the centerpiece

Designing an effective registry framework presents several research challenges, the most important of which have been outlined here. Although these challenges have been identified and addressed with varying depth and success, we advocate an architecture that uses registry as the centerpiece and addresses the challenges in comprehensive manner.

**Modeling semantics in the registry:** The registry must support languages that allow service providers to express their functional and non-functional capabilities. The registry could then select services based on the semantic information that is described.

**Support for publishing non-functional capabilities:** For the registry to select services based on the functional and non-functional requirements, it must allow service providers to publish their non-functional capabilities.

**Flexibility:** The criteria for selection must not be too restrictive, since it may be very difficult to find services that exactly match the requirements. The requester must be able to specify the expected level of match for the different aspects of the request. For example, a requester can specify that an exact match is needed with respect to the operation while a *sufficiently similar* match would suffice for the input and output parameters.

**Ranking:** The registry must be able rank the set of selected services based the functional and non-functional match.

**Descriptive requirements:** Finally, the requester must be able to create requirements that describe his functional and non-functional requirements. Such a requirement must also allow the requestor to specify the level of match expected for the different elements of the requirement.

SAWSDL provides an elegant way to add semantic annotations to a Web service description.
in WSDL. By supporting publication of services described in SAWSDL we can address the first challenge. Semantic templates discussed in Section 2.1.3.2 offer a way to create rich service requirements, and their use would allow a requester to create descriptive requirements. We briefly discuss SAWSDL and semantic templates.

**5.2 Service Models**

**5.2.0.1 SAWSDL**

SAWSDL is a W3C recommendation for adding semantic annotations to WSDL (Verma and Sheth (78)) and is discussed in detail in Section 2.1.3.1. A semantic template captures the functional and non-functional requirements of a service requestor. Section 2.1.3.2 presents a detailed discussion of semantic templates. In our registry, services that are published are described in SAWSDL and discovery requests are made using semantic templates.

**5.3 Business assertions in WS-Policy**

The ability to describe the functional and non-functional properties of a service is a first step toward selecting services that fulfill the functional and the non-functional requirements. The functional properties can be described using SAWSDL and semantic templates. As illustrated in Section 7.1, the non-functional properties include business rules that capture business level capabilities or requirements. In this section, we discuss our approach to describe business rules within the WS-Policy specification. WS-Policy is a W3 specification for representing the policies of a Web
service. The WS-Policy specification defines a policy as a collection of alternatives; each policy alternative is a collection of assertions. WS-Policy provides a flexible grammar for describing the non-functional capabilities and requirements of a Web service. Leveraging this flexibility we define a new class of assertions called business assertions to describe business rules.

Each business assertion describes a business rule. Formally a business assertion is defined as,

$$ \alpha = (C, O, V, U_V, T) \quad (5.1) $$

where $C$ is the assertion term, $O$ is the assertion operator, $V$ is the assertion value, $U_V$ is the assertion unit in which the value is represented and $T$ is the assertion type. The assertion term refers to a business concept in the semantic meta-model on which the rule is defined. The assertion value captures the value associated with the business concept in the rule. The value can either be numeric or non-numeric. The assertion operator defines the relation between the assertion term and the assertion value. The operator can be one of Equals, LessThan, GreaterThan and NotEqual in the case of numeric values. In case of non-numeric values, the operator can be one of Equals or

1http://www.w3.org/Submission/WS-Policy/
Contains. The units in which the value is represented is described in the unit term in the business assertion. If the assertion is non-numeric, then the assertion unit is left empty. The assertion type defines if the assertion is a requirement or a guarantee. Business assertions can be combined to create policy alternatives using the *All* and *ExactlyOne* operators. The normal form of a WS-Policy is a disjunction of alternatives and conjunction of all assertions in an alternative. Figure 5.1 illustrates a nesting of the operators.

We illustrate the notion of business assertion with an example. Consider a business rule of a supplier. The supplier is capable of producing the product in less than 3 days and provides alternative shipping methods. The supplier can deliver the product in one day or in three days. The policy describing this business rule is illustrated in Figure 5.1.

### 5.4 System Design

In this section, we discuss the design of a registry middleware that addresses the challenges outlined in Section 5.1.2. The design is illustrated in Figure 5.2. The system is composed of a storage component that contains the objects in the registry data model and a set of middleware components that are responsible for service publication and selection.

We adopt an evolutionary approach for designing the system. Rather proposing an architecture that is completely de novo we extend the data model in current registry architectures to store the additional information and add middleware components that enhance discovery and selection. This is one of the first attempts to integrate semantic reasoning seamlessly into a UDDI based registry. Current approaches to supporting semantic descriptions in UDDI describe the additional se-
5.4. SYSTEM DESIGN

Semantic information within existing objects in the data model (Paolucci et al. (50), Sivashanmugam et al. (66)). By enhancing the data model with objects that describe the additional semantic information, we adopt a nonintrusive approach to supporting semantic descriptions. The evolutionary nature of the design makes it easier for adoption while the nonintrusive approach ensures that the current abilities of the registry are not affected by the extensions. Further, the registry employs a hierarchical and independent matching technique that allows the matching process to be parallelized, thus making it efficient. Figure 5.3 gives an overview of the matching process, discussed later in the chapter.

Figure 5.2: (a) System Architecture (b) Extending a UDDI based registry

Figure 5.3: Hierarchical matching of the SEMRE registry
5.4.1 Components in the registry middleware

The middleware components of the registry are responsible for service publication and selection. The main components that are responsible for publication are the publication coordinator and the publication. The publication coordinator coordinates the various tasks during publication. These tasks include parsing of the service description, capturing the semantic annotations of various service elements and fetching and storage of semantic meta-models. The publication is responsible for publishing the service along with the semantic descriptions in the registry. During this process, it also coordinates the calculation of interface relationships between the interface that is being published and those that are already in the registry. The selection coordinator is responsible for coordinating the various tasks involved in service selection such as discovering services that fulfill the functional requirements of the request, selecting services that fulfill the non-functional requirements of the request, and ranking the set of selected services.

5.4.2 Data Model

When a service is published, data models in registries such as UDDI have objects to store information about the publisher of a service, the technical information about the service interface and the information necessary to invoke the service. A service has a service interface element that contains the operations and the data elements of a service. The information regarding the message formats and protocol details that are essential to invoke a service are described in the binding element of a service. In other words, the binding element contains the necessary information to invoke an implementation of the service interface. The technical information described in the the service interface element is stored in service interface info object. Each interface element in the service
interface info object has uniquely identifiable service interface identifier. Service interface info is the same as the TModel object in the UDDI data model, and the TModel Key is the service interface identifier. The information necessary to invoke the service is stored in service implementation info object along with its service interface identifier. Each binding information in the service implementation info object has a unique binding identifier. The service implementation info is same as the binding template object in the UDDI data model.

In order to support semantic descriptions and service selection based on functional and non-functional properties, we to enhance the data model with semantic interface link, interface relationship and business assertions objects. The semantic interface link object describes the link between the interface information of a service and the semantic annotations on the interface, its operations and their inputs and outputs. It contains a reference to the service interface identifier of a service interface and the semantic annotations of the interface, operations and data elements of the interface.

The interface relationship object is an exhaustive collection of semantic relationships between the domain, operations and data elements of two service interfaces. The interface relationship object contains the relationships between all pairs of semantic interface signatures and along with their respective service interface identifiers. The non-functional properties of a service are stored in the business assertions object. Since each implementation can have its own non-functional requirements, the business assertions object also stores the binding identifier of the service.

In addition to this, for efficiency and optimization the registry also has provisions for storing the various semantic meta-models that are referenced in the semantic annotations. These are stored in the semantic models object.
5.5. Publication and Selection

In this section, we present the publication and selection algorithms used in our framework. First we define a few concepts that would be used in our discussion of the algorithms.

5.5.1 Definitions

**Definition 4 Semantic Interface Signature:** The semantic interface signature of a service is a two tuple that captures the semantic annotations of the domain and the operation information of a service interface (SI) or a template term in a semantic template. Formally, 

\[ S = (M^S, \theta) \]

where \( M^S \) is the semantic annotation on the interface or the template term element that describes the domain information and \( \theta \) is a collection of operations defined in the interface or in the template term. Each operation \( \theta_i \in \theta \) is a tuple consisting of the annotation on the operation and the data elements. Formally, 

\[ \theta_i = (M^{\theta_i}, D^{\theta_i}) \]

where \( M^{\theta_i} \) is a semantic annotation and each data element \( D^{\theta_i} \) consists of the annotations on the input and the output of the operation. Formally, \( D^{\theta_i} = (M^{I_{\theta_i}}, M^{O_{\theta_i}}) \) where \( M^{I_{\theta_i}} \) is the semantic annotation on the input and \( M^{O_{\theta_i}} \) is the semantic annotation on the output.

**Definition 5 Interface Relationship:** Interface relationship captures the relationship between two semantic interface signatures over a semantic meta-model. Formally it is defined as,
\[ I_R(S_i, S_j) = (R^S, R^\theta, R^D) \]

where

\[ S_i = (M^{S_i}, \theta_i) \]

\[ S_j = (M^{S_j}, \theta_j) \]

\( S_i \) and \( S_j \) are the two interface signatures, \( R^S \) is the relationship over their interface elements, \( R^\theta \) is the relationship over their set of operations and \( R^D \) is the relationship between their data elements.

The relationship between interface signatures is calculated based on the relationship between the semantic annotations on their entities in the semantic meta-model. We now define the relationships on the domain, operation, input and output entities of a semantic interface signature. For the definitions that follow, we consider semantic interface signatures:

\[ S_i = (M^{S_i}, \theta_i) \]

\[ S_j = (M^{S_j}, \theta_j) \]

- Relationship over the interface element \( (R^S) \): \( R^S(S_i, S_j) \) is defined on the semantic annotations on the interface element of a semantic template signature. The relationships over the interface element are defined as:
  
  - **Equivalent over interface** \( (\equiv_S) \): \( S_i \equiv_S S_j \) if \( M^{S_i} \equiv M^{S_j} \) in the semantic meta-model.
  
  - **Generalized-Similar over interface** \( (\sqsupseteq_S) \): \( S_i \sqsupseteq_S S_j \) if \( M^{S_i} \sqsupseteq M^{S_j} \) in the semantic meta-model.

  - **Subsumption-Similar over interface** \( (\sqsubseteq_S) \): \( S_i \sqsubseteq_S S_j \) if \( M^{S_i} \sqsubseteq M^{S_j} \) in the semantic meta-model.

\( R^S(S_i, S_j) \) exists between two semantic interface signatures, if any of the above relationships can be defined.
5.5. PUBLICATION AND SELECTION

- Relationship over operations ($R^θ$): $R^θ(S_i, S_j)$ is defined on the semantic annotation on the operation elements in a semantic interface signature. In order to define the relationship over the operation elements between two semantic interface signatures, there must exist a relationship between their corresponding interface elements. If a relationship exists between the interface elements, we define the relationship over the set of operations as:

  - **Equivalent over operations** ($≡_θ$): $S_i ≡_θ S_j$, if $∀θ_{i_m} ∈ θ_i, ∃θ_{j_n} ∈ θ_j | M^{θ_{i_m}} ≡ M^{θ_{j_n}}$ in the semantic meta-model and $∀θ_{j_n} ∈ θ_j, ∃θ_{i_m} ∈ θ_i | M^{θ_{i_m}} ≡ M^{θ_{j_n}}$. The $≡_θ$ relation is left total and surjective.

  - **Generalized-Similar over operations** ($⊒_θ$): $S_i ⊒_θ S_j$, if $∀θ_{j_n} ∈ θ_j, ∃θ_{i_m} ∈ θ_i | M^{θ_{i_m}} ⊒ M^{θ_{j_n}}$ in the semantic meta-model. The $⊒_θ$ relation is surjective.

  - **Subsumption-Similar over operations** ($⊑_θ$): $S_i ⊑_θ S_j$, if $∀θ_{j_n} ∈ θ_j, ∃θ_{i_m} ∈ θ_i | M^{θ_{i_m}} ⊑ M^{θ_{j_n}}$ in the semantic meta-model. The $⊑_θ$ relation is surjective.

$R^θ(S_i, S_j)$ exists between two semantic interface signatures, if any of the above relationships can be defined.

- Relationship over data elements ($R^D$): $R^D(S_i, S_j)$ is defined on the semantic annotation on the data elements in a semantic interface signature. In order to define the relationship over the data elements between two semantic interface signatures, there must exist a relationship between their corresponding interface and operation elements. We define the relationship over the set of data elements as:

  - **Equivalent over data elements** ($≡_D$): The equivalent over data elements between two service interfaces is defined as, $S_i ≡_D S_j$, if $M^I_{θ_i} ≡ M^I_{θ_j}$ and $M^O_{θ_i} ≡ M^O_{θ_j}$.
5.5. **PUBLICATION AND SELECTION**

### August 25, 2009

- **Acceptable-Similar over data elements** $(\gamma_D)$: $S_i \gamma_D S_j$, if $M^I_{\theta_i} \subseteq M^I_{\theta_j}$ and $M^O_{\theta_i} \supseteq M^O_{\theta_j}$.

- **Partial-Similar over data elements** $(\lambda_D)$: $S_i \lambda_D S_j$, if $M^I_{\theta_i} \supseteq M^I_{\theta_j}$ and $M^O_{\theta_i} \subseteq M^O_{\theta_j}$.

$R^D(S_i, S_j)$ exists between two semantic interface signatures, if any of the above relationships can be defined.

**Definition 6** Semantic interface signatures $S_i$ and $S_j$ are equivalent if they are equivalent over interface, operations and data elements.

\[
S_i \equiv S_j \text{ if } (S_i \equiv_S S_j) \land (S_i \equiv_\theta S_j) \land (S_i \equiv_D S_j).
\]

The algorithms for publication and discovery can now be defined based on the above definitions.

