# EFFICIENT WEB SERVICES DISCOVERY AND COMPOSITION

**YANAN HAO**

**DOCTOR OF PHILOSOPHY**

2009

**VICTORIA UNIVERSITY**

## DECLARATION

I, Yanan Hao, declare that the PhD thesis entitled *Efficient Web Services Discovery and Composition* is no more than 100,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work.

Signature                                                                 Date

# PUBLICATIONS

1. **Yanan Hao**, Yanchun Zhang, "A Two-phase Rule Generation and Optimization Approach for Wrapper Generation". *Proceedings of the 17th Australasian Database Conference* (ADC'06), pp.39-48. Hobart, Tasmania, Australia, January 16-19 2006.

2. **Yanan Hao,** Yanchun Zhang, "Web Services Discovery Based on Schema Matching". *Proceedings of 23rd Australasian Computer Science Conference* (ACSC'07), pp. 107-113. Ballarat, Australia, 29 Jan - 2 Feb, 2007.

3. **Yanan Hao,** Yanchun Zhang and Jinli Cao, "WSXplorer: Searching for Desired Web Services", *Proceedings of the 19th International Conference on Advanced Information Systems Engineering* (CAiSE'07), pp.173-187. Trondheim, Norway, 11-15 June 2007.

4. Hongcan Yan, Dianchuan Jin, Lihong Li, Baoxiang Liu and **Yanan Hao**, "Feature Matrix Extraction and Classification of XML Pages", *Proceedings of APWeb 2008 International Workshops*, pp.210-219. Shenyang, China, April 26-28, 2008.

5. **Yanan Hao**, Jinli Cao and Yanchun Zhang, "Efficient IR-Style Search over Web Services", *Proceedings of the 21st International Conference on Advanced Information Systems Engineering* (CAiSE'09), pp.305-319. Amsterdam, The Netherlands, 8-12 June 2009.

6. **Yanan Hao**, Yanchun Zhang, "Efficient IR-Style Search over Web Services". Submitted to *Future Generation Computer Systems*.

7. **Yanan Hao**, Yanchun Zhang, "A Relaxation-Based QoS Model and Computation Framework for Service Selection". Submitted *EDBT/ICDT 2010 Joint Conference*.

# ACKNOWLEDGMENTS

First and foremost, I would like to express my deep gratitude to my supervisor, Professor Yanchun Zhang, for his exceptional support, encouragement, patience during all stages of this dissertation.  This dissertation would not have been possible without his invaluable advice and guidance. I feel so lucky to have Prof. Zhang as my supervisor. I owe a special debt to him forever.

I also want to thank my co-supervisor, A/Prof. Yuan Miao for his constant help, discussion, and many constructive suggestions throughout my doctoral study. I have had the great pleasure to work with my colleagues in Centre for Applied Informatics Research (CAI) at Victoria University. I would like to thank them for their valuable suggestions and discussions during the process. I thoroughly enjoyed their fruitful collaboration, and I gained invaluable skills by working with them.

I am grateful to the School of Engineering and Science for supplying very good research environment, and the staff members who offered countless help, particularly, Ponnusamy Rajendran, Danh Ho, Mark Mojic, the School Postgraduate Coordinator Dr. Gitesh Raikundalia, Janet Grady, A/Prof. Xun Yi, Dr. Bai-ling Zhang, Dr. Fuchun Huang, Professor Pietro Cerone, A/Prof. Hao Shi, and the Scholarship Coordinator of Office for Postgraduate Research, Ms. Lesley Birth.

Special thanks go to Dr. Jinli Cao for her trust, support and recommendation. I appreciate her help forever.

I wish to extend my deepest gratitude to my parents and my little sister for their love, support and encouragement throughout my entire life. I thank my be-loved wife, Lina Wang. Without her endless love and support, I could not possibly have reached this far. I would also like to thank my daughter, Qihe Hao, for being there and making everything I do meaningful.

# ABSTRACT

As an emerging cross discipline area for distributed computing, *Service-Oriented Computing* (SOC) paradigm promises to bridge the gap between Business Services and IT Services, enable technology to help people perform business processes more efficiently and effectively, and enable businesses and organizations to collaborate globally through standard services. With the rapid development and popularity of Internet and e-commerce, business organizations are investigating ways to expose their current software components into web services so as to make use of distributed service computing resources. Business organizations are also investigating ways on how to incorporate services running on different platforms and hosted by service providers outside of their boundaries into more complex, orchestrated services.

This conceptually new approach brings multiple issues to researchers. As the number of web services and SOC applications increases, there is a growing need for mechanisms for discovering services efficiently. Effective mechanisms for web services discovery and ranking are critical for organizations to take advantage of the tremendous opportunities offered by web services, to engage in business collaborations and service compositions, to identify potential service partners, and to understand service competitors and increase the competitive edge of their service offerings.

Apart from functional specification required by users during service discovery, some

non-functional requirements such as *Quality of Service* (QoS) and transactional properties are also major concerns, because users need to select and compose web services not only according to their functional requirements but also to their transactional properties and QoS characteristics defined using a quality model. On one hand, in service-oriented environments multiple web services may provide similar functionality, but may offer different non-functional properties (e.g., price). When selecting existing web services from web service communities to generate composite services, the number of composed services may be very large. Therefore, it is a challenge on how to select appropriate web services to satisfy users' global QoS requirements based on a set of given QoS preferences. On the other hand, as web services operate in a highly dynamic distributed environment and interact with each other, the possibility of unexpected behavior is high. The unexpected behavior from an individual service may bring negative impact on all the component services in the composition, even lead to failure of the running of the composite service. Thus we need a mechanism that provides transactional support to service composition to ensure the overall consistent and reliable execution of business processes.

In this thesis, we investigate the problem of efficient web services discovery and composition in service oriented environments. Firstly, we present an efficient IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. We introduce the notion of *preference degree* for a web service, and suggest *relevance* and *importance* as two desired properties for measuring its preference degree. Also, various algorithms are given to obtain service relevance and importance. The key part for computing service importance is a new schema tree matching algorithm, which catches not only structures, but even better semantic information of schemas defined in web services. Moreover, we develop an approach to identify associations between web-service operations based on service operations matching. This approach uses the concept of *attribute closure* to obtain sets of operations. Each set is composed of associated web-service operations. Experimental results show the proposed IR-style search strategy is efficient and practical.

Secondly, we propose a novel QoS model for performing flexible service selection in web service composition with QoS constraints. The key idea of the model is to relax users' QoS constraints and try to find the most possible services satisfying users' QoS requirements. Based on the proposed QoS framework, we develop various algorithms for making service selection on individual and composite services. We also introduce a top-*k* ranking strategy to reflect users' personalized requirements. We evaluate the performance of the model in terms of time and space cost.

Finally, we propose a strategy to verify at design time whether a service composition can be implemented by a set of selected web services according to the transactional requirements specified by a user. We define transactional properties for component web services and analyze the dependencies among them. Next, based on the *Automata theory*, we model the transactional behavior of both component web services and composite services as transition systems, and use *Accepted Termination States* (ATS) to describe the correct execution of composite services. Then, by analyzing the *message exchanges* between component services of a composite service we present a novel algorithm to verify all possible state transitions of the composite service to ensure consistent termination states according to the ATS specified by the user. In addition, for the cause of efficiency we use Temporal Logic to describe the transactional properties of a composite service, and then we employ *SPIN*, an automata-based model checker to carry out automated verification of temporal logic properties on the composite service.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1

# INTRODUCTION

The Internet is one of the most important inventions in the world. It has already made a profound impact on the way people work, live and communicate. Thanks to its basic features such as widespread usability and access, individuals and organizations can easily than ever publish and share information, interact with each other. The Internet has also redefined how businesses operate. Realizing that the Internet has become a large market and has the advantage of low-cost and high efficiency, more and more companies today are engaging in business through the Internet, also known as e-commerce. With the increasing number of e-commerce between companies, there is a strong demand for automating business activates in e-commerce applications and making them run efficiently, effectively and reliably. Web service technology provides a solution for these challenges. Our research is motivated by the need to facilitate the discovery and composition of web services effectively.

This chapter is organized as follows. We first retrospect some backgrounds of business process, SOC, SOA and web service technology in Section 1.1. In Section 1.2, we give an outline of the research issues on web services discovery and composition in service-oriented environments. Section 1.3 summarizes our solutions and Section 1.4 presents the thesis structure.

## *1.1 Business Process and Web Service*

*Business Processes* play an important role in modern information systems. A business process is a collection of related, structured activities that produce a service or product that meets the needs of a client. Roughly speaking, the concept of process can be regarded as tasks, production and outputs. Business processes are critical to any organization as they generate revenue and often represent a significant proportion of costs. Business process management (BPM) attempts to continuously improve processes. It could be described as a “process optimization process” focusing on aligning organizations with the wants and needs of clients, which aims to promote business effectiveness and efficiency while striving for innovation, flexibility and integration with technology [2]. BPM is now widely accepted as a design paradigm for organizational and IT-driven redesign initiatives. Figure 1.1 gives an example of business process management (BPM) service pattern [21]. As can be seen, through the orchestration of activities between people and system, BPM tools can manage business processes effectively.

Increasingly, business processes are seen from the viewpoint of how well they contribute to the efficient and coordinated delivery of services. Identification of inefficient processes may lead to the planned improvement of those processes, or, in some cases, to the decision to scrap the processes and outsource them to another organization [13]. With the development of global economics, now business organizations operate their business process in a globalized way. Business-to-Business (B2B) integration has become more important than ever before in today’s IT markets. As the Internet proliferates and the e-commerce activities explode, the B2B integration is driven by the automation of business processes among business organizations through the Internet and the needs of cutting the

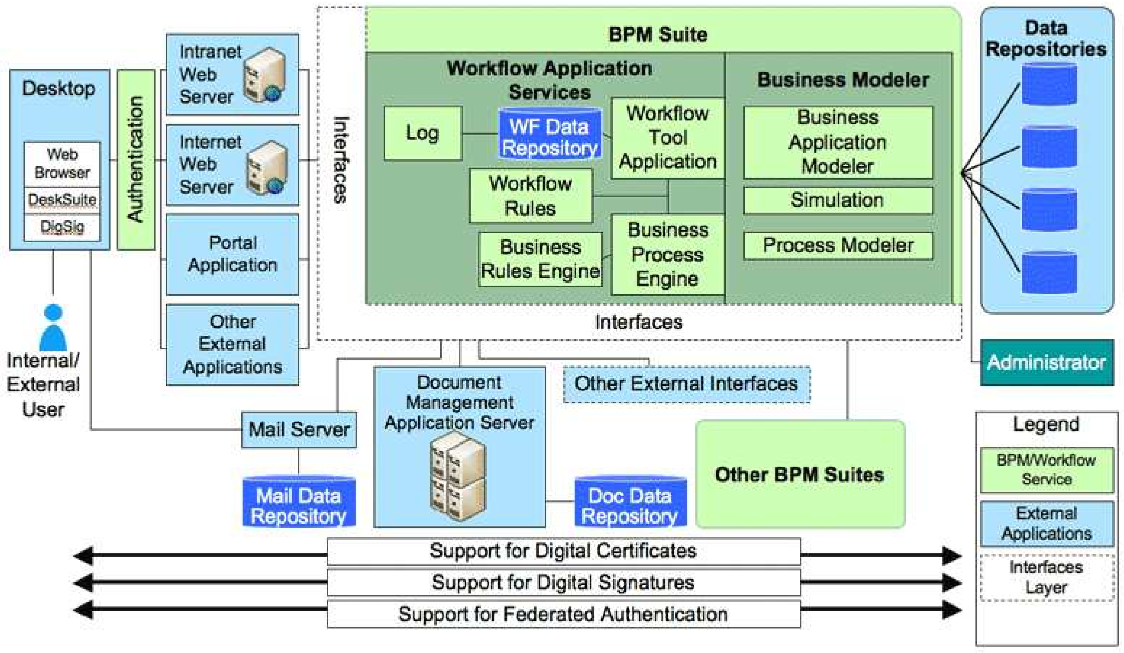


Figure 1.1: An example of BPM service pattern.

cost of developing, maintaining and integrating enterprises applications.

As an emerging cross discipline area for distributed computing, *Service-Oriented Computing* (SOC) paradigm promises to bridge the gap between business services and IT services, enable technology to help people perform business processes more efficiently and effectively, and enable businesses and organizations to collaborate globally through standard services. The scope of services oriented computing covers the whole lifecycle of web services research that includes business modulization, services modeling, services creation, services realization, services annotation, services deployment, services discovery, services composition, services delivery, service-to-service collaboration, services monitoring, services optimization, as well as services management. Also, a lot of technical aspects are currently covered in the service oriented computing paradigm, among which two important technology are *Service-Oriented Architecture* (SOA) and *Web service* [6].

The Service-Oriented Architecture (SOA) provides the foundations for organizations to identify and choose external services they would like to use, and also offer their own services to other organizations or businesses. In other words, SOA supports web services - the provision of business processes to interested parties via the Web. Web services are the fundamental building blocks of SOA. Web services are loosely coupled, autonomous, platform-independent, and reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols [4]. They can be described, published, discovered, orchestrated and programmed using standard protocols for collaborating applications distributed within and across organizational boundaries. Web services provide the basis for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols - they therefore enable the creation of dynamic virtual enterprises. Due to standard interfaces, web services have evolved as a dominant technology for integrating business processes and applications across organization boundaries on the Web [13]. Figure 1.2 and Figure 1.3 [92] give an intuitive impression on how enterprises use web services as an application distribution and integration tool, and how the B2B business is viable through web services.