#### 5.5.2 Publication

Computing interface relationship is one of the key components of our publication algorithm. We first discuss algorithm for calculating the interface relationship between two semantic interface signatures. The relationship over interface between two semantic interface signatures is computed by first identifying the relationship over the interface element. The semantic annotation on the interface element describes the domain of the service interface. If the relationship over interface doesn’t exist between two semantic interface signatures, it very unlikely that the relationship over operations and data, even if they exist would be useful in publication or discovery. Hence, if there is no relationship between the interface elements we do not define any relationship between the two signatures. Since relationship over interface is defined on two concepts, unlike relationships over operation and data which are defined sets, computing relationship over interface is the least
expensive operation. We identify the relationship over the set of operations, if there exists one.

Finally, we identify the relationship over data elements. It is not mandatory for the relationship over data elements to exist. The algorithm for computing the semantic interface is shown below.

**Algorithm 1 Compute Interface Relationship**

\[
R^S = R^S(S_1, S_2)  
\]

if \(R^S\) exists then

\[
R^\theta = R^\theta(S_1, S_2)  
\]

if \(R^\theta\) exists then

\[
R^D = R^D(S_1, S_2)  
\]

end if

end if

\[
I_R(S_1, S_2) = (R^S, R^\theta, R^D)  
\]

When this service is published, we first create the semantic interface signature of the service. From the semantic interface link data structure in the registry, we identify the semantic interface signature that is equivalent to that of the vendor’s service. If such an interface signature already exists, we identify the service interface info information referenced in the semantic interface link data structure. The implementation information of the vendor’s service is then added to the Service Implementation Info data structure along with the service interface info information identified. If there is no semantic interface signature in the semantic link data structure that is equivalent to that of the vendor’s service, we add the interface information of the service to the service interface info data structure. This information along with the semantic interface signature is then added to the semantic interface link data structure. The relationships between the semantic data link of the vendor’s service interface and the semantic interface links of all the other service interfaces that are already published is then calculated using the semantic interface signatures defined in the semantic interface link data structure. These relationships are then added to the interface relationship data structure. The non-functional properties of the service including the cost information are then
added to the business assertions data structure. The algorithm for service publication is listed below.

Algorithm 2 Publish Service -1

```
publish-service[Semantically annotated service description (SD) and its policy (π)][h]

S = Semantic interface signature of the of the service interface (SI) defined in SD.
C_S = Collection of all the semantic interface signatures published in the registry.

if (∃S' ∈ C_S | S ≡ S') then
    PublishService(SD)
    Add the SI to the service interface object
    Add SI and S to the semantic interface links object
    for S' ∈ C_S do
        Calculate (I_R(S, S'))
        if R^ε(S, S') and R^θ(S, S') exists then
            Add I_R(S, S') to the interface relationship info object
        end if
    end for
end if
```

If one were to provide just a service interface instead of semantically annotated service descriptions and policies, then that service interface is published in the registry. In that case the Publish Service method is not executed.

We now discuss the algorithm for service selection.

5.5.3 Selection

The main motivation behind creating the registry middleware is to enable selection of services that fulfill the functional and non-functional requirements of a requestor. In addition to enabling service selection, the registry must also provide the requestor with the flexibility to specify the expected match level for there requirements. The service selection algorithm discussed below takes a semantic template as the input. Service requesters describe their functional and non-functional
5.5. PUBLICATION AND SELECTION

Algorithm 3 Publish Service-2

Add binding information to the service implementation object

\[ \pi = \text{Compute the normal form of the service policy} \]

\[ \text{for Alternative } A \in \pi \text{ do} \]

\[ \text{Add } A \text{ to the business assertions object} \]

\[ \text{end for} \]

requirements along with the desired level of match for each requirement in the semantic template. The desired level of match for the domain and data elements can be one of equivalent, Generalization-Similar or subsumption-similar. The level of match over operations can be one of equivalent, Generalization-Similar or subsumption-similar. For non-functional requirements can be specified as either mandatory or optional. The ordering of the match level for \( R^S \) relationships is \( \equiv_S > \sqsupseteq_S > \sqsubseteq_S \). The ordering of the level of match for the \( R^\theta \) relationship is \( \equiv_\theta > \sqsupseteq_\theta > \sqsubseteq_\theta \). The ordering of the level of match for \( R^D \) relationships is \( \equiv_D > \gamma_D > \lambda_D \).

We first discuss our approach to matching the functional requirements of a requester. From the set of services that match the functional requirements, we identify the set of services that match the non-functional requirements. Finally, the set of services that fulfill both the functional and non-functional services is ranked.

5.5.3.1 Matching Functional Requirements

The first step in service selection is to calculate the semantic interface signature \( (S) \), of the semantic template that describes the requirement. Once the semantic interface signature is calculated, we identify the set of domain, operation, and data relationships which must exist between a semantic interface signature \( S' \) and \( S \) for \( S' \) to meet the requestor’s functional requirements. We call this the fulfillment set. Let \( R^S_{E}, R^\theta_{E} \) and \( R^D_{E} \) be the expected level of match over domain, operations
and data elements respectively. The fulfillment set over domain \( (R_S^F) \) consists of all domain relationships that are stronger than the expected level of match. The fulfillment set over operations \( (R^\theta_F) \) and data elements \( (R^D_F) \) are similarly defined. For \( S' \) to fulfill the requirements of the service \( I_R(S, S') \) must belong to the cartesian product of \( R_S^F \), \( R^\theta_F \) and \( R^D_F \) given by

\[
R_S^F \times R^\theta_F \times R^D_F
\]

The next step in selection is to identify all semantic interface signatures \( S' \) such that \( I_R(S, S') \in R_S^F \times R^\theta_F \times R^D_F \). To do this, we use the interface relationship object. We here recall that the interface relationship object is an exhaustive collection of interface relationships between all pairs of semantic interface signatures. We identify all \( S' \) such that the interface relationship \( I_R(S, S') \) in the interface relationship object is in the cartesian product defined above.

Once all \( S' \) are identified, the service interface for each \( S' \) can be obtained from the semantic interface link object. Services that fulfill the functional requirements can be identified from the service implementation info object using the service interface. From this set of services that match the functional requirements, we present our approach to identify to all services that match the non-functional requirements in Section 5.5.3.2. The algorithm for finding services that match the functional requirements in given the algorithm below.

5.5.3.2 Matching Non-Functional Requirements:

Non-functional requirements are captured as operation, term and template policies in the semantic template discussed in Section 2.1.3.2. The effective policy \( (\pi_\theta) \) of an operation in the semantic
Algorithm 4 Select services that fulfill the functional requirements

\begin{algorithm}
\begin{align*}
S &= \text{Semantic interface signature of the template term in the semantic template.} \\
\pi_{\text{ef}} &= \text{Effective policy of the semantic template.} \\
R_{S}^{E} &= \text{Expected level of domain match} \\
R_{\theta}^{E} &= \text{Expected level match over the set of operations} \\
R_{D}^{E} &= \text{Expected level match over the set of data elements} \\
\text{Compute } R_{S}^{E}, R_{\theta}^{E}, R_{D}^{E} \\
\text{Identify all semantic interface signatures } S' \text{ such that } \text{IR}(S, S') \in R_{S}^{E} \times R_{\theta}^{E} \times R_{D}^{E} \\
\text{For each } S' \text{ fetch the service interface from semantic interface link object} \\
\text{Using the service interface object, fetch the set of services that fulfill the functional requirements from the service implementation info object.}
\end{align*}
\end{algorithm}

A template is defined as the conjunction of the operation policy, the term policy of the enclosing template term and the template level policies. We say that a service matches the non-functional requirements, if its published policy matches the effective policy of the each operation in the semantic template.

Verma et al. [77] propose a method for semantic matching WS-Policies. We recall here that a policy (\(\pi\)) is defined as a collection of alternatives (\(A\)) and alternative is a collection of assertions (\(\alpha\)). Formally,

\[ \pi = \{ A_1, A_2, ..., A_n \} \text{ where each } A_i = \{ \alpha_1, \alpha_2, ..., \alpha_n \} \]

To match two policies, we compare their alternatives. Policy \(\pi_1\), is said to match policy \(\pi_2\), if for at least one alternative \(A_i \in \pi_1\), there is an alternative \(A_j \in \pi_2\) such that \(A_i \equiv A_j\). Equivalence between alternatives is defined on their set of assertions. Alternative \(A_i \equiv A_j\) if for each assertion \(\alpha_s \in A_i\) there is an assertion \(\alpha_t \in A_j\) such that \(\alpha_s \equiv \alpha_t\).

We fetch the published policy (\(\pi\)) of each service that fulfills the functional requirement from the registry and match it with the effective policy of each operation in the semantic template by
5.5. PUBLICATION AND SELECTION

August 25, 2009

comparing their alternatives. Let $A^S = \{A^S_i\}$ be a collection of alternatives such that $\pi = A^S$ and $A^T = \{A^T_j\}$ be a collection of policies such that $\pi_\theta = A^T$.

We compare each alternative $A^S_i \in A^S$ with all the alternatives $A^T_j \in A^T$. $A^S_i = \{\alpha^u\}$, where each $\alpha^u$ is a business assertion and $A^T_j = \{\alpha^v\}$, where each $\alpha^v$ is a business assertion. For each assertion $\alpha^u$ we check if there is an assertion $\alpha^v$, such that $\alpha^u \equiv \alpha^v$. If $\alpha^v$ exists for all $\alpha^u$, then $A^S_i \equiv A^T_j$ and $\pi$ matches $\pi_\theta$. If $\alpha^v$ does not exist, then we look at the semantic meta-model to see if there is any rule defined to compute the assertion term of $\alpha^v$. If there is such a rule, we identify the assertions whose assertion terms are the parameters of the rule. We evaluate the rule and compare the result with $\alpha^u$. If they are equivalent then $A^S_i \equiv A^T_j$ and $\pi$ matches $\pi_\theta$.

Similarly, we compare $\pi$ with the effective policy of the remaining operations in the semantic template. If $\pi$ is equivalent to the effective policy of each operation, then S fulfills the non-functional requirements.

5.5.3.3 Ranking

Once the set of services that match the functional and non-functional requirements are discovered, we rank the set of services based on the degree of match. The degree of match of a service is an aggregate of the degree of functional requirements match and the degree of non-functional requirements match.

We assign weights for the relationship over domain, relationship over operations and relationship over data elements.

To compute the degree of functional requirements match, we compute the interface relationship $I_R(S, S')$ between the semantic service signature ($S'$) of the service and the semantic interface
signature(S) of the semantic template. The degree of match then is the sum of the weights of the $R^S$, $R^\theta$ and $R^D$ in the interface relationship. For example, if $I_R(S, S') = (\equiv_S, \equiv^\theta_P, \sqsubseteq^D)$, then the degree of match is sum of the weights of $\equiv_S, \equiv^\theta_P$ and $\sqsubseteq^D$.

Computing the degree of match over interface relationship does not include the level of non-functional match. All the services that are selected fulfill the mandatory non-functional requirements. Hence it is not possible to separate one from another based on the mandatory non-functional requirements. However, a service that fulfills all the optional requirements in addition to the mandatory requirements is a better match. We use this intuition and assign a equal weight of 1 for each optional non-functional requirement fulfilled.

The degree of match between a service and a semantic template is then calculated as the sum of the degree of match over functional requirements and the non-functional requirements. The set of selected services is then ranked according to the degree of match. In case of two services having the same degree of match, the service with a higher degree of functional requirements match is ranked higher.

5.6 Evaluation

5.6.1 Implementation

In this Section we discuss the implementation of the registry. Our implementation is based on open source implementations. Our middleware has two main capabilities: 1) Service selection and publication and 2) Matching of non-functional requirements. The publication and selection
registry middleware was implemented as an open source framework by extending the source of jUDDI \(^2\) open source registry framework and Apache Neethi WS-Policy processing framework \(^3\). To support storage of semantic metadata in the registry, we added the objects discussed in Section 5.4 to the UDDI data model in jUDDI. The SAWSDL4J \(^4\) implementation of the SAWSDL specification was integrated into the jUDDI source to support publication of services described in SAWSDL. Components to support semantic templates during discovery were also added to the jUDDI implementation. The system uses the Jena semantic framework for storing and querying the semantic meta-models. The current implementation of the system supports both direct subclass querying and subsumption reasoning. We use the Jena implementation of the SPARQL to query language to query for direct classes. The Racer reasoner framework handles the subsumption reasoning responsibilities in the middleware.

Our approach to matching non-functional requirements is based on comparing the business assertions described in WS-Policy. Towards this end, we defined a new class of policy assertions called business assertions. We extended the source of Apache Neethi framework to support the processing of business assertions. We expressed our business rules such as supply time is a sum of production time and shipping time in the Jess rule language \(^5\) and the Jess rule engine was used for processing the rules. Semantic meta models in the system are described in OWL.

### 5.6.2 Evaluation

The objective of this evaluation is four-fold.

\(^2\)http://ws.apache.org/juddi/
\(^3\)http://ws.apache.org/commons/neethi/
\(^4\)http://knoesis.wright.edu/opensource/sawsdl4j/
\(^5\)http://www.jessrules.com/
5.6. EVALUATION

- To study the performance of the system during the publication of service interfaces,
- To study the performance of the system during the publication of services,
- To compare the accuracy of the framework in selection of services against other approaches to service selection,
- To demonstrate the usefulness of the flexibility of the proposed approach.

5.6.2.1 Test case generation

In order to perform the experiments, we generated 200 unique service interface definitions in SAWSDL. We selected ten domains from the NAICS industrial classification hierarchy. For each domain, concepts in the supply chain ontology that would define the operations were identified. We then generated SAWSDL interfaces for all the NAICS domains over these supply chain ontology concepts. The input and output properties for each concept in the supply chain ontology were then added as the input and output annotations. To these 200 interfaces, we added binding information and generated 1000 services.