This conceptually new approach brings multiple issues to researchers. In the next two sections, we outline the research issues in this thesis and summarize our contributions.

## *1.2 Problem Statement*

A Web Service is a set of related functionalities that can be loosely coupled with other services programmatically through the Web. With the rapid development and popularity

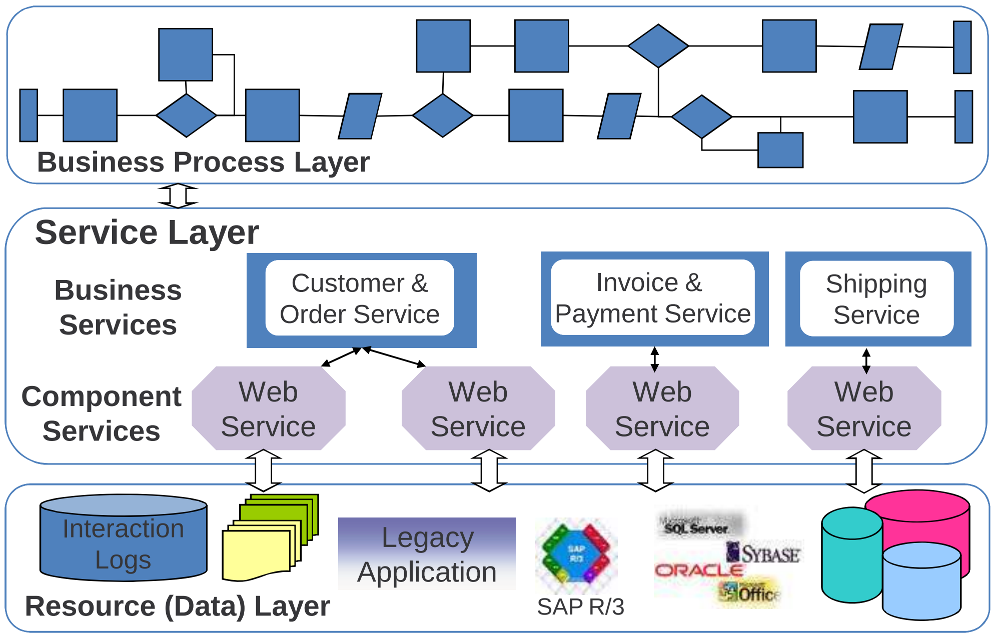


Figure 1.2: Web service architecture in enterprises.

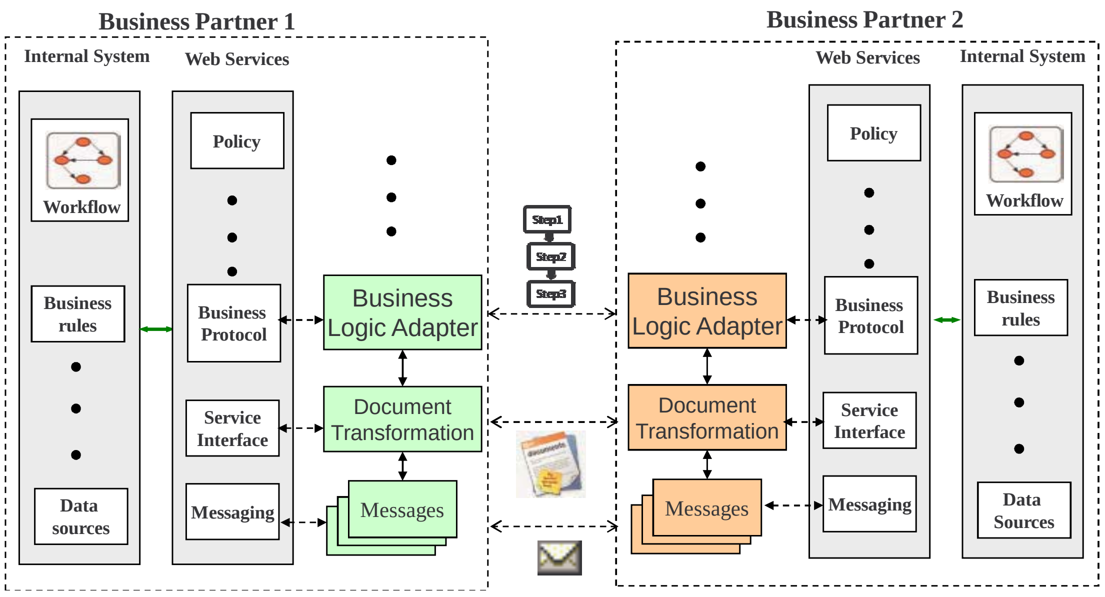


Figure 1.3: B2B integration via web services.

of Internet and e-commerce, business organizations are investigating ways to expose their current software components into web services so as to make use of distributed service computing resources. They are also investigating ways on how to incorporate services running on different platforms and hosted by service providers outside of their boundaries into more complex, orchestrated services. As the number of web services available on the Web and SOC applications increases, there is a growing need for mechanisms for discovering and composing services efficiently. Actually, how to discover appropriate web services and compose them has become an ever more important problem for organizations to take advantage of the tremendous opportunities offered by web services, to engage in business collaborations and service compositions, to identify potential service partners, and to understand service competitors and increase the competitive edge of their service offerings.

Apart from functional specification required by users during service discovery, some non-functional requirements such as *Quality of Service* (QoS) and transactional properties are also major concerns, because users need to select and compose web services not only according to their functional requirements but also to their transactional properties and QoS characteristics defined using a quality model. When users search and compose web services to deploy business processes, there are three major issues that need to be tackled, which are listed below.

- Efficient web services discovery
- Dynamic web service selection with QoS constraints
- Reliable execution of composite services

### *1.2.1 Efficient web services discovery*

The problem of web services discovery has become a hot research topic in service-oriented computing area. Currently, the *Universal Description, Discovery, and Integration* (UDDI) [114] registries are the dominating scheme for web services discovery. The UDDI specification provides a strategy to build an open framework for describing services, discovering businesses, and integrating business services across the Internet. UDDI serves as a repository for businesses to publish their web services specifications, which are defined clearly and described in a standard service description language - the *Web Service Description Language* (WSDL) [36]. However, the UDDI-based web services discovery faces three hurdles:

Firstly, in order to implement the inter-operations between distributed business services, users usually search the UDDI repositories for desired services publicly available on the Internet. The current UDDI repositories only enable users to retrieve services based on keywords [84][70]. However, service registries in UDDI are classified manually into different categories according to their commercial objectives described by WSDL [36] rather than the functions provided by them [44]. This category-based approach is quite informal and relies extensively on the shared common understanding of publishers and consumers. It does not provide any support for selecting competing alternative services that could potentially be reused. As a result, simple keyword-based service discovery method usually returns many useless retrieval results, which makes the task of finding relevant or desired services within the search results time-consuming and difficult, and thus can not satisfy common users. The task becomes even more challenging as the number of web services increases and the UDDI repository is filled with more and more published services. It would be helpful and significant if there is a ranking mechanism that can consider users' specific requirements and output desired web services within the top results. Furthermore, UDDI can not enable users to efficiently discover web services based on certain requirements, which can be either functional (the purpose of the service), and non-functional (constraints on various properties such as Quality of Service, transactional characteristics, etc) [27]. Therefore, the current UDDI repositories cannot meet the needs of the business processes in efficiency and requirement. The thesis aims to facilitate the discovery of web services based on user requirements in service-oriented environments by proposing novel approaches and tools.