The NAICS ontology is based on the NAICS industrial classification taxonomy. The supply chain ontology is derived from the RosettaNet standards for supply chain. The experimental setup also included a QoS ontology. We extended the OWL-QoS ontology developed by various members of the semantic Web community to model supply chain specific QoS metrics such as production cost, supply time, discounts etc. Rules are associated with the concepts in the QoS ontology and were modeled in Jess. For experiments related to matching of non-functional requirements, we created 40 policies that contained various business assertions. These were attached
to the services using WS-Policy attachment.

5.6.2.2 Experimental Results

The intention behind our first experiment is to study the time taken for publishing service interfaces. This experiment would give us a good estimate of the impact of computing the interface relationship during publication. The results of this experiment are presented in Figure 5.4. The time taken to add a new interface when there are no interfaces in the registry was around 100ms. This remained almost constant for the first few interface publications. The time taken to add a new interface increases with the number of interfaces. As illustrated in Figure 5.4, this increase is not monotonic because of the way the interface relationship is computed. If a service interface belongs to a domain that has a small number of published interfaces, then the time taken to add this service interface will be significantly lower than the time it takes to add a new interface that belongs to a domain that has a large number of published service interfaces. We can infer that the only factor that determines the publication time of a new service interface is the number of published interfaces.
that share a relationship over domain with the new interface.

Our second experiment studies the time taken to publish a new service. In this experiment we publish 1000 services. We study the time taken to publish a service both when a conforming interface is present in the registry and when it is not. The result of this experiment is illustrated in Figure 5.5. We first study the time taken to publish a service when the conforming interface is present. As expected the time taken to publish the service shows very little variation. Because we store the interface relationships in the registry, the problem of finding the conforming interface reduces to querying the semantic signature table. Hence the publication time is almost constant. The second part of the experiment was to study the publication of services when the conforming interfaces are not published in the registry. The series in Figure 5.5 corresponding to this publication shows significant variation.

The third experiment evaluates the accuracy of our discovery approach when using different match criteria. The results are illustrated in Figure 5.6. We set this experiment up by identifying
42 services in the registry that would fulfill our requirement. We first discover services based only on the input and output elements. The matching algorithm considered only the relationships over data elements. The number of services that matched the input and output was 153. This meant an accuracy of less than 30%. When the match criteria was extended to include operations, the number of services that matched went down to 87. The accuracy went up by more than 20 percentage points. When the match criteria included the domain, operation and data elements we obtained the desired result of 42. This experiment demonstrates the importance of considering all the elements of a service while matching.

We then examined the performance of our algorithm to identify services that fulfill the functional requirements. The results are illustrated in Figure 5.7. The registry performed fairly well in this experiment with the maximum time taken to discover a service being only 60 ms. Our approach to discovery relies on the interface relationship between the requirement and the services in the registry. This interface relationship is computed during the publication time and is stored in the registry. This experiment underlines the effectiveness of the interface relationships in optimizing
service discovery.
Data Mediation, Mediatability and

Mediation as a service

Mediation and integration of data are significant challenges because the number of services on the Web, and heterogeneities in their data representation, continue to increase rapidly. To address these challenges, a new measure named mediatability, which is a quantifiable and computable metric for the degree of human involvement in XML schema mediation, is introduced, along with an efficient algorithm for computing the same. Experimental studies demonstrate the efficiency of the algorithm while highlighting the effect of semantic annotations in easing the mediation process between two schemas. Scalability of the system is also discussed.

The increased adoption of the REpresentational State Transfer paradigm (Fielding (14)) has made it easier to create and share services on the Web. RESTful services often take the form of RSS/Atom feeds and AJAX-based light-weight services. The XML-based messaging paradigm of RESTful services has made it possible to bring discrete data from services together and create more meaningful data sets. This is commonly referred to as building a mashup. A mashup is the Web application created using two or more existing Web application interfaces. Some impediments in the creation of mashups are: 1) the programming skill required to develop such
applications (largely due to the complexity of languages such as javascript) and 2) the arduous task of mapping the output of one service to the input of another. Frameworks such as Google Mashup Editor\(^1\) and IBM Sharable Code\(^2\) have addressed the first problem with reasonable success by creating programming-level abstractions. However, little work has been done towards helping the developers in the task of data mediation.

The importance of understanding and addressing the problem of data mediation in distributed systems is underscored by the volume of research in matching and mapping heterogeneous data. *Matching* is the task of finding correspondences between elements in schemas or instances. Once the corresponding elements are identified, *mapping* defines the rules to transform elements from one schema into another. Matching and mapping have been well studied by various researchers including Kahsyap and Sheth\(^{23}\), Nagarajan *et al.*\(^{47}\) and Madhavan *et al.*\(^{32}\) in different contexts. Considerable research effort has gone into creating frameworks that attempt automated and semi-automated matching and mapping of heterogeneous data. These efforts, have yielded limited success, however, and developers are often left with the difficult task of performing the mediation manually.

The end goal of traditional schema matching has been to establish semantic similarity between schema elements. However, semantic equivalence does not guarantee interoperation. Depending on the heterogeneities between the schemas, mediation is harder or even impossible to automate (Nagarajan *et al.*\(^{47}\)). Even when mediation is manual, it is hard to estimate the degree of human involvement in performing mediation between the two schemas. The goal of this work is to go a step beyond matching and define mediatability as a measure of the degree of human involvement.

\(^1\)http://editor.googlemashups.com/editor
\(^2\)http://services.alphaworks.ibm.com/isccore
Evaluations demonstrate that such a measure would help users in selecting services, especially in the light-weight services scenario, where often one has to choose from a plethora of services that offer the same or similar features with little separation.

Our experience with IBM Sharable Code (Maximilien and Ranabahu (35)) largely motivated this work in quantifying ease of mediation. In creating the data components for the IBM sharable code mashups, a significant amount of effort was needed to pick the correct data elements, often from large and complex schemas. To illustrate, programmableWeb.com³, a popular RESTful API directory, returns 71 services for the search keyword mapping. Most real-world services (for example Amazon⁴, Microsoft Live⁵) model a rich schema, making them large and verbose. We believe, based on our experience on creating real-world mashups (Sheth et al. (62)), having a quantifiable measure of the degree of human involvement in mediation, would serve as a useful metric in the selection of services.

This work makes two unique contributions.

- First, we introduce the concept of mediatability as an indicator of the degree of human involvement in mediation between two schemas. Further, we provide a quantifiable definition of mediatability that takes into account the element level similarity and the structural similarity of the two XML schemas.

- Second, we provide an efficient two-pass algorithm for computing the mediatability. The similarities are computed in the top-down pass and the mediatability is computed in the bottom-up pass. Further, we discuss an optimization technique to get a better average case

³http://www.programmableWeb.com/apitag/?q=mapping, 03/14/2008
⁴http://soap.amazon.com/schemas2/AmazonWebServices.wsdl
⁵http://soap.search.msn.com/Webservices.asmx?wsdl
There has been activity in semantically annotating schemas and since they are a high indicator of semantic similarity between two elements, it is valuable to see what this brings to the problem of computing mediatability. We provide an experimental study to analyze the impact of having semantic annotations in determining the ease of mediation between two schemas. We validate our approach by comparing the mediatability scores generated by our system against that of user perceived difficulty in mediation. We also evaluate the scalability of our system.

6.1 Motivation

We illustrate the need for and the use of mediatability by the example of a developer trying to create a mashup in which one of the services is an image search service. Examples of such mashups can be found at Yahoo! Inc. Services such as Microsoft live search and Yahoo image search return image results for a given search string, and the developer has to choose one
of these services. Snippets of the Yahoo image search and Microsoft live search result schemas along with the desired target schema of the developer are illustrated in Figure 6.1. For purposes of the example, we consider the schemas of Live and Yahoo image search to be the source schemas. As we can see from Figure 6.1, the live result schema is nested and deep, while the Yahoo schema is shallow. Given that both Live and Yahoo image search services return a set of images for a given search query, one metric that can help differentiate between the two services is the ease with which the developer can mediate between the schema of the service provider and the target schema. Mediatability is the measure of the ease of performing this mediation.

In the next section, we define mediatability and illustrate with an example based on the source and target schemas illustrated in Figure 6.1.

### 6.2 Mediatability: Definition and Computation

In this Section we present the conceptual definition of mediatability between two schemas and discuss our approach to calculating a concrete quantifiable metric. Mediatability is defined as the measure of the degree of human involvement in mediation between two schemas based on their semantic and structural similarities. The value of mediatability between two schemas lies between 0 (hardest to mediate; indicates significant human effort) and 1 (easy to mediate; indicates little effort). Formally, mediatability between a target schema $T$ and a source schema $S$ is defined as

$$\sigma(T, S) = x : x \in [0, 1]$$

While we believe that such a notion can be defined between any two schemas (databases, ontologies), in this chapter we focus on computing the mediatability for XML schemas. The conceptual
definition of mediatability cannot be used directly. We present a computable and quantifiable definition of mediatability between two schemas and discuss our approach toward calculating mediatability between two schemas.

6.2.1 Overview
Mediatability between two schemas is computed by first computing the mediation similarity between the elements of the two schemas. The mediation similarity between elements is a function of their element similarity and structural similarity. Element similarity between two elements is a function of Semantic Similarity, Wordnet Similarity, Lexical Similarity and Type Similarity.

To compute the structural similarity, we first identify the nearest similar ancestor of the two elements. The nearest similar ancestor between an element \( e^t_i \) in the target schema and an element \( e^s_j \) in the source schema is a pair of elements \( e^t_p \) in target schema and \( e^s_q \) in source schema such that \( e^s_q \) belongs to the similarity set of \( e^t_p \) and \( e^t_p \) is the nearest such element to \( e^t_i \) in the target schema. The mediation similarity between \( e^t_i \) and \( e^s_j \) is defined as a measure of the structural and the semantic similarity between the two elements and is a function of the element similarity between them, the mediation similarity between their nearest similar ancestor elements (NSA) and the distance between the elements and their NSA.

The mediatability between an element in the target schema and an element in the source schema is computed in a recursive manner by computing the mediatability between the elements in the two schemas. The computation is performed in a bottom-up manner, beginning with the leaf elements and terminating at the root element. This is illustrated in Figure 6.3(b). The mediatability between two elements is the average mediatability between their respective child elements. If
6.2. MEDIA TABILITY: DEFINITION AND COMPUTATION

August 25, 2009

an element in the target schema is a leaf node, then the mediatability between that element and
an element in the source schema is same as the mediation similarity between them. The formal
definition and a detailed discussion about computing the mediatability is presented in Section 6.2.5.

We now present our approach for computing the mediatability in detail.

![Diagram](image_url)

Figure 6.2: (a) Computing Element Similarity (b) Nearest Similar Ancestor

### 6.2.2 Computing Element Similarity

Converting the source and target schemas into schema hierarchy trees is the first step in computing
the mediatability. The schema hierarchy trees are created by converting each element in the XML
schema to a node that contains the name of the XML element, the semantic annotation on that
element and the XML data type of the element. If the type of an XML element is a complex type,
then the data type property of that node is empty. Complex types and references are expanded
in place. The in place expansion allows us to model the schema as a tree and removes the links
between different elements in the schema. In our discussion we denote the source schema hierarchy
tree as $H_s$ and the target schema hierarchy tree as $H_t$. Elements in the source schema hierarchy
tree are denoted by $e^s_j$ and the elements in the target schema hierarchy tree are denoted by $e^t_i$.

Once the schema hierarchy trees are constructed, we compute the element similarity between
the elements in $H_t$ and $H_s$. This is computed in a top-down manner starting with the root of the
target schema hierarchy. To compute the element similarity, we compare the elements in the target and source trees. The element similarity computation is illustrated in Figure 6.2(a).

- **Semantic Similarity:** If semantic annotations are present in both the target and source elements, concept similarity is calculated by computing the relationship between the concepts in the semantic model referenced by the annotations. If the relationship between the concepts is one of subclass, superclass or equivalence, then the semantic relationship is used in defining the semantic similarity. Since the SearchResult element of the target schema and the Result element of schema A in Figure 6.2(a) have annotations and the annotations are equivalent, the semantic similarity between them is 1. This is defined as,

\[
S_{sim}(e^t_i, e^s_j) = \begin{cases} 
W_{sub} & t_a \sqsubseteq t_s \\
1 & t_a \equiv t_s \\
W_{sup} & t_a \sqsupseteq t_s \\
0 & t_a, \text{ other relationships}
\end{cases}
\]

where \(W_{sub}\) and \(W_{sup}\) are scores assigned to subclass and superclass relationships and \(t_a\) and \(t_s\) are the ontology concepts referenced by the source and target annotations respectively.

- **Wordnet Similarity:** If the semantic similarity cannot be computed or is zero, we compute the wordnet similarity between the element names based on the relationship between them in Wordnet (Miller (41)). In Figure 6.2(a), the Photo element of the target schema and the Image element in schema A are not annotated. Hence the similarity between them is computed using wordnet. Since they are synonyms, their wordnet similarity is 1. The wordnet
6.2. MEDIATABILITY: DEFINITION AND COMPUTATION

similarity is defined as,

\[
W_{\text{sim}}(e^t_i, e^s_j) = \begin{cases} 
\frac{W_{\text{hypo}}}{d} & e^t_i \text{ hyponym of } e^s_j \\
1 & e^t_i \text{ synonym of } e^s_j \\
\frac{W_{\text{hype}}}{d} & e^t_i \text{ hypernym of } e^s_j \\
0 & \text{ otherwise.}
\end{cases}
\]

where \(W_{\text{hypo}}\) and \(W_{\text{hype}}\) are scores assigned to hyponym and hypernym relationships respectively and \(d\) is the depth of the relationship.