Secondly, since many web services are created from existing applications and may not contain enough semantic annotations or sufficient text documentation, simple keyword-based queries may lead to low precision and recall ratio. Also, a good searching strategy should not check the full semantic compliance between service specifications and user requests, but return ranked lists of potentially useful services that are closest to the requests. In this thesis, we achieve this goal by integrating both syntactic and semantic matching into the web services discovery process.

Finally, in web service composition, based on the request information, a web services discovery broker often generates composite services by combine a group of structural individual services according to some business patterns. Then the discovered composite service can be invoked to concrete the underlying business process. However, the current UDDI model only enables the discovery of simple services. It does not provide a mechanism assisting users in selecting relevant services and composing with them. In other words, the implementation of automatically discovering or dynamically composing services is missing in the UDDI model.

### *1.2.2 Dynamic web service selection with QoS constraints*

In service-oriented environments, multiple web services may provide similar functionality, but may offer different non-functional properties (e.g., different prices, different response time, etc). Web services with similar functionality are typically grouped together in a single community. The members of a community are differentiated according to their non-functional properties. In the selection of a web service, the quality of service (e.g., execution duration and execution price, etc.) is a key factor for users, so it is important to consider both functional and non-functional properties in order to satisfy the constraints or requirements of users. To support dynamic composition of services, web services that satisfy users' functional requirements need to be located from a large number of service providers based on their quality of service. Therefore, when selecting existing web services from web service communities to generate composite services, the number of composed services may be very large and it is a challenge on how to select appropriate web services that are able to satisfy users' global QoS requirements based on a set of given user QoS preferences.

Furthermore, in addition to satisfying user QoS requirements by obtaining optimized service selection, users' personalized preferences need to be considered. Depending on their context, different users may have different preferences about the services they need. For instance, a user may say an expensive service *a* is better than a cheap service *b*, because although *a* is more expensive than *b* and exceeds the user's financial requirement a bit, it has much better reliability, which is crucial to her because of her strict reliability requirement. Similarly, another user running out of money prefers a cheaper service, while a cautious user may prefer services with "excellent" reputation, etc. So, whether a service is good or not in QoS depends on the user's context and preference. It is essential to model the users' personalized preferences and requests.

### *1.2.3 Ensure reliable execution of composite services*

The composition of web services provided by different organizations provides an efficient way to build complex application logics. However, as web services operate in a highly dynamic distributed environment and interact with each other, the possibility of unexpected behavior is high. For example, the invocation of a service may fail because of temporary unavailability of the service. The unexpected behavior from an individual service may bring negative impact on all the component services in the composition, even lead to failure of the running of the composite service. We need a mechanism that not only allows selecting a set of component web services satisfying a user's QoS criteria, but also provides transactional support to business integration via composing individual web services to ensure the overall consistent and reliable execution of business processes. A main problem that remains is, given a user's transactional requirements, how to select individual services and verify their transactional properties to ensure a correct composition and a reliable execution of a composite service.

Although there exist several transaction web service standards, they only provide limited support without giving much thought to the transactional features [80]. We need to investigate further the transactional properties of component services, as well as the workflow patterns which are often used as basis to compose services and which also determine the transactional behavior of composite services. In addition, we need an efficient verification mechanism to determine whether a given composite service has valid transactional characteristics or not according to some acceptable properties specified by the user.

## *1.3 Solution Overview*

In this thesis, we present a web service composition framework for efficient web services discovery and composition in service oriented environments. Various experiments have been carried out to demonstrate the efficiency and effectiveness of the methods proposed to support the framework. The contributions of the thesis are composed of three aspects, which are listed below:

- Efficient IR-style search over web services.
- A relaxation-based QoS model and computation framework for service selection.
- Verification of transactional requirements in web service composition.

### *1.3.1 Efficient IR-Style Search over Web Services*

We propose a novel IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. The features of our approach are summarized as follows:

- We propose algorithms for supporting web service operations matching. The key part of our algorithms is a schema tree matching algorithm, which employs a new cost model to compute tree edit distances. Our new schema tree matching algorithm can not only catch structures, but also the semantic information of schemas.
- Based on service operations matching, we use the agglomeration algorithm to cluster similar web service operations. Also, an approach to identify associations between web-service operations is presented. This approach uses the concept of *attribute*

*closure* to obtain sets of operations. Each set is composed of associated web service operations.

- We introduce the notion of *preference degree* for web services and then we define *service relevance* and *service importance* respectively as two desired properties for measuring the preference degree.

- We design novel algorithms for computing the relevance and importance degree of services. Our algorithms take into account both textual and structural information of web services.

- We define *service connectivity*, a novel metric to evaluate the importance of services.

- We do various experiments to search for desired web services. Initial results show the proposed IR-style search strategy is efficient and practical.

### *1.3.2 A Relaxation-Based QoS Model and Computation Framework for Service Selection*

In this part, we solve the problem of how to aggregate and leverage individual services' QoS information to derive the optimal QoS of composite services, given user QoS constraints. In particular, the main contributions of this work are listed as follows:

- We propose a novel personalized QoS model for performing flexible and adaptable service selection. The key idea of the model is to use skyline technique to relax users' QoS constraints and try to find the most possible services which meet users' requirements. The problem of QoS computation and policing is converted into the skyline deduction problem.

- Based on the proposed QoS framework, we develop various algorithms for making service selection on individual and composite services, respectively. We also introduce a top-*k* ranking strategy to reflect a user's personalized requirements.

- We present the experimental results of a thorough evaluation. Experimental evaluation shows the proposed QoS model is efficient and practical.

### *1.3.3 Verification of Transactional Requirements in Web Service Composition*

We propose a novel verification method to support transactional composition of web services. Service selection passing the verification can ensure a correct composition and a reliable execution of a composite service according to user transactional requirements. In particular, the contributions of this work can be summarized as below:

- We define a transactional model for composite web services. The model is based on the transactional properties of component web services contributing to the composite service, and the dependencies between component services, which are defined by the workflow patterns that specify how services are combined together and how the behavior of a service interacts with other services.

- We model web services as automata. Based on automata theory, we model both component web services and composite services as transition systems, and use *Accepted Termination States* to describe the transactional properties of composite services. A verification algorithm is developed to carry out automated verification of Accepted Termination States on composite services.

- In addition, for the cause of efficiency we use Temporal Logic to describe the transactional properties of a composite service, and then we employ the *SPIN* model checker to carry out automated verification of temporal logic properties on the composite service.

## *1.4 Thesis Structure*

The reminder of this thesis is organized as follows. In Chapter 2, we introduce some basic concepts and definitions used in the thesis and survey some technologies that are related to web services discovery and composition in service-oriented environments. Chapter 3 presents a novel IR-style web services discovery scheme. In Chapter 4, we give a personalized QoS-driven service selection technique in dynamic service environments. In chapter 5, we focus on the verification of service selection for satisfying user transactional requirements of composite services. Finally, in Chapter 6, we conclude the thesis and outline some future work. Figure 1.4 illustrates the structure of the thesis chapters.

Chapter 2 starts from introducing the current web service specification and the state-of-the-art in service computing. Afterwards, the methodology for modeling web service match is illustrated. The next section of the chapter discusses the skyline technique in spacial database and shows this idea can be applied into our personalized QoS-driven web services discovery model. Then we review some classic model checking methods and tools, including temporal logic, TLA logic, SPIN, CPN etc. Last, we give a brief summary to conclude the chapter.

Chapter 3 firstly reviews research on web services discovery. Secondly, it introduces the conception of preference degree for service ranking. Thirdly, it presents models and definitions for service relevance and service importance, and then various algorithms for

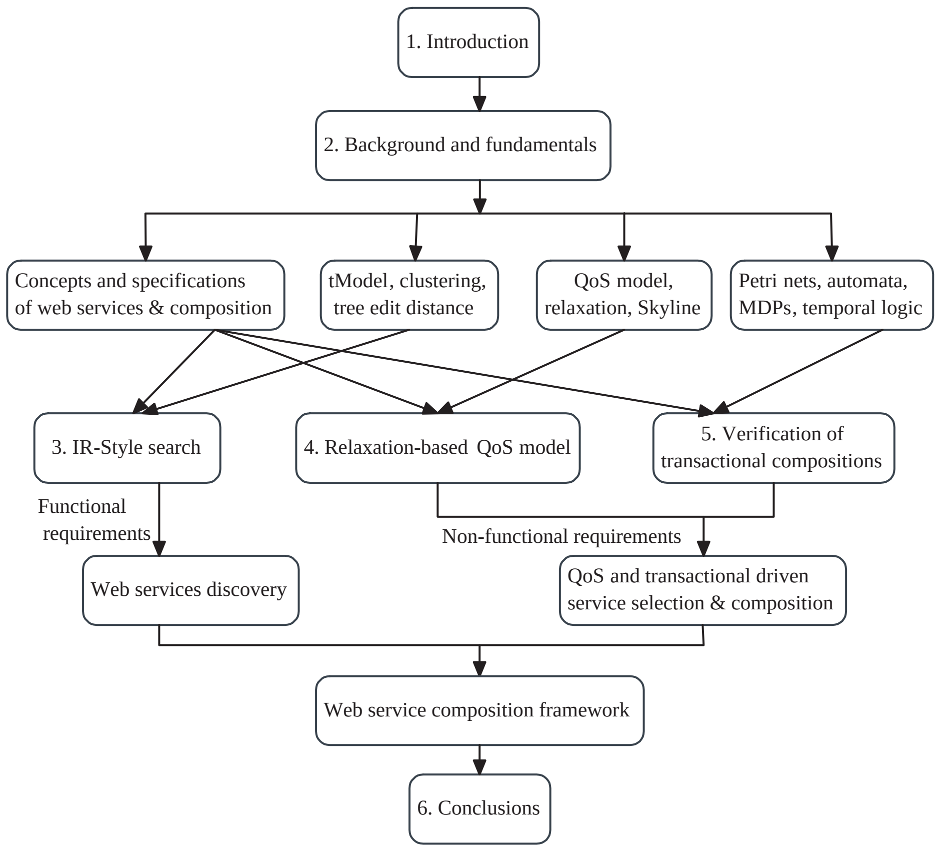

1. Introduction
2. Background and fundamentals
Concepts and specifications of web services & composition
tModel, clustering, tree edit distance
QoS model, relaxation, Skyline
Petri nets, automata, MDPs, temporal logic
3. IR-Style search
4. Relaxation-based QoS model
5. Verification of transactional compositions
Functional requirements
Non-functional requirements
Web services discovery
QoS and transactional driven service selection & composition
Web service composition framework
6. Conclusions

Figure 1.4: Structure of the thesis.

ranking web services are provided, followed by a thorough experimental evaluation. The last section gives concluding remarks.

Chapter 4 first gives a motivating example for personalized QoS-driven service discovery. Secondly, it introduces the computing model for individual services and composite services, and describes the corresponding QoS query algorithms on the two models, respectively. Then we present the personalized service selection strategy using a relaxation approach and its justification. Afterwards, we provide examples and experiments to demonstrate the validity of our methods. Next, it discusses related work on QoS-driven web services discovery and selection. Finally, the chapter is concluded with a short summary of its contribution.

Chapter 5 first introduces the main points in our verification strategy. Then we explain the transactional properties of web services and how they determine the transactional properties of the composite service they belong to. Next, we model both component web services and composite services as transition systems, and use *Accepted Termination States* to describe the transactional properties of composite services. A verification algorithm is developed to carry out automated verification of Accepted Termination States on composite services. Afterwards, we present the formal expression of user transactional requirements using temporal logic. We illustrate how our formal expression can be verified by the classic model checking tool *SPIN* so as to assist designers to compose valid composite services. Finally, we discuss some related work, followed by some concluding remarks.

Finally, in Chapter 6, we give the concluding remarks of the thesis and point out some future work.

# Chapter 2
# BACKGROUND AND FUNDAMENTALS

This chapter presents basic concepts and techniques for better understanding the algorithms and methods developed in this thesis. We first introduce various web service specifications and models, including WSDL, UDDI, SOAP, XML schema and BPEL. Then we introduce the concept, models and research issues of web service composition. After that, we spend more space on query and matching techniques for web services discovery such as TF/IDF, clustering, tree edit distance, as they are very important foundations of our research topic. Afterwards, we give some basics about Skyline and its computing algorithms, which are used in our relaxation technique for QoS-driven service selection. Finally, we review some formal representation and verification concepts, models and tools, including temporal logic, automata, SPIN, etc, which are the basis of our verification scheme for transactional composite web services. All the contents given here provides a foundation for further study of web services discovery and composition, and will be the frame of references throughout the remainder of the thesis.