- **Lexical Similarity:** If both the semantic similarity and the wordnet similarity is zero, we compute the lexical similarity between the element names using edit distance. This is denoted by \(L_{\text{sim}}\). In the example illustrated in Figure [6.2](a), the lexical similarity between the SearchResult element of the target schema and the SearchResponse element of Schema A is computed, since their semantic and wordnet similarities are zero.

- **Type Similarity:** The type similarity \((T_s(e^t_i, e^s_j))\) between the elements is calculated by comparing the xsd:type of the elements and the similarity value is based on the two types being compared. If the types match, then the type similarity is exact.

We define the element similarity as,

\[
E_s(e^t_i, e^s_j) = C_s(e^t_i, e^s_j)T_s(e^t_i, e^s_j)
\]  

(6.1)
where, $C_s(e_i^t, e_j^s) = \begin{cases} S_{sim}, & \text{if semantic similarity} \\ W_{sim}, & \text{if wordnet similarity} \\ L_{sim}, & \text{if lexical similarity} \end{cases}$

Figure 6.3: (a) Computing mediation similarity (b) Mediatability Computation

### 6.2.3 Factoring Structural Similarity

In computing the element similarity, we only consider the similarity between the semantic annotations and the element names along with the type similarity. The structural similarity, which plays a crucial role in determining the mediation similarity cannot be ignored. For example, the width element, which is a child of element of the photo element in the target schema in Figure 6.2(b) would match completely with the width elements contained in both Image and Video elements in schema A, if one were to only consider the annotation and type similarities. However, the the similarity between the image element in the schema A and the photo element in the target schema is higher than that between the video element in schema A and photo element in the target schema . Factoring this information, we can say that the width element under the image element is more similar to the width element in the target schema. We define the nearest similar ancestor between an element in the target hierarchy and an element in the source hierarchy.
The Nearest Similar Ancestor ($NSA(e^i_t, e^s_j)$) is the pair of elements $(e^t_p, e^s_q)$ such that $e^s_q$ belongs to the similarity set of $e^t_p$. This is defined as,

$$NSA(e^t_i, e^s_j) = (e^t_p, e^s_q) : e^s_q \in S_{e^t_p} \quad \land \quad e^t_p \text{ is the nearest ancestor of } e^t_i.$$ (6.2)

where $S_{e^t_p}$ is the similarity set of $e^t_p$. The similarity set of an element is defined later in the Section. The definition of nearest similar ancestor between two elements in a hierarchy is inspired by the definition of nearest common ancestor proposed by Harel and Tarjan in Harel and Tarjan (22).

### 6.2.4 Computing mediation similarity

Using the element similarity and the nearest similar ancestor, we define the mediation similarity between $e^t_i$ and $e^s_j$. Two elements may have an element similarity of 1, but if there is very little structural similarity between the two schemas, the mediation similarity would be significantly lower. The structural similarity depends on the level of the target and source elements in the respective hierarchy trees from their nearest similar ancestors. If the NSA $(e^t_i, e^s_j)$ exists, the mediation similarity is measured by factoring their element similarities, the mediation similarity between the NSA elements and the distance between $e^t_i$, $e^s_j$ and their respective ancestors in the NSA. If there is no similar ancestor between $e^t_i$ and $e^s_j$, the mediation similarity is computed factoring in the element similarity and the depth of the elements in the hierarchy. If either of the two elements is the root element, then its depth is taken to be 1. The formulae for computing the mediation
similarity is.

\[
OS(e_i^t, e_j^s) = \begin{cases} 
E_s(e_i^t, e_j^s) \frac{OS(e_i^t, e_j^s)}{d_{ip}d_{jq}} & \text{if NSA is empty.} \\
E_s(e_i^t, e_j^s) \frac{1}{d_i \cdot d_j} & \text{if NSA is empty.}
\end{cases}
\]  

(6.3)

where \(d_{ip}\) is the depth of \(e_i^t\) from its nearest similar ancestor, \(d_{jq}\) is the depth of the \(e_j^s\) from its nearest similar ancestor, \(d_i\) is the depth of \(e_i^t\) and \(d_j\) is the depth of \(e_j^s\). We now illustrate with an example. Consider the target schema and schema in Figure 6.3 (a). The element similarity between the SearchResult element in the target schema and the Result element in schema A is 1. Now the depth of the Result element in schema A is 4, while the SearchResult element in the target schema is the root and hence its depth is taken to be 1. The mediation similarity between the two elements is 0.25. Now we consider the Photo element of the target schema and the Image element of schema A. The NSA(Photo, Image)= (SearchResult,Result). The element similarity between the photo and the image elements is 1 and the mediation similarity of the NSA elements is 0.25, from the above. Using the formula for mediation similarity defined in Equation (6.3), the mediation similarity between photo and the image element is 0.25.

The similarity set of \(e_i^t\) \((S(e_i^t))\) is the set of elements \(e_j^s\) in the source schema that have the maximum similarity value with \(e_i^t\).

\[
S(e_i^t) = \{ e_j^s : OS(e_i^t, e_j^s) \text{ is maximum} \}
\]  

(6.4)

As an example, the similarity set of the photo element of the target schema is \{ Image \}.

The mediation similarity coefficient of a target element \(e_i^t\) is the maximum mediation similar-
6.2. MEDIATABILIT Y: DEFINITION AND COMPUTATION

ity value between $e_i^t$ and any source element.

$$OS_C(e_i^t) = \text{maximum mediation similarity value between}$$

$e_i^t$ and any source element. \hspace{1cm} (6.5)

As an example, in Figure [6.3]a), the mediation similarity coefficient of the Photo element of the target schema is 0.25.

6.2.5 Calculating Mediatability

We now discuss the calculation of the mediatability between two schemas. While element similarity is computed in a top-down manner, mediatability is computed in a bottom up manner, beginning with the leaf elements of the target schema.

Mediatability of an element $e_i^t$ in the target schema is denoted by $\sigma$. If an element $e_i^t$ is a leaf element, the mediatability of $e_i^t$ is the same as its mediation similarity coefficient defined in Equation [6.5]

$$\sigma(e_i^t) = OS_C(e_i^t) \hspace{1cm} (6.6)$$

The width element in the target schema in Figure [6.3]b) is a leaf element. Hence its mediatability is the same as its mediation similarity coefficient, which is 0.25. For each $e_i^t$ that is not a leaf element, the mediatability of $e_i^t$ defined as the average of mediatability between its immediate children.

$$\sigma(e_i^t) = \frac{1}{z} \sum_{m=0}^{z} \sigma(e_m^t) \hspace{1cm} (6.7)$$

where $z$ is the number of immediate children of $e_i^t$. The mediatability of the photo element in
the target schema in Figure 6.3(b) is the average mediatability of its children. Since all the child elements of the photo element have a mediatability of 0.25, the mediatability of the photo element is 0.25.

Before we define the mediatability between the source and target schemas, we make a small but important observation. Once the mediatibilities are computed for all elements, it is possible that the root element of the target schema has more than one member in its similarity set, implying that the source schema may have more than one substructure that can be mediated with the target schema. To reflect the effort needed to identify the correct substructure, we consider the cardinality of the root element’s similarity set in defining the mediatability between the two schemas. We now define the mediatability between the target and source schemas as the ratio of the mediatability of the root element of the target schema and the cardinality of its mediatable set.

\[
\sigma(H_t, H_s) = \frac{1}{|S(\text{root of } H_t)|}\sigma(\text{root of } H_t)
\]  

(6.8)

The mediatability between the two schemas in Figure 6.3 is computed as follows. The mediatability of the root element (SearchResult) is 0.25. The similarity set of the SearchResult element, which is the root, is \{Result\}. The cardinality of the similarity set of the root is 1 and its mediatability is 0.25. Computing the mediatability between the two schemas as defined in Equation 6.8, we get 0.25.
6.2.6 Optimizing Time Complexity

One of the drawbacks of the approach to comparing every element in the target schema is that the computational complexity is \(O(n^2)\). This inefficiency is further enhanced by the fact that often times, the comparison will yield no meaningful results. As a way of optimizing this comparison, we define the scope of comparison. We adopt a method similar to \(\alpha\beta\) pruning to reduce the number of elements in the source schema that need to be compared with a given element in the target schema. The children of an element \(e^t_i\) in the target schema would be compared only with the children of those elements in the similarity set of \(e^t_i\). The children of those elements in the source schema that belong to the similarity set of \(e^t_i\) are the scope of comparison for the children of \(e^t_i\). Defining the scope of comparison would help reduce the complexity of the average running time of element similarity computation. In our example, the width element in the target schema would be compared with the children of the image element in the source schema, since the image element in source schema A is in the mediatability similarity set of the parent element of width.

6.3 Evaluation

In this Section we present the empirical evaluations of our algorithm. The objective of our empirical evaluations is three fold: 1. Evaluate the accuracy of our approach through a user study; 2. Study the impact of semantic annotation on mediatability and 3. Demonstrate the scalability of our algorithm.

In our experiments, we compare a target schema with 5 different source schemas. The source schemas are created by studying the results schemas of Yahoo Web Search \(^6\) (schema A), Google

\(^6\)http://developer.yahoo.com/search/Web/V1/WebSearch.html
6.3. **EVALUATION**

Search (schema B), Microsoft Live Search (schema C), Yahoo Image Search (schema D) and Flickr (schema E). The schemas for Google Web Search and Flickr search were created by studying their responses, since they do not provide XML schemas explicitly.

In our experiments, subclass similarity is assigned a value of 0.5 and superclass similarity is assigned a value of 0.8. Hyponym and hypernym scores are calculated as $\frac{1}{l}$, where $l$ is the length of the hyponym or hypernym relationship in wordnet. The Levenshtein measure is used in the computation of lexical similarity.

### 6.3.1 Evaluating Accuracy

Our first experiment compares the mediatability scores obtained by our algorithm with a set of normal and expert users. The set of expert users comprised of committers of XML centric Apache projects including Apache Axis and Apache XML Schema. Normal users consisted of mashup developers having minimal programming and XML expertise. We included the normal users to compare our scores with the perceived difficulty of average developers, who we believe will have the most benefit from our work. Users were asked to rank the mediatability between the source and the target schemas using a Web application. Our results, illustrated in Figure [6.4](#), show that the calculated mediatability scores match fairly well with the perceived mediatability values and agree well with the expert opinions. The average margin of error between the system calculated mediatability and the perceived mediatability of the normal users was less than 15%, while the margin of error with expert users was less than 10%. We make a special observation about schemas A and E. We recall here that schema A was derived from Yahoo Web Search. This schema did not

---

10[^flickr]: [http://www.flickr.com/services/api/flickr.photos.search.html](http://www.flickr.com/services/api/flickr.photos.search.html)
have any image element in its result set and hence was given a low mediatability score to account for the loss of information. However, users perceived the mediatability to be twice as easy as the system calculated value. This indicates that our approach is very conservative and does not overestimate. Similarly, schema E (derived from Flickr), had a structural heterogeneity, that was penalized by the system.

### 6.3.2 Impact of Annotation

This experiment measures the impact of semantic annotations in determining the mediatability. We annotated the source and target schemas with concepts from the semre descriptor ontology, a categorization of Web API’s derived from ProgrammableWeb. The mediatability was calculated when the schemas have no annotations, partial annotations and complete annotations. The schemas were annotated using the techniques described in the SAWSDL recommendation discussed by Verma and Sheth (78). Schemas with partial annotations were created by adding top-level annotations to complex types. Schemas with complete annotations were created by adding annotations to the leaf
6.3. EVALUATION

August 25, 2009

Figure 6.5: Impact of Semantic Annotation

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Annotations</td>
<td>0.115</td>
<td>0.293</td>
<td>0.381</td>
<td>0.373</td>
<td>0.306</td>
</tr>
<tr>
<td>Partial Annotations</td>
<td>0.231</td>
<td>0.592</td>
<td>0.771</td>
<td>0.755</td>
<td>0.615</td>
</tr>
<tr>
<td>Complete Annotations</td>
<td>0.323</td>
<td>0.763</td>
<td>0.927</td>
<td>0.927</td>
<td>0.793</td>
</tr>
</tbody>
</table>

elements in addition to the top-level annotations. Figure 6.5 illustrates the impact of annotation on mediatability. In the case of schema A, where the target schema has more elements than the source schema, the mediatability is low in all the three cases. However, we can see that semantic annotations considerably improve the mediatability score. Having partial annotations does not impact the mediatability in the case of schema A, since there are no complex types in the source schema. In the case of schemas B, C and D that contain complex types, one can see that complete annotations significantly improves the mediatability score and even partial annotations have an impact on the mediatability. On average our experiments demonstrate that partial annotations improve the mediatability by a factor of 2 while having complete annotations improves the mediatability by a factor of 3.

6.3.3 Evaluating the Scalability

Our third experiment demonstrates the scalability of our algorithm. The algorithm was tested on two systems with different computing resources. System 1 is a Mac Book Pro running OSX 10.5
with 2 GB RAM and Intel Dual Core 2.0 GHz processor. System 2 is a Dell server running Fedora Core 5 with 16 GB RAM and AMD Quad Core 2.4 GHz processor. As illustrated in Figure 6.6, we see that in the worst case, system 1 takes 36 seconds to compute the mediatability and system 2 accomplished the task in 25 seconds. This demonstrates the scalability of our algorithm. Figure 6.6 measures the scalability when the source schema has 364 elements and is 6 levels deep and the number of elements in the target schema are varied from 13 to 364. The depth of the target schema was varied from 3 to 6.