## *2.1 Specification and Modeling of Web Services*

Web service [48, 136] have become the preferred technology for realizing the SOC (Service-oriented Computing) paradigms. A web service is a set of related application functions that can be programmatically published, located, and invoked across the Web. Web services perform encapsulated business functions that can range from simple request-reply to full

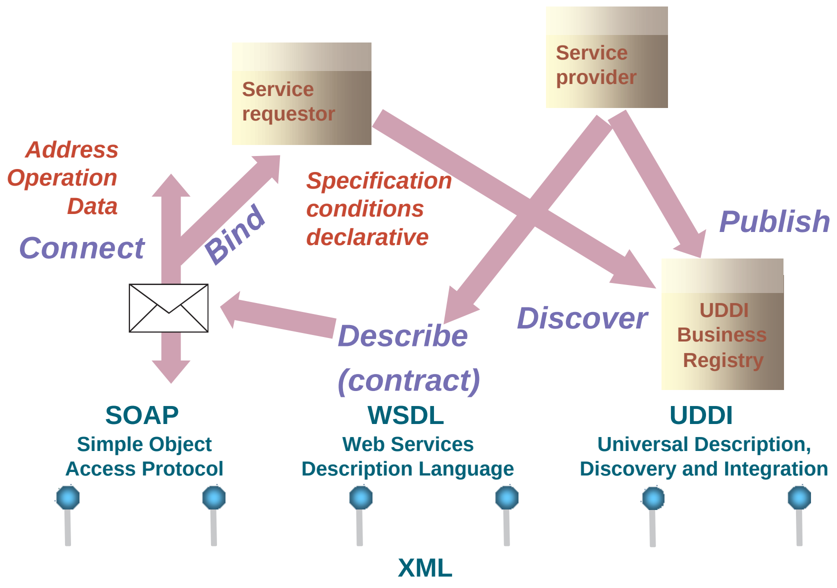


Figure 2.1: Web service roles and interactions through standardized protocols.

business process interactions. Typical web services, for example, are currency conversion service, stock quotes/stock charting service, credit card verification/payment processing service, integrated travel planning service, etc.

The main conceptual characteristics and properties of web services are: *self-contained*, *self-describing* and *modular* [48]. Firstly, web services are self-contained software components, that is, no additional software is required on the client side. Client users can easily start with programming languages support such as XML and HTTP. An existing application can be invoked without writing a single line of code. Secondly, those software components are self-describing. The client and server only need to recognize the format and content of request and response messages. The definition of the message format is included in the message. Finally, web services are modular, reusable software components, that is, they allow developers to reuse the building blocks of code created by others. For example, simple web services can be aggregated to form more complex web services by using workflow techniques through the BPEL language.

A group of standards have been established to manage web service lifecycles and service-client communication. These standardized protocols play key roles in web services, including: Extensible markup Language (XML), Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL), and Simple Object Access Protocol (SOAP). Figure 2.1 [136] presents the web service roles and interactions through these protocols. The protocols mentioned here are described in the following sections.

### *2.1.1 XML, XML schema and DOM*

*EXtensible Markup Language*(XML) [131][123] is a subset of the *Standard Generalized Markup Language* (SGML), the standard for creating markup documents. XML is classified as an extensible language, because XML tags are not predefined and users can define their own tags. XML is self-descriptive and it is designed to carry data, not to display data. Via XML we can define the content of a document separately from its formatting, which makes it easy to reuse that content in other applications or other presentation environments. Most importantly, XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion [116][54]. Therefore, XML is ideal for use on the Internet. Figure 2.2 is a sample document taken from [88] showing how data can be represented in XML format.

The *XML Schema* [111][122] language is also referred to as *XML Schema Definition (XSD)* . An XML schema is a description of a type of XML document, typically expressed

```
<?xml version="1.0"?>
  <books>
    <book>
        <author>Carson</author>
        <price format="dollar">31.95</price>
        <pubdate>05/01/2001</pubdate>
    </book>
    <pubinfo>
        <publisher>MSPress</publisher>
        <state>WA</state>
    </pubinfo>
  </books>
```

Figure 2.2: An example of XML file

```
<xs:element name="book" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="price" type="xs:decimal"/>
      <xs:element name="pubdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 2.3: An example of XML schema

in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself. An XML schema provides a view of the document type at a relatively high level of abstraction [110]. Figure 2.3 is a segment of the XML schema file describing the XML file in Figure 2.2.

The XML DOM [86][124] defines the objects and properties of all XML elements, and the methods (interface) to access them. It provides a standard way to get, change, add, or delete XML elements. In XML DOM, an XML document is presented as a tree structure, with elements, attributes, and text as nodes. Figure 2.4 shows how memory is structured

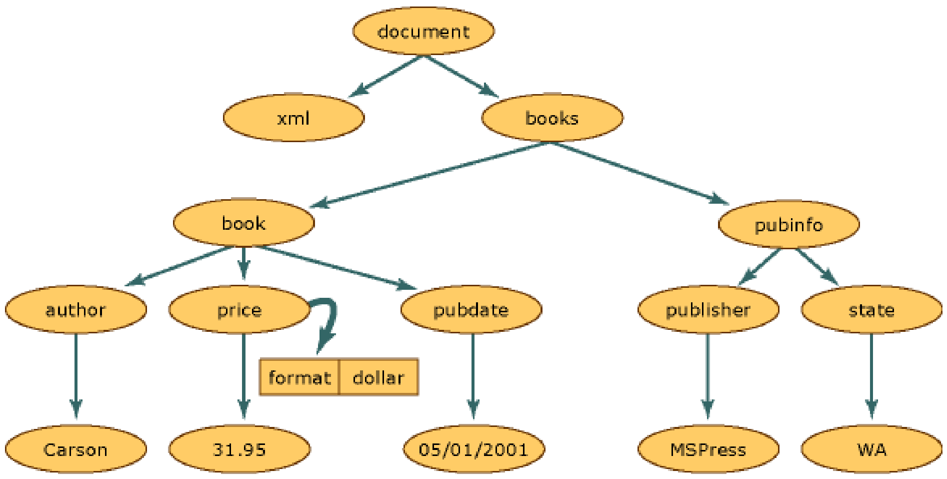


Figure 2.4: XML DOM tree structure

when the sample XML data [88] is read into the DOM tree structure.

Each circle represents a node in this XML DOM tree structure. Each node is called an XmlNode object, which is the basic object in the DOM tree.

### *2.1.2 WSDL*

The Web Services Description Language (WSDL) [132] is an XML-based language that provides a model for describing web services. A WSDL document describes how to invoke a service and provides information on the data being exchanged, the sequence of messages for an operation, protocol bindings and location of a service.

More specifically, a WSDL document defines *services* as collections of network endpoints, or *ports*. In WSDL, the abstract definition of endpoints and messages is separated from concrete implementation. *Messages* and *port types* provide abstract definitions of the data being exchanged and the operations performed by a service. The concrete protocol and data format specifications are associated with a particular port type, forming a reusable binding. By associating a network address with a reusable *binding*, a port can be defined. A collection of ports make up a service. To sum up, the elements defined in a WSDL document are listed as follows [48][132]:

- **Types** - the structure of the data contained in messages, often defined by XML schema specification.
- **Message** - an abstract definition of the data being transferred.
- **Operation** - an abstract description of an action a service can perform. Operation defines which message is the input and which message is the output.
- **Port Type** - a collection of all operations exposed by the web.
- **Binding** - a specific protocol, data format or structure for an abstract message, operation or endpoint.
- **Port** - a single endpoint defined by specifying a single network address for a binding endpoint.
- **Service** - a group of related ports or endpoints.

WSDL documents provide an efficient way for web developers to expose their applications as services accessible on the Internet. Once published through UDDI, WSDL documents can be found by other applications and bound with them to execute other complicated business processes.

As an example, we use the classic *Loan Approver* [76] web service to show the components of a WSDL file. The *Loan Approver* service contains one single operation, called

```
<definitions targetNamespace="http://tempuri.org/services/loanapprover"
                  xmlns:tns="http://tempuri.org/services/loanapprover"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:loandef="http://tempuri.org/services/loandefinitions"
                  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <import namespace="http://tempuri.org/services/loandefinitions"
        location="http://localhost:8080/bpws-samples/loanapproval/loandefinitions.wsdl"/>

  <message name="approvalMessage">
    <part name="accept" type="xsd:string"/>
  </message>

  <portType name="loanApprovalPT">
    <operation name="approve">
      <input message="loandef:creditInformationMessage"/>
      <output message="tns:approvalMessage"/>
      <fault name="loanProcessFault"
            message="loandef:loanRequestErrorMessage"/>
    </operation>
  </portType>

 <binding ...> ... </binding>
 <service name="LoanApprover">....</service>
 </definitions>
```

Figure 2.5: *Loan Approver* WSDL

*approve*, to process a loan request. *approve* takes as input the information about the customer, and outputs an approval message. The WSDL file defining the *Loan Approver* web service is shown in Figure 2.5.

### *2.1.3 UDDI*

Universal Description, Discovery, and Integration (UDDI) is an XML-based protocol which provides a universally recognized directory model for publishing and discovering web services. UDDI provides a foundation for developers and administrators to readily share information about internal web services across the enterprises and public web services across the Internet [48][88][114]. UDDI manages the discovery of web services by relying on a distributed registry of businesses and their service descriptions implemented in a common XML format. Business registration is the main part of UDDI. A UDDI business registration consists of three components: white pages, yellow pages and green page. White pages include business name, contact information, and a text description of the business's services. Yellow pages include industrial categorizations based on standard taxonomies. Green pages are composed of technical information and references to specifications for services exposed by the business.

The data structure within UDDI is comprised of four constructions: a *businessEntity* structure, a *businessService* structure, a *bindingTemplate* structure and a *tModel* structure. The relationship between these four core structures [94] is shown in Figure 2.6. The businessEntity represents a web Service provider; the businessService is a group of services that may reside in a single businessEntity; the bindingTemplate gives technical information needed to bind and interact with the target web service; whereas tModels is used to represent interfaces. Classification information and namespace can be added into the interface to make UDDI search easier by tModels, which is playing an important role in UDDI.

### *2.1.4 SOAP*

*Simple Object Access Protocol* (SOAP) is a simple XML-based protocol enabling applications to exchange of information in a decentralized, distributed environment over HTTP. To put it simply, SOAP is a protocol for accessing a web service. A SOAP message is a transmission of information from a sender to a receiver. In SOAP protocol, each message is composed of three parts: (1) an envelope that defines a framework for describing what is in a message and how to process it; (2) a set of encoding rules for expressing instances of application-defined datatypes, and (3)a convention for representing remote procedure calls

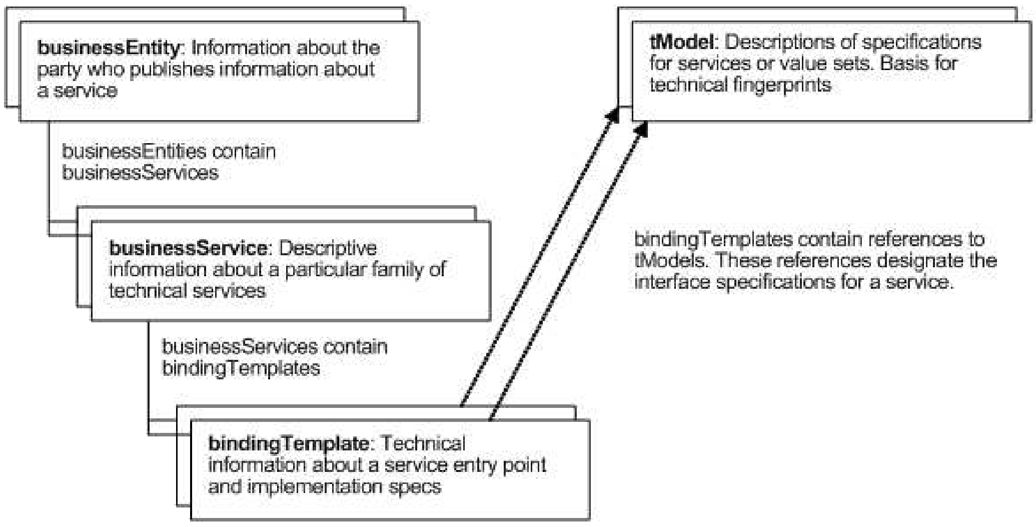

businessEntity: Information about the party who publishes information about a service
businessEntities contain businessServices
businessService: Descriptive information about a particular family of technical services
businessServices contain bindingTemplates
bindingTemplate: Technical information about a service entry point and implementation specs
tModel: Descriptions of specifications for services or value sets. Basis for technical fingerprints
bindingTemplates contain references to tModels. These references designate the interface specifications for a service.

Figure 2.6: Four core constructions/types of UDDI

and responses [48][23].

SOAP aims "*to help developers build web services and link heterogeneous components over the Internet. It provides an open, extensible way for applications to communicate using XML-based messages over the Web, regardless of what operating system, object model or language particular applications may use. SOAP facilitates universal communication by defining a simple, extensible message format in standard XML and thereby providing a way to send that XML message over HTTP*" [104].

## *2.2 Web Service Composition*

### *2.2.1 Introduction*

*Business Processes* play an important role in modern information systems. A business process is the scheduling of a set of activities that collaborated with each other for achieving certain goals. The activities involved in a business process can be carried out within a single organization or across several. Workflow technology aims for “automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for actions according to a set of procedural rules” [45].

With the rapid development of distributed computing and the Internet, *web service composition* technology emerges as a new approach for efficient automation of business processes based on Service-Oriented Architecture (SOA). SOA provides a standardized platform for organizations to expose and access the functionality of their applications as services, to communicate and manage services, and to combine exposed functionalities of simple applications, possibly offered by different companies, into complex, orchestrated business processes. Composite web services are often created from simple, basic and loose coupling services. To get an intuitive understanding of web service composition, let us present an example in the following.

Consider a *travel reservation scenario* [112] [10] illustrated in Figure 2.7, where a travel agent offers customers to book complete travel packages, including air tickets, hotels, car rental and excursions. The travel agency cooperates with external specialized service providers (airlines, hotel chains, etc) that offer web services to query their offerings and perform reservations. Also, Credit card companies provide services for the travel agency to guarantee payments made by consumers. Because of the loosely coupled-nature of web services, the credit card companies and the service providers can offer their services broadly to satisfy their customers. Therefore, the travel agent is able to have access to more services, offering more options to its customers. In this situation, once receiving the customer request the travel agent will choose the required hotel and flight booking for it. Then the agent will request specialized services, which work independently of each other and try to book the desired reservations. Once both services are successfully completed, the agent asks for