Figure 6.6: Measuring Execution Time
Bringing Them All Together: Composition and Configuration

This chapter addresses services composition in the context of heterogeneities across independently created and autonomously managed Web service requesters and Web service providers. Previous work in this area either involved significant human effort or in cases of the efforts seeking to provide largely automated approaches, overlooked the problem of data heterogeneities, resulting in partial solutions that would not support executable workflow for real-world problems. This chapter discusses a planning-based approach to solve both the process heterogeneity and data heterogeneity problems. We adopt a declarative approach to capture the partner specifications external to the process and demonstrate the usefulness of this approach in adding more dynamism to Web processes. We demonstrate the effectiveness of our approach by solving a non-trivial problem posed by the Semantic Web Services challenge.1

1http://sws-challenge.org/
7.1 Motivating Scenario

The 2006 SWS Challenge mediation scenario version 1, illustrated in Figure 7.1, is a typical real-world problem where distributed organizations are trying to communicate with each other. A customer (depicted on the left side of the Figure 7.1) desires to purchase goods from a provider (depicted on the right side of the figure). The anticipated process, i.e., the answer to this problem, is depicted in the center of the figure which should be generated by a mediation system automatically. Both process and data heterogeneities exist in this scenario. For instance, from the point of view of the service requester called Blue, placing an order is a one-step job (send PO), while the service provider called Moon, involves four operations (searchCustomer, createNewOrder, addLineItem, and closeOrder). The message schemas they use are not exactly the same. For example, Blue uses “fromRole” to specify the partner who wants to place an order, while Moon uses “billTo” to mean the same thing. The structures of the message schemas are also different. To make matters worse, an input message may involve information from two or more output message. For example, the operation “addLineItem” requires information from the order request message by Blue and the newly created order ID from the output message of operation “createNewOrder”. In order to solve this problem successfully and automatically, the composition system should at least be able to do the following: generate the control flow of the mediator that involves at least two workflow patterns (Sequence and Loop) based on the specification of the task and the candidate Web service(s), and convert (and combine if needed) an input message to an acceptable format annotated with appropriate semantics.
7.2 Declarative Approach towards Solution

One of the evaluation measures to determine the efficiency of the composition approach is the ability to manage change with minimal programming efforts. Systems developed using conventional approaches where the requirements and the services are not externalized from the actual system itself, may often prove to be inflexible. To overcome this limitation, we adopt a declarative approach to capture the requirements of the process and the service description of partner services. Our system generates a plan based on the requirement and discovers partner services based on their descriptions. A Web process is then generated that can be deployed and executed. When there is a change in the requirement, a new process can be generated using the changed requirements. The requirements are captured as a semantic template and partner services are described using SAWSDL. The non-functional properties of both the requirement and the service can be cap-
7.2. DECLARATIVE APPROACH TOWARDS SOLUTION  

We defined a new class of assertions called business assertions that can be added to WS-Policy to describe business level non-functional properties such as shipment destinations and shipment weight. It is our belief that the availability of visual XML editors and WSDL editors would make it easier to change these specifications. Further, this externalization eliminates re-compilation of the system for each change. Semantic templates and business assertions used in this chapter are discussed in chapter 5.

7.2.1 Formal model of abstract Web services:

WSDL is a widely accepted industry standard (a W3C recommendation) for describing Web services. SAWSDL is expressive for functional and data semantics, and sufficient to solve the problem of semantic discovery and data mediation. We extend SAWSDL by adding preconditions and effects in the operations for process mediation. Preconditions and effects are necessary because not all the states of a Web service are represented by the input/output message. For example, both a book buying service and book renting service may take as the input the user ID and the ISBN, and give as the output the status succeed or fail. Importance of pre-condition and effects have been recognized by major semantic Web services initiatives including OWL-S, WSMO and WSDL-S, here we do that by extending the emerging standard of SAWSDL.

For the purpose of service composition, our model only focuses on the abstract representation of Web services, i.e., operations and messages, but does not consider the binding detail. Before giving our formal model, we need to introduce some definitions of the basic building blocks. Most classic AI planning problems are defined by the STRIPS representational language (or its variants like ADL), which divides its representational scheme into three components, namely, states, goals,
and actions. For the domain of Web service composition, we extend the STRIPS language as the representational language of our method.

- **Extended state**: We extend a state by adding a set of semantic data types in order to ensure that the data for the input message of an operation is available before the operation is invoked.

An extended state \( s \) has two components:

\[
s = < SSF, SDT >,
\]

where:

- \( SSF \) is a set of status flags, each of which is an atomic statement with a URI in a controlled vocabulary. \( SSF \) defines the properties of the world in the specific state. We use ternary logic for status flags, thus the possible truth values are True, False, and Unknown. We use the open-world assumption, i.e., any status flag not mentioned in the state has the value unknown.

- \( SDT \) is a set of semantic data types representing the availability of data. A semantic data type is a membership statement in Description Logic of a class (or a union of classes) in an ontology. An example state could be:

\[
< \{\text{orderComplete} = \text{True}, \text{orderClosed} = \text{False}\}, \{\text{ONT1#OrderID}(\text{Msg}_1)\} >
\]

The reason why we use predicate logic for status flags is because it is simple for the user to specify the values of status flags in predicate logic, and computationally efficient. On
the other hand, we use description logic for semantic data types since it makes it easier to express relationships such as sub-class relationships.

- **Abstract semantic Web service** Verma \(^{(76)}\): Our definition of an abstract semantic Web service is built upon SAWSDL for WSDL and working group \(^{(16)}\) An abstract semantic Web service SWS can be represented as a vector:

\[
SWS = (sop_1, sop_2, E, sop_n)
\]

Each sop is a semantic operation, which is defined as a 6-tuple

\[
sop = \langle \{op, in, out, pre, eff, fault\} \rangle
\]

where,

- *op* is the semantic description of the operation. It is a membership statement of a class or property in an ontology.

- *in* is the semantic description of the input message. It is a set of semantic data types, stating what data are required in order to execute the operation.

- *out* is the semantic description of the output message. It is a set of semantic data types, stating what data are produced after the operation is executed.

- *pre* is the semantic description of the precondition. It is a formula in predicate logic of status flags representing the required values of the status flags in the current state before an operation can be executed.
Table 7.1: Representation of Order Management System Web service

- **eff** is the semantic description of the effect. It can be divided into two groups: positive effects and negative effects, each of which is a set of status flags describing how the status flags in a state change when the action is executed.

- **fault** is the semantic description of the exceptions of the operation represented using classes in an ontology.

Table 7.1 illustrates an example of the representation of part of the Order Management System Web service described in our running scenario.

### 7.3 Automatic Web service composition

#### 7.3.1 Formal definition of Web service composition

A semantic Web service composition problem involves composing a set of semantic Web services (SWSs) to fulfill the given requirements, or in our case a Semantic Template. Figure 7.2 illustrates our approach.

A semantic operation \(\text{Operation}_k\) in Figure 7.2 has to be checked by the *satisfy* operator \((X\ \text{in } 7.2)\) against the current extended state before it can be added in the process specification.
After it is added, a successor extended state is created by applying the \textit{apply} (+ in Figure 7.2) operator. We will give the formal definition of \textit{satisfy} and \textit{apply} operators below. For convenience, we use the following notations.

\textbf{Satisfy operator} is a function mapping an extended state $s_i$ and a semantic operation $sop_k$ to $T$ or $F$. Formally \textit{textitsatisfy} is defined as: That is, the precondition of $sop_k$ holds based on the truth values of the status flags in state $s_i$, and the semantic data types of $s_i$ together with the ontology schema entails the input of $sop_k$. For example, the following state will satisfy the operation $sop_3$ in table 7.1:

\[
< \{orderComplete = True, orderClosed = False\}, \{ONT1\#OrderID(Msg)\} >
\]
Here the semantic data type OrderID comes from an output message of any previous operation, or the initial message of the Semantic Template, so we put $M_{sgx}$ in the above example.

**Apply operator** is a function mapping an extended state $s_i$ and a semantic operation $sop_k$ to a new extended state $s_j$. Formally this is defined as

**Definition 7**  $\text{apply}: (s_i, sop_k) \rightarrow s_j$

Alternatively, we write $s_i + sop_k \rightarrow s_j$ This operator does the transition both on status flags and semantic data types.

- **For status flags:**

  \[
  \forall sf \in \text{positive}(\text{eff}(sop_k)), \text{value}(sf, s_j) = True \\
  \forall sf \in \text{negative}(\text{eff}(sop_k)), \text{value}(sf, s_j) = False \\
  \forall sf \in (\text{eff}(sop_k)), sf(s_j) = sf(s_i)
  \]

  That is, a status flag in the positive effects is true in $s_j$, a status flag in the negative effects is false in $s_j$, while any status flag in $s_i$ but not in the effect is assumed to be unchanged in $s_j$.

- **For semantic data types:** $\text{SDT}(s_j) = \text{SDT}(s_i) \cup \text{out}(sop_k)$ That is, the semantic data types (membership statements) in $s_j$ are the union of the semantic data types in $s_i$ and the output of $sop_k$.

As an example, if we apply the operation $sop_3$ in 7.1 to the state

$< \{\text{orderComplete} = True, \text{orderClosed} = False\}, \{\text{ONT1#OrderID}(M_{sgx})\} >$
we will get a new state:

\[
< \{\text{orderComplete} = \text{True}, \text{orderClosed} = \text{True}\}, \{ ONT1\#\text{OrderID}(\text{Msgx}), \\
ONT1\#\text{ConfirmedOrder}(\text{sop}_3\text{OutMsg})\} >
\]

### 7.3.2 Composition of semantic Web services

We consider an SWS composition problem as an AI planning problem such that the semantic operation template defines the initial state and the goal state of the problem specification: **Initial state** is the extended state at the beginning of the process. It is defined by the precondition and initial message of the semantic operation template \( \psi \).

\[
s_0 = < ssf_0(\text{sop}), in(\text{sop}) >
\]

**Goal state** is a requirement of the extended state at the end of the process. It is defined by the goal and output of \( \text{sop} \).

\[
goalstate = < gl(\text{sop}), out(\text{sop}) >
\]

**Composition of semantic Web services** is a function

\[
sWSC : (\text{sop}, \text{SWS}_s) \rightarrow \text{plan}
\]
Where,

- sopt is a semantic operation template.
- SWSs is the set of the semantic operations in the semantic Web services.
- \( \text{plan} \) is a DAG (Directed Acyclic Graph) of operations. Every topological sort of the DAG (say one of them is \( sop_1, sop_2, \ldots, sop_n \)) must conform to the following restrictions:

\[
\begin{align*}
- & s_0 \times < \text{pre}(sop_1), \text{in}(sop_1) > \\
- & s_0 + sop_1 \rightarrow s_1 \\
- & s_{i-1} \times < \text{pre}(sop_i), \text{in}(s_i) > \\
- & s_{i-1} + sop_i \rightarrow s_i \\
- & s_n \times \text{goalstate}
\end{align*}
\]

That is, every topological sort of the plan must transform the initial state into the goal state by conforming to the \textit{satisfy} and \textit{apply} operators. Loops are generated in a post-process step that is explained in Section 7.3.5.

### 7.3.3 Planning For Process Mediation

AI planning is a way to generate a process automatically based on the specification of a problem. Planners typically use techniques such as progression (or forward state-space search), regression (or backward state-space search), and partial-ordering. These techniques attempt to use exploration methods such as searching, backtracking, and/or branching techniques in order to extract such a
solution. There are two basic operations in every state-space-based planning approach. First, the precondition of an action needs to be checked to make sure it is satisfied by the current state before the operation can be a part of the plan. Second, once the operation is put into the plan, its effect should be applied to the current state and thus produce a consecutive state. We address the significant differences between classic AI planning and semantic Web service composition as follows:

1. Actions in AI planning can be described completely by its name, precondition, and effect, while Web services also include input and/or output message schema.

2. For AI planning, it is assumed that there is an agreement within an application on the terms in the precondition and effect. Terms with same name (string) mean the same thing, while terms with different name (string) mean different things. For example, in the famous block world scenario, if both block and box exist in the precondition/effect, they are treated as different things. This obviously does not carry over to the resources on the Web, thus it is necessary to introduce semantics in Web service composition.

3. More workflow patterns such as loops are desired in Web service composition. We address this problem by a pattern-based approach.

As discussed in the previous sections, both Web services and the specification of the task, i.e., Semantic Template are described in extended SAWSDL standard, so the terms in the precondition, effect, and input/output messages reach an agreement which is captured by the ontologies. For the first two types of differences we mentioned above, to apply AI planning techniques to semantic Web service composition, any state-space-based planning algorithm needs to be revised according
1. State space should include status flags, as in the existing AI planning approaches, and semantic data types to represent the availability of data.

2. For each candidate action, besides checking its precondition against the status flags in the current state, it is also necessary to check its input message schema against the semantic data types in the current state. This reduces the search space and eliminates plans containing operations whose input message is unavailable in the state.

3. Since the states and the actions/operations are semantically annotated by referring to ontologies, the checking in the previous step involves reasoning based on the ontologies, not just comparing the name of the terms.

4. Once an action/operation is added into the plan, not only the status flags are updated by applying the effect, the semantic data types should also be updated by put a new semantic data type based on the output message schema.

### 7.3.4 Extended GraphPlan Algorithm

Although most AI planning algorithms are suitable for the task here, we use GraphPlan algorithm (Russell and Norvig (58)). It is sound and complete thus we can always construct correct plans if there exist any, and its compact representation of the states makes it space efficient while doing a breadth-first style search. It also uses mutex links to avoid exploring some irrelevant search space. Like other classical AI planning algorithms, GraphPlan only considers the precondition and effect of actions, thus does not take into account the input/output message of actions. Our approach
requires an extension of the algorithm to accommodate the semantic data types defined above. An operation may only be added in the next action level when its preconditions hold based on the current state level of the planning graph and the data types of the input message of the operation can be entailed by the union of ontology and the current state level. When an operation is placed in the next action level, its effects as well as output data types are applied to the current state level, and thus produce the next state level. Afterwards, mutex links between actions must be evaluated and placed so that they may be used when backtracking through the graph for the solution. Note that the creation of the mutex links should also consider the semantic data types accordingly.