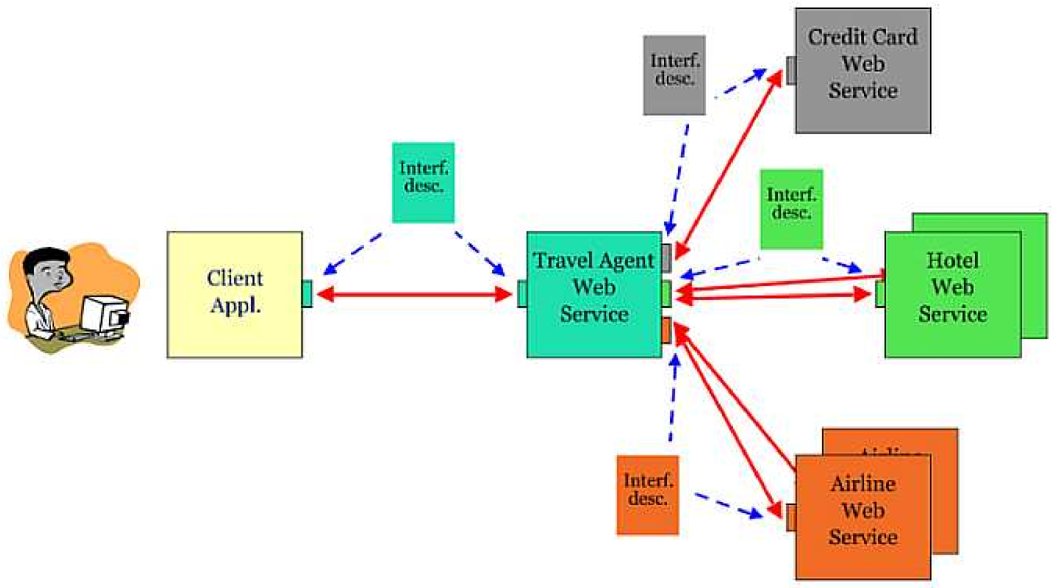


Figure 2.7: A travel reservation scenario

confirmation from the customer. After getting the confirmation, the travel agent contacts the bank service that the user chose to finish payment. We assume that all the services are using common concepts (e.g. flight, economy class, room, etc), which can be described by a travel industry ontology.

### 2.2.2 *Issues of Web Service Composition*

Composite services are the automation of business processes. Since composite services operate in an open, different and dynamic environment, we need better tools and methods to assist designers/developers to form process from services, to enable flexible and effective enactment of processes and to improve the running performance of the processes composed with services. There are many issues with the problem of web service composition.

Specifically, a web service composition method should consider a group of principles and issues [85][51][115][143], including *Service Description*, *Service Discovery*, *Conversation Modeling*, *Execution monitoring*, *Exception Handling and Recovery*, and *Transactions Management.*

- **Service Description**. Web service composition requires each service provider involving in the composition to describe the properties of their explicitly in a descriptive file, for example, using WSDL. The properties include functional properties, where the functionalities, input and output of a service are stated; and non-functional properties, including QoS (quality of service), transaction, conversation, dependability, security, etc. Any tools and languages need to support description of both functional and non-functional properties of a web service.

- **Service Discovery**. A web services discovery mechanism is needed to support finding candidate web services using matching algorithms that compares "service requests" with "service advertisements". Matching strategies can be based on keywords, the types of inputs/outputs, or semantic information, and so on in WSDL files.

- **Conversation Modeling**. A conversation [40] is the global sequence of messages exchanged among the components of a composite service. Conversation modeling provides a formal method for specifying and analyzing the interactions among the services participating in a composite web service. It facilitates service discovery and dynamic binding, service composition validation, service composition framework generation and analysis of conversations and conversation among services.

- **Execution monitoring**. In practice, there are two different approaches to monitor the execution of composite services: Centralized and distributed. Centralized mode is similar to the traditional client-service scheme. A central controller schedules the execution of component services. Whereas in distributed mode, participating web services share their execution context. The component services coordinate with each other to execute the data and control flow, so as to guarantee a correct ordered execution of the services.

- **Transactions Management**. In order to guarantee the interactions among services, it is important to add a transaction management policy into the composite service execution environments. Typical transaction management protocols are WS-Transaction standard, proposed by IBM and WS-TXM suggested by Sun. WS-Transaction builds upon WS-Coordination framework, based on which the protocols for centralised and peer-to-peer transactions are defined. WS-TXM is designed to ensure that a composite application always either reaches successful completion or transits to predictable, known state if one or more of the individual web services fail in the composite application.

- **Exception Handling and Recovery**. There should be an exceptional module that monitors the exceptions web services throw. If a failure occurs at some point in the execution of a process instance, some recovery actions need to be done, such as compensation, backward recovery, forward recovery, etc.

### *2.2.3 Web Service Composition Standards*

#### *WSCL*

The *Web Service Conversation Language* (WSCL) [18] was submitted to W3C by HP and was published as Technical Notes. WSCL defines the overall input and output message sequences for one web service using a finite state automaton FSA over the alphabet of message types. It specifies the XML documents being exchanged, and the allowed sequencing of these document exchanges. WSCL conversation definitions are themselves XML documents, and can therefore be interpreted by web services infrastructures and development tools. WSCL may be used in conjunction with other service description languages like WSDL, e.g., to provide protocol binding information for abstract interfaces and to specify the abstract interfaces a concrete service is supporting.

#### *WSCI*

The *Web Service Choreography Interface* (WSCI) 1.0 [15] specification was submitted to W3C by BEA Systems, BPMI.org, Commerce One, Fujitsu Limited, Intalio, IONA, Oracle Corporation, SAP AG, SeeBeyond Technology Corporation, and Sun Microsystems. It is an XML-based interface description language that describes the flow of messages exchanged by a web service participating in choreographed interactions with other services.

WSCI describes the dynamic interface of the web service participating in a given message exchange by means of reusing the operations defined for a static interface. WSCI works in conjunction with the Web Service Description Language (WSDL), the basis for the W3C Web Services Description Working Group; it can, also, work with another service definition language that exhibits the same characteristics as WSDL.

WSCI describes the observable behavior of a web service. This is expressed in terms of temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. WSCI also describes the collective message exchange among interacting web services, thus providing a global, message-oriented view of the interactions.

*BPEL4WS*

Developed by BEA, IBM, Microsoft, SAP, and Siebel, BPEL4WS (*Business Process Execution Language for