### 7.3.5 Pattern-Based Approach For Loop Generation

The GraphPlan algorithm may generate plans only with sequence and AND-split workflow patterns, such as those discussed in van der Aalst and Hofstede (74). However, loops are also a frequently used pattern. Loop generation (or iterative planning) itself is a difficult and open problem in AI. Much work on iterative planning is based on theorem-proving (Biundo (4)). It is believed by Stephan and Biundo in Stephan and Biundo (70) and other researchers that iterative planning cannot be carried out in a fully automatic way. Levesque (29) proposes a new way that is not tied to proving a theorem, but it is only correct for a given bound or a certain class of simple planning problems.

Here we propose a pattern-based approach for loop generation. It is based on the observation of frequently used patterns of iterations. For example, in the motivation scenario, the order request includes multiple line items (an array of line items) while the addLineItem operation takes as input only one line item. It is obvious that the process needs to iterate all the line items in the order.
request. We may extract the pattern as follows. If an operation has an input message including an element with semantic annotation $SDT_i$ and attribute “maxOccurs” in XML Schema whose value is 1, while the matched (see “satisfy” operator) semantic data type in the current state is from an output message where the corresponding element in that message has “maxOccurs” with value “unbounded” or greater than 1, then a loop is needed for this operation to iterate the array. Our approach avoids the computationally hard problem by restricting possible patterns of loops. The limitation is that the patterns need to be identified and put in the code beforehand.

7.3.5.1 Lifting and Lowering Mechanism of Data Mediation

The data mediation approach is primarily based on the lifting and lowering mechanism presented in SAWSDL (for WSDL and working group (16)). This Section looks in detail of how this lifting and lowering mapping schema functions. The base technique is to convert the message into an intermediate semantic data model and re-convert the semantic data model back into the required specific format. Converting from the message to the intermediate model is known as lifting and the reverse conversion is known as lowering. It is important to note that the data heterogeneities cannot be overcome merely by attaching an ontology reference. These conversions require specific references to templates or other conversion resources in order to carry out the lifting and lowering. Due to the use of XML as the primary message format, the most commonly used intermediate model is also XML and hence the conversion references are often references to XSLT documents.

To understand the importance of this approach rather than the direct use of XSLT to transform between each and every message format consider the following example. Given that there are five heterogeneous (but convertible) messages that require conversion from message A, a direct
conversion technique such as XML transformation, would require ten conversion specifications. The intermediate semantic model if used, would require twelve steps. However, the advantage of the intermediate model can be seen when there is another message added along with A. To process the new message, one has to repeat the mediation process, in case of a direct transformation. The intermediate model enhances the reusability and adds only a lifting and lowering step to the original process. It can be clearly seen that the intermediate model approach is the scalable mediation strategy.

Figure 7.3 we describe the different heterogeneities that can exist between two XML schemas and how such heterogeneities can effect the mediations as discussed in Nagarajan et al. (48)
### Automatic Web Service Composition

#### 7.3. Automatic Web Service Composition

---

**Attribute level Incompatibilities** – differences that arise because of using different descriptions for semantically similar attributes

<table>
<thead>
<tr>
<th>Heterogeneities / Conflicts</th>
<th>Examples - conflicted elements shown in color</th>
<th>Mediatability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Naming conflicts</strong></td>
<td>Web service 1 Student(id, Name) Web service 2 Student(SSN, Name)</td>
<td></td>
</tr>
<tr>
<td>Two attributes that are semantically alike might have different names (synonyms)</td>
<td>Web service 1 Student(id, Name) Web service 2 Book (id, Name)</td>
<td></td>
</tr>
<tr>
<td>Two attributes that are semantically unrelated might have the same names (homonyms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data representation conflicts</strong></td>
<td>Web service 1 Student(id, Name) id defined as a 4 digit number Web service 2 Student(id, Name) id defined as a 9 digit number</td>
<td></td>
</tr>
<tr>
<td>Two attributes that are semantically similar might have different data types or representations</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data scaling conflicts</strong></td>
<td>Web service 1 Marks 1-100 Web service 2 Grades A-F</td>
<td></td>
</tr>
<tr>
<td>Two attributes that are semantically similar might be represented using different precisions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Entity level Incompatibilities** – differences that arise because of using different descriptions for semantically similar entities

<table>
<thead>
<tr>
<th>Heterogeneities / Conflicts</th>
<th>Examples - conflicted elements shown in color</th>
<th>Mediatability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Naming conflicts</strong></td>
<td>Web service 1 Employee (id, Name) Web service 2 Worker (id, Name)</td>
<td></td>
</tr>
<tr>
<td>Semantically alike entities might have different names (synonyms)</td>
<td>Web service 1 Ticket (TicketNo, MovieName) Web service 2 Ticket (FlightNo, Arr. Airport, Dep. Airport)</td>
<td></td>
</tr>
<tr>
<td>Semantically unrelated entities might have the same names (homonyms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Schema Isomorphism conflicts</strong></td>
<td>Source PERSON (Name, Address, HomePhone, WorkPhone) Target PERSON (Name, Address, HomePhone)</td>
<td></td>
</tr>
<tr>
<td>Semantically similar entities may have different number of attributes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Abstraction Level Incompatibility** – Entity and attribute level differences that arise because two semantically similar entities or attributes are represented at different levels of abstraction

<table>
<thead>
<tr>
<th>Heterogeneities / Conflicts</th>
<th>Examples - conflicted elements shown in color</th>
<th>Mediatability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generalization conflicts</strong></td>
<td>Web service 1 Grad-Student (ID, Name, Major) Web service 2 Student (ID, Name, Major, Type)</td>
<td></td>
</tr>
<tr>
<td>Semantically similar entities are represented at different levels of generalization in two Web services</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Aggregation conflicts</strong></td>
<td>Web service 1 Professor (ID, Name, Dept) Web service 2 Faculty (ID, ProfId, Dept)</td>
<td></td>
</tr>
<tr>
<td>Two semantically similar entities, one represented as an aggregate of another</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Attribute Entity conflicts</strong></td>
<td>Web service 1 Course (ID, Name, Semester) Web service 2 Dept (Course, Sem, …)</td>
<td></td>
</tr>
<tr>
<td>Semantically similar entity modeled as an attribute in one service and as an entity in the other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Interoperation between services needs transformation rules (mapping) in addition to annotation of the entities and/or attributes indicating their semantic similarity (matching).

---

Figure 7.3: Different Heterogeneities
Event Identification in SOA

We propose a framework for automatically identifying events as a step towards developing an adaptive middleware for Service Oriented Architecture (SOA). Identifying events that can impact the non-functional objectives of a service request is a key challenge towards realizing a more adaptive services environment. These events can either be user triggered in interactive applications such as mashups or can be triggered by providers themselves. Our approach allows users to capture their requirements in a descriptive manner and uses this description for identifying events of importance. This model is extended to adjust the relevance of the events based on feedback from the underlying adaptation framework. We present an algorithm in Section 8.2 that utilizes multiple ontologies for identifying relevant events and present our evaluations that measure the efficiency of both the event identification and the subsequent adaptation scheme are discussed in Section 8.4.

Businesses increasingly operate in a distributed and dynamic environment where complex intra- and inter-organizational interactions are the norm. Handling the many and varied events that arise during these interactions is a growing challenge. In the dynamic global marketplace, an organization’s success or failure depends on its ability to identify and respond to events arising from all sources, direct and indirect. Nokia and Ericsson, for example, sourced their chips from a common supplier. When a fire affected the supplier’s fabrication unit, Nokia reacted promptly to
lock up all alternate suppliers. Ericsson could not procure chips quickly enough to prevent a major loss of market share that led to the sale and merger of its business with Sony (Sheffi (60)).

The industry has moved toward adopting SOA-based approaches to realize complex business processes. The loosely coupled nature of SOA has driven this movement, but it has been challenging for organizations to model and identify events in the context of SOA. An effective model should allow organizations to capture their functional and non-functional objectives in order to develop strategies to adapt to events that affect their objectives.

In this chapter we present a framework that identifies events that can affect the non-functional objectives of an organization. Two types of events that arise during service execution can be categorized as:

- Event of Direct Consequence (EDC), resulting from an action by the provider or the requester. An example is a delay in shipment during fulfillment of a purchase order.

- Event of Indirect Consequence (EIC), or exogenous event, arising outside the requester-provider framework. A shift in currency exchange values is an example.

Creating a middleware system with the ability to monitor and adapt to both types of events can be viewed as a two-step problem. The first step is for the requestor to identify and subscribe to the events to which the system might need to adapt. The second step is to adapt to those events as and when they occur. Some work addresses the second part of the problem by building systems that adapt to failures when they occur. ADEPT by Reichert and Dadam (56), supports manual adaptation of process schemas, and METEORS (Sheth et al. (63)) supports automatic adaptation.
based on a MDP-based framework. The problem of identifying what events to adapt to, however, has not received much attention.

Our proposed framework builds on current research in semantic Web and semantic Web services to identify both EDCs and EICs. We approach the problem by first modeling the functional and non-functional objectives of a service requester. Our model is based on SAWSDL, the W3C candidate recommendation for adding semantic annotations to Web service description (Verma and Sheth (78)). We extend the notion of semantic templates proposed by Verma (75) to capture the functional and non-functional objectives. A semantic template is a Web service description annotated with data, functional, and non-functional semantics. The data and the functional semantics are captured using SAWSDL. The non-functional semantics are captured using WS-Policy constructs in a manner similar to our work on the use of semantic annotations of WS-Agreement using multiple ontologies including Quality of Service (QoS) and domain ontologies. The annotations in the semantic template refer to a functional and a non-functional ontology.

The functional ontology models the actions of the provider and requester, the relationships between these actions, and events that arise during the execution of these actions (EDCs). The RosettaNet ontology is an example of a functional ontology. The non-functional ontology models the relationships between various non-functional metrics used in the domain of Web services, as well as between exogenous events (EICs) and the Quality of Service metrics. Event-QoS ontology used in this work is an example of a non-functional ontology.

Our proposed framework identifies events by capturing the functional and non-functional re-
quirements of the service requestor using semantic templates in conjunction with a functional and a non-functional ontology. It also computes the relevance of events to the non-functional objectives. Semantic associations reveal complex relationships between entities (e.g., nodes in the graph representing semantic data).

We define and use an extension to the $\rho_{path}$ ontology query operator, first proposed by Anyanwu et al in Anyanwu and Sheth (2) to discover relationships between events and the functional and nonfunctional requirements.

The main contributions of our event identification work are:

- Identification of events using a framework of semantic associations
- Improving accuracy of identification using feedback
- Demonstrating the value of semantic Web in the realization of a rich event-management infrastructure for SOA.

This work was done as a part of the METEOR-S project, which aims to create frameworks to support the complete life cycle of services and to enhance standards and specifications already widely used. True to this philosophy, our functional ontology for the e-commerce domain is created from the RosettaNet standard for global supply chain management and the non-functional ontology is created by extending the OWL-QoS ontology, a collaborative effort by researchers to model the Quality of Service metrics for Web services in OWL.

http://www.comp.lancs.ac.uk/computing/users/dobsong/owl_qos/index.php/Main_Page
8.1 Overview of the Framework for Event Identification

8.1.1 Overview of the Functional and Non-Functional Ontologies

The functional ontology used in the framework is based on the RosettaNet standard for supply chain. This ontology models the relationships between the RosettaNet Partner Interface Processes (PIPs) specified in segments 3A and 3B of the order management cluster of the RosettaNet specification. The main classes of importance to us in this ontology, in the context of this work, are Act_PIP, Notify_PIP and Event. The Act_PIP class models the actions that fulfill a request. Members of the Act_PIP are used in semantic annotation of operations in the semantic template. The RequestPurchaseOrder PIP (PIP3A4) is an example of Act_PIP. The Notify_PIP models the various actions that notify of events. The Notify_of_Shipments_Tendered_PIP (PIP 3B6) is an example of Notify_PIP. The Event class models the various events. The members of the Event class model the EDCs. An example of an event is delay in Shipment.

The non-functional ontology used in the framework is an extension to the OWL-QoS ontology for Web services. The OWL-QoS ontology is an effort to integrate the different QoS ontologies available for Web services. We extended this ontology to include events and associate these events with the nonfunctional metrics modeled in this ontology. We added events from industrial use cases such as the Nokia-Ericsson scenario. The EICs were added by studying the PIPs in the Inventory management Cluster of RosettaNet. In addition to the EICs, all the EDCs modeled in the functional ontology were also added into the nonfunctional ontology. The QoS concepts from this ontology are used to annotate the assertion constraints in the semantic template. Adding the EDCs to the nonfunctional ontology is essential to our approach, since we want to compute the
relevance of the EDCs to the various nonfunctional objectives captured in the semantic template, discussed in section 2.1.3.2.

8.1.2 Outline of the Proposed Framework

We sketch an outline of the proposed framework to identify relevant events for a given operation in the semantic template. We first identify all EDCs by querying the functional ontology for the given operation. Then we identify all exogenous events (EICs) that affect the nonfunctional metrics in the effective policy of that operation. Querying the nonfunctional ontology identifies the EICs. The relevance of each event in the set of events identified to the non-functional metrics in the effective policy of the operation is computed. Based on the computed relevance, the set of relevant events is created. Figure 8.1 illustrates our approach to event identification. However it is possible for events that were either not identified or thought irrelevant to occur during the fulfillment of this request. It is also likely that an event previously identified as relevant becomes irrelevant. In both cases, the framework will adjust the relevance of the event after receiving feedback from either a human or the underlying adaptation mechanism. After each feedback cycle, the set of relevant events is recomputed. The framework is explained in detail in Section 8.3.1.

8.2 Defining the $\rho_{path}^{\text{path}}$ Query Operator

A path $p_{e_i \rightarrow e_j}$ in an ontology graph is a collection of intermediate entities (vertices) and relationships (edges) between the entities, such that $e_j$ can be reached from $e_i$ by traversing the intermediate entities and relationships. The $\rho_{path}$ operator defined in (2) queries an ontology to
8.2. DEFINING THE $\rho_{PATH}$ QUERY OPERATOR

Figure 8.1: Identifying Events from semantic Template

discover semantic associations between different entities. The semantic association returned is represented as a path between the entities in the ontology. There can be more than one association between two entities. In such cases, the $\rho_{path}$ operator would return a set of paths. $\rho_{path}$ is defined as $\rho(e_i, e_j) = \{p_{e_i \rightarrow e_j}\}$. For example in the snapshot of the functional ontology in Figure 8.1, $\rho_{path}(\text{RequestPurchaseOrder, ShipmentConfirmation})$ would return the following two associations:

- RequestPurchaseOrder $\rightarrow$ is_followed_by $\rightarrow$ QueryOrderStatus $\rightarrow$ has_notification $\rightarrow$ Notify_Shipment $\rightarrow$ notifies_event $\rightarrow$ ShipmentConfirmation (example 1)

- RequestPurchaseOrder $\rightarrow$ has_notification $\rightarrow$ Notify_Shipment $\rightarrow$ notifies_event $\rightarrow$ ShipmentConfirmation (example 2)
In the context of our work, we seek to identify all events that have semantic associations with the functional requirements in the functional ontology and non-functional requirements in the non-functional ontology. Hence we modified the $\rho_{path}$ operator to query between an entity and a class. This is defined as

$$\rho(e_i, C) = \{ p_{e_i \rightarrow e} : e \text{ is a member of } C. \}$$  \hspace{1cm} (8.1)

While $\rho(e_i, e_j)$ returns all associations between two entities $e_i$ and $e_j$, $\rho(e_i, C)$ returns a collection of paths between $e_i$ and all members of $C$. A class-relationship($\mathcal{C}$) constraint is a collection of classes and relationships in the ontology. We define a class relationship constraint as

$$\mathcal{C} = \{ C,R \text{ where } C \text{ is a set of classes and } R \text{ is a set of relationships in the ontology.} \}$$  \hspace{1cm} (8.2)

The following example describes an example $\mathcal{C}$ constraint, defined on the snapshot of the functional ontology in Figure [8.1]

$$\{\{\text{Act}_{-}\text{PIP}, \text{Notify}_{-}\text{PIP}, \text{Event}\},$$

$$\{\text{has}\_\text{Notification}, \text{notifies}\_\text{event}\}) \text{ (example 3)}$$

A semantic association $p_{e_i \rightarrow e_j}$, is said to satisfy the constraint $\mathcal{C}$, if every intermediate entity $e_m$, in the path is a member of some class in the set of classes in $\mathcal{C}$ and every intermediate relationship $r_n$.
is a member of some relationship in the set of relationships in \( \mathcal{C} \). This can be formalized as,

\[
\{ e_i \rightarrow e_j \} \models \mathcal{C} \text{ if } \forall (e_m, r_n) \in \{ e_i \rightarrow e_j \},
\]

\[
(\exists C_k \in C : e_m \text{ member of } C_k) \land \nabla \tag{8.3}
\]

\[
(\exists R_l \in R : r_n \text{ member of } R_l)
\]

where \((C, R) \in \mathcal{C}\)

The semantic association mentioned in example 2 is an example of an association that would satisfy the \( \mathcal{C} \) constraint given in example 3. Path length of a semantic association, \( l(\{ e_i \rightarrow e_j \}) \) is the number of intermediate relationships between \( e_i \) and \( e_j \) in the path.

A Bounded \( \mathcal{C} \) (\( \mathcal{B} \)) constraint is defined on a set of associations. \( \mathcal{B} \) is same as the \( \mathcal{C} \) constraint defined in Equation 4, with an additional constraint on the maximum path length. This maximum path length is the limiting path length of \( \mathcal{B} \) and is denoted by \( \mathcal{L} \).

\[
\mathcal{B} = \{ C, R, \mathcal{L} \text{ where } C \text{ is a set of classes, } \]

\[
R \text{ is a set of relationships in the ontology, } \tag{8.4}
\]

\[
\text{and } \mathcal{L} \text{ is the limiting path length}\}.
\]

An example of \( \mathcal{B} \) constraint is given in example 4 below.

\[
\mathcal{B} = (\{ \text{Act\_PIP}, \text{Notify\_PIP} \}, \{ \text{is\_followed\_by}, \text{notifies\_event} \}, 2). \text{(example 4)}
\]

where Act\_PIP and Notify\_PIP are classes in the functional ontology and is\_followed\_by and notifies\_event are relationships in the functional ontology, as noted earlier and 2 is the limiting
A association \( p_{e_i \rightarrow e_j} \), is said to satisfy the constraint \( \mathcal{B} \), if it satisfies \( \mathcal{C} \) as defined in Equation 4 and its path length is less than or equal to the limiting path length of \( \mathcal{B} \). This can be formalized as,

\[
\begin{align*}
    p_{e_i \rightarrow e_j} &\models \mathcal{B} \text{ if } (p_{e_i \rightarrow e_j} \models \mathcal{C}) \land \\
    l(p_{e_i \rightarrow e_j}) &\leq L.
\end{align*}
\]

The association described in example 2 satisfies the \( \mathcal{B} \) constraint mentioned in example 4. Bounded Constrained \( \rho_{\text{path}}^\mathcal{B} \): The Bounded Constrained \( \rho_{\text{path}} \) query operator is a modification of the \( \rho_{\text{path}} \) query operator, such that every association \( p_{e_1 \rightarrow e_2} \) returned by the query satisfies the \( \mathcal{B} \) constraint as defined in Equation 6.

\[
\begin{align*}
    \rho_{\text{path}}^\mathcal{B}(e_i, C) &= \{ \rho_{\text{path}}(e_i, C) : \forall p_{e_i \rightarrow e_j} \in \rho_{\text{path}}(e_i, C), \\
    p_{e_i \rightarrow e_j} &\models \mathcal{B} \}. 
\end{align*}
\]

### 8.2.1 Event Identification using the \( \rho_{\text{path}}^\mathcal{B} \) query operator:

We first identify the EDC’s of a given operation from the semantic template. We recall here that the EDC’s are members of the Event class and the semantic annotations on the operation in a semantic template are members of the Act_PIP class, in the functional ontology. These annotations are captured in the semantic template as model references. An event \( E \), can arise because of an action \( \omega \) in two scenarios: 1) \( \omega \) generates \( E \) and 2) \( E \) is generated by another action, which is executed
as a part of the invocation of $\omega$. In both these cases, there is a semantic association between the event and the model reference ($M_r^{\omega}$) of the action. To identify the EDC’s with respect to a given operation ($\omega$) in the semantic template, we get the model reference ($M_r^{\omega}$) of $\omega$. We construct a $\rho_{path}^{\omega}$ with the model reference and the ontology class, Event as parameters. The $\mathcal{B}$ constraint is defined as

$$\mathcal{B} = (\{\text{Act\_PIP, Notify\_PIP, Event}\},$$

$$\{\text{generates\_event, is\_followed\_by,}\}$$

$$\text{notifies\_event}, n). \quad (8.7)$$

In the above constraint, the limiting path length is set to a variable. The effect of varying the limiting path length is discussed in the evaluation. Having defined the $\mathcal{B}$ constraint we now proceed to compute the set of EDC’s using the following definition.

$$EDC_\omega = \{e: \rho_{path}^{\omega}(M_r^{\omega}, \text{Event}) \text{ is not NULL, where}$$

$$\mathcal{B} \text{ is defined using } 11\} \quad (8.8)$$

The set of EIC’s is computed in a similar manner. The set of EIC’s are defined on the effective policy ($\pi_{eff}$) of the given operation, $\omega$. In order to compute the effective policy, we first get the term policy $\pi_\omega$, of the template term containing the operation. We get the template level policy, $\pi_s$ from the semantic template which contains $\theta$. Effective policy is then computed using 2. From the effective policy, we identify the model references, $M_\alpha^{\omega}$ modeled in the assertions, $\alpha$. We recall here that these model references are modeled in the non-functional ontology as QoS concepts. The
Qos concepts in the non-functional ontology are categorized into three types: 1) Business level 2) Service level and 3) System level concepts. The business level concepts model all the QoS metrics that affect the business level guarantees. Examples of this include SupplyTime and OrderCost. The service level concepts model the QoS metrics concerning service implementation and performance. Examples of this include Reliability and Security. The system level QoS concepts model the QoS metrics at the system level. Bandwidth and NetworkTime are a couple of examples of this. We define a \( \mathfrak{B} \) constraint on the nonfunctional ontology for each of the three categories.

\[
\mathfrak{B}_{\text{business}} = (\{\text{Time, Cost, Event}\},
\{\text{is\_part\_of, consists\_of,}
\text{is\_a\_component\_of}\}, n).
\]

\[
\mathfrak{B}_{\text{service}} = (\{\text{Reliability, Security, Transactions, Event}\}, \{\text{is\_part\_of,}
\text{is\_a\_component\_of, consists\_of}\}, n).
\]

\[
\mathfrak{B}_{\text{system}} = (\{\text{Resource, Availability, Event}\},
\{\text{is\_part\_of, is\_a\_component\_of,}
\text{consists\_of}\}, n).
\]

We compute the EIC’s in three parts. We compute the business level EIC’s, service level EIC’s and system level EIC’s by querying the functional ontology \( \rho^{PATH} \) query operator. These are computed
as defined below.

\[
EIC_{(\omega,\text{business})} = \{ e : \rho_{\text{path}}(M^\alpha_r, \text{Event}) \neq \text{NULL} \}
\]

\[
EIC_{(\omega,\text{service})} = \{ e : \rho_{\text{path}}(M^\alpha_r, \text{Event}) \neq \text{NULL} \}
\]

\[
EIC_{(\omega,\text{system})} = \{ e : \rho_{\text{path}}(M^\alpha_r, \text{Event}) \neq \text{NULL} \}
\]

(8.10)

The events of indirect consequence for an operation are then identified as \( EIC_\omega = EIC_{(\omega,\text{business})} \cup EIC_{(\omega,\text{service})} \cup EIC_{(\omega,\text{system})} \). Once EDC’s and EIC’s are identified, we define the set of Identified events for \( \omega \) (\( \mathcal{E}_\omega \)), as

\[
\mathcal{E}_\omega = EIC_\omega \cup EDC_\omega.
\]

(8.11)

### 8.3 Computing event relevance

The set \( \mathcal{E} \), is the set of all events that we have identified. However, not all events will be relevant to the non-functional objectives of the requestor. To identify the relevant events, we define the relevance metric \( \chi \) between an event \( e \), and a non-functional metric, \( M^\alpha_r \). The farther an event \( e \) is from a non-functional metric, the lesser is the relevance of the event to that non-functional metric.

This is the intuition behind defining the relevance function. For an event \( e \) and a non-functional
metric $M^\alpha_r$, the relevance is defined as,

$$\chi(e, M^\alpha_r) = 1 - \left( \frac{P^S_{e\rightarrow M^\alpha_r}}{L, \text{the limiting path length of } \mathcal{B}} \right),$$

where $P^S_{e\rightarrow M^\alpha_r} \in \rho_{\text{path}}(e, M^\alpha_r)$ is the shortest path. (8.12)

$\chi(e, M^\alpha_r)$ gives the relevance of one event e to one non-metric $M^\alpha_r$. However we need to compute the relevance of each event across all non-functional metrics. To do that, we first construct a n X m matrix, where n is the total number of events identified and m is the non-number of non-functional metrics in the effective policy, called the relevance matrix ($\mathcal{M}$). Each element $M_{ij}$ in the matrix has the relevance of event $e_i$ to the non-functional metric $M^\alpha_{r(j)}$. We will illustrate with an example.

Consider the following $\rho_{\text{path}}$ query, defined on the non-functional ontology. We define $\mathcal{B}$ for this query to be $\mathcal{B} = (\{\text{Time, Event}\}, \{\text{is\_related\_to, has\_effect\_on, is\_a\_component\_of}\}, 5)$. Here 5 is the limiting path length. The query that is constructed using the $\mathcal{B}$ constraint is $\rho_{\text{path}}(\text{Event, SupplyTime})$.This query returns two associations:

- $\text{Inventory\_Drop} \rightarrow \text{is\_related\_to} \rightarrow \text{ProductionDelay} \rightarrow \text{has\_effect\_on} \rightarrow \text{Production\_Time} \rightarrow \text{is\_a\_component\_of} \rightarrow \text{SupplyTime}$

- $\text{ShipmentDelay} \rightarrow \text{has\_effect\_on} \rightarrow \text{Shipment\_Time} \rightarrow \text{is\_a\_component\_of} \rightarrow \text{SupplyTime}$.

The relevance of the event ShipmentDelay to the non-functional metric supply time,

$$\chi(\text{ShipmentDelay, SupplyTime}) = 1 - \left( \frac{2}{5} \right) = 0.6.$$
Similarly we can compute the relevance of the event *Inventory_Drop* on the constraint Supply-Time to be 0.40. The part of the relevance matrix depicting these values is shown below:

\[
\begin{pmatrix}
\text{Inventory}_\text{Drop} & \text{ShipmentDelay} & \ldots \\
\text{supplyTime} & 0.40 & 0.60 & \ldots \\
\end{pmatrix}
\]

The cumulative relevance \( \chi_{(e_i, \pi_{eff})}^C \), is defined as the relevance of the event \( e_i \) to all the non-functional metrics in the effective policy of a given operation \( \omega \). For example, the cumulative relevance measures the impact of a shipment delay across all the non-functional metrics in the effective policy. An event which has a higher cumulative relevance, is more important than an event that has a lower cumulative relevance. This is computed as,

\[
\chi_{(e_i, \pi_{eff})}^C = \sqrt{\frac{\sum_{j=0}^{m} (\mathcal{M}_{ij})^2}{m}}
\]

(8.13)

The set \( \chi(\mathcal{E}_\omega, \pi_{eff}) \) is a set of the cumulative relevances of all the events in \( \mathcal{E}_\omega \) to the non-functional metrics in the effective policy. \( \chi_{\text{max}} \) maximum cumulative relevance value in this set and \( \delta_\chi \), denotes the standard deviation of the cumulative relevance values. Cutoff relevance \( (r) \) is computed as,

\[
r = \chi_{\text{max}} - \delta_\chi.
\]

(8.14)

Based on the earlier definitions, set of relevant identified events \( (\mathcal{E}_\omega^R) \), is defined as a collection of all events \( e \), such that the \( \chi_{(e, \pi_{eff})}^C \), defined in Equation 15, is greater or equal to the cutoff relevance. All the other events, that do not belong to set are classified as non-relevant events and the set of non-relevant events is denoted by \( \mathcal{E}_\omega^N \). The relevance status \( (s) \) of an event indicates if the
framework identifies that event to be relevant or not. The relevance status of an event is 1 if the event is identified as relevant and 0 otherwise.

### 8.3.1 Adjusting the relevance based on feedback

After the events are identified, it may happen that an event belonging to $E^R$, may not after all be relevant. On the same note, it is also possible that an event in $E^N$ is actually relevant. To address these issues, we use a feedback-based mechanism, which adjusts the cumulative relevance. This feedback can be come from a human or can also be obtained by observing the behavior of the underlying adaptation framework, when the event happens. The feedback mechanism for improving the accuracy of the identified events uses an adjustment scheme to adjust the relevance of an event.

One approach to improving the efficiency is when a human validates the set of relevant events identified by the system and gives a feedback, if there are any events of non-relevance that were identified as relevant and if the set of events identified as relevant is complete. Another approach is to monitor the behavior of the underlying adaptation framework, if that is possible. The intuition in this approach is that, if the adaptation framework does indeed adapt to an event, then it is relevant. The feedback that is obtained is captured in the feedback status ($f$). The feedback status of an event is 1 if the event is considered relevant by the entity providing the feedback (Either a human or the adaptation framework) and 0 otherwise. We define an adjustment metric called $\Delta$. $\Delta$ is the numerical adjustment made to the cumulative relevance $\chi(e_i, \pi_e, f)C$, of an event $e_i$, based on the feedback. If the feedback status of an event is 0, but the relevance status of that event is 1, then the cumulative relevance of that event is decremented by $\Delta$. If the feedback status of an event is 1
and the event is identified as non-relevant by our framework, then the cumulative relevance of the event is incremented by $\Delta$. We define the adjustment on the cumulative relevance as,

$$
\chi_C^{(e_i, \pi_{e,f f})} = \chi_C^{(e_i, \pi_{e,f f})} - \Delta, \text{ if } s(e_i) = 1 \text{ and } f(e_i) = 0
$$

$$
\chi_C^{(e_i, \pi_{e,f f})} = \chi_C^{(e_i, \pi_{e,f f})} + \Delta \left( \frac{i}{n} \right), \text{ if } s(e_i) = 0 \text{ and } f(e_i) = 1,
$$

where $i$ is the total number events with feedback status 0 and $n$ is the total number events with relevance status 0. (8.15)

When making an adjustment to the of non-relevant events, we multiply $\Delta$ by a factor of $\left( \frac{i}{n} \right)$. This keeps the the mean value of the set of cumulative relevance numbers constant. This is essential in order to ensure that the changes to the value of the cutoff relevance for relevance is negligible after each feedback iteration. After the cumulative relevance values are adjusted, the cutoff relevance $(r)$ is recomputed using Equation 16. and the set of relevant events $E^R_\omega$ is identified as before. This adjustment technique is called “Fixed Adjustment”.

Another strategy to incorporate the feedback is to adjust the recalculated cutoff relevance value, in addition to changing the cumulative relevance values. Apart from adjusting the values using Fixed Adjustment, the $\Delta$ change is applied to the cutoff relevance as well. The cutoff relevance of $E^R_\omega$ is adjusted from $r$ to $r - \Delta$. After each adjustment, we compute the entropy of the system.

The entropy of the system is the number of events whose relevance status has changed from non-relevant to relevant, after the adjustment. These events belonged to the set of non-relevant events before the adjustment but belong to the set of relevant events after. The framework stops asking for a feedback, once the entropy is zero.
The performance of both these adjustment schemes with respect to the time it takes to identify all the relevant events and the percentage of non-relevant events identified during the process is discussed in the evaluation Section. It is however important to note here that the choice of delta is very important in both the approaches. If $\Delta$ is very small, it may not have a significant impact in the computation of cumulative relevance. A large $\Delta$ on the other hand, can make an event oscillate between relevance and irrelevance. Hence the choice of $\Delta$ is important. In the next Section we discuss the evaluation of our framework.

8.4 Evaluation

In this Section we present the empirical evaluations of our system. The experiments in this evaluation, demonstrate the ability of the framework to identify relevant events and to adjust the relevance based on a feedback. The objective of this evaluation is three-fold.

- To study the increase in the accuracy of the system before and after making adjustments based on the feedback,
- To measure the performance of the event identification and feedback-based adjustment scheme, for each of the two approaches discussed and
- To measure the accuracy of the framework in identifying the relevant events, for each of the two feedback approaches mentioned.

The experimental setup consisted of the relevance matrix, a feedback component and our event identification framework. The relevance matrix is created by computing the semantic associations
between the events and non-functional metrics in the OWL-QoS ontology, using the $\rho_{\text{path}}$ query operator. The constraints used in this query are described in Equation 11. The feedback component simulates the human feedback and is aware of all the events that are relevant to a given operation. Given an event, the feedback component will return the feedback status of that event. We here recall that the feedback status of an event is 1 if the event is relevant and 0 otherwise. We study the performance and the accuracy of the system, using this experimental setup by varying the following parameters: a) the constraining path length in the query, b) the total number of events and c) the percentage of relevant events to the total number of events.

Our experiment describes the accuracy of the system before and after making feedback-based adjustments. The accuracy of the system is defined by the ratio of relevant events identified to the actual number of relevant events in the feedback component. In this experiment we study the variation in accuracy with respect to the variations in the constraining path length. The results are illustrated in Figure 8.2. The experiment demonstrates the direct relevance of the constraining path length to the accuracy of the event identification technique. The accuracy of the system improved with the increase in the constraining path length, both before and after feedback-based adjustments. The system was able to identify all the relevant events, after the feedback-based adjustment, when the constraining path length was 6. The impact of the feedback-based adjustment is also demonstrated in this experiment. For a constraining path length of 3, the accuracy of the system improved from 20% before the adjustment, to about 45% after the adjustment. This experiment achieves the first objective of our evaluation. Our next set of experiments study the performance of our system with respect to the number of feedback iterations required before the set of identified events stabilized. In all the subsequent experiments, we use a constraining path
In the next experiment, we study the performance of our system by varying the total number of events that are identified. We varied the total number of events identified from 100 to 900. The performance of our framework is illustrated in Figure 8.3. It can be seen from Figure 8.3 that the number of iterations taken to reach the stability is more for the fixed approach than the hybrid scheme for adjustment. In the hybrid approach, the value of the cut-off relevance is reduced after each feedback iteration, increasing the number of events in the set of relevant events. The number of iterations taken by each adjustment scheme is almost constant with change in the total number of events. This is because both relevant set computation and feedback-based adjustment are independent of the number of events.

Figure 8.3: Studying the performance of hybrid and fixed Adjustment schemes with variation in the total number of events.
Our next experiment studies the performance of the system. In this experiment the percentage of relevant events to the total number of events is changed and the impact of varying this on the number iterations is studied. As illustrated in Figure 8.4, the time to reach stability increases with the increase in the percentage of relevant events. This can be attributed to two reasons: 1) the higher percentage of relevant events can significantly reduce the number of events with a feedback value of 0 in the set of relevant events, and 2) as the percentage of relevant events increases, the chances of more events which are classified as non-relevant by the framework, being actually relevant goes up. When the number of events with a feedback value of 0 reduces, the factor in the adjustment scheme described in Equation 17, also reduces and this in effect lessens the change in the cumulative relevance of the events in the non-relevant set after each feedback iteration. The reduced change in the value of the cumulative relevance coupled with the increased number of events classified as non-relevant by the framework, being actually relevant, increases the iterations needed to stabilize the system.

![Figure 8.4: Performance of hybrid and fixed adjustment schemes with variation in the percentage of relevant events](image)

The hybrid approach gives a better performance. However the variation to the cut-off relevance can have an impact on the accuracy of the system. Our next set of experiments study the accuracy of the system with respect to variations in the number of events and the percentage of
relevant events. The fourth experiment measures the variation in accuracy of both the feedback schemes when the total number of events is changed. This is illustrated in Figure 8.5. The drop in the precision of the hybrid approach with the increase in the total number of events is more pronounced than that of the fixed approach. The adjustment made to the cutoff relevance in the hybrid approach is responsible for this. This adjustment makes it possible for an event identified as non-relevant by the feedback to manager to be identified again as relevant by the framework.

The following example illustrates an anomaly in the hybrid approach to feedback-based adjustment. For the sake of this example, consider an event $e_i$ with cumulative relevance of 0.53. Let the cut-off relevance to be 0.5. Let $\Delta$ be 0.05. The framework chooses $e_i$ as a relevant event. After the first feedback iteration, $e_i$ is identified as non-relevant and its relevance is reduced to 0.48, by Equation 17. The set of relevant events is computed again. After the computation, let us assume for the sake of this example, the new cut-off relevance to be 0.52. The adjusted relevance value of $e_i$ is less than the cut-off relevance, making $e_i$ a non-relevant event. However, in the hybrid approach we adjust the value of the cutoff relevance by $\Delta$, reducing its value to 0.47. Now when the set of relevant events is computed using this adjusted cutoff value, $e_i$ would be identified as relevant by the framework, though the feedback indicated otherwise. This anomaly affects the accuracy of the hybrid approach.
Our last experiment measures the accuracy of our framework by varying the percentage of relevant events keeping the total number of events constant. The results are presented in Figure 8.6. As we can see from the results, the accuracy of both the approaches increases with the increase in the number of relevant events. However, the hybrid approach shows more improvement. This is because, with the increase in the percentage of relevant events, the number of events with lower cumulative relevance value that are actually relevant are more. This explains the improvement shown by the hybrid approach.
Conclusion

Despite their growing adoption, the stated objectives of Service Oriented Architecture, namely interoperability, agility, and flexibility have been hard to achieve. In this dissertation, we study the various problems that are currently the roadblocks towards achieving these objectives. Our research presents a systematic and holistic approach towards realizing the goals, by addressing the problems of searching and ranking services, data mediation and composition. In doing so, we demonstrate the value of employing semantic Web techniques such as domain model creation, semantic annotation and reasoning in each of the different problems.

Conceptually, our research is one of the earliest attempts to provide an integrated approach towards reconciling two different approaches to services. We adopt ideas from multiple areas, such as artificial intelligence, text mining and analytics, and software engineering. In addressing the problem of searching and ranking of services in chapter [4], we provide a two-fold solution. Our search engine based approach for the popular consumer services is one of the first attempts towards finding service APIs. The idea of serviut rank for ranking APIs based on their utilization, discussed in Section [4.5] implicitly acts as a quality of service measure. While much of the work in the area of service selection has been towards reasoning and semantic matching based on inputs and outputs, we adopt an approach that involves non-functional attributes as well. Further, our
The hierarchical matching algorithm discussed in Section 5.4 lays emphasis on efficiency and this is demonstrated in our evaluation.

Similar to our contributions in the area of searching and ranking of services, our research in the area of data mediation takes an alternate route to existing approaches. The area of data mediation and integration has been extensively studied, with much of the focus surrounding automatic mediation. However, the current state of research in automatic mediation leaves a lot to be desired. Chapter 6 discusses a metric called mediatability that aids the end user in data mediation. Our experience shows that such a metric would be very useful, especially in the context of developing hybrid Web applications or mashups.

One of the biggest barriers towards realizing composition is the idea of data mediation. While conventional planning approaches, to some extent, automate the task of process creation, they fall short in addressing data mediation. Our research adopts a novel mediation-as-a-service paradigm and incorporates mediating services as a part of the composition, discussed in Chapter 7. Further, we also adopt a declarative approach, where an end user can model his goals in a richer manner.

The final contribution of this dissertation is the automatic identification of events during the execution of a services centric software. Chapter 8 presents a classification of events that arise during execution. We extend the basic semantic operations discussed by Anyanwu and Sheth (2), and adapt it to discovering events. We also discuss a feedback based approach for changing the relevance of an event based on the reaction of the underlying system.

Much of the work discussed in this dissertation owe their origins to research in the area of semantic Web services at the Large Scale Distributed Systems Lab at the University of Georgia and at the kno.e.sis center at Wright State University. In addition to the research discussed, we
have also contributed to services specification, data mediation and event identification.

An emerging area of research in which many findings of this research can be applied is social computing. The data integration and composition approaches can be employed in social data integration, especially in the context of citizen sensor services. Nagarajan et al. (45) provide an early evidence for the potential application of this research in social computing. We envision to explore this area further in future.
Bibliography


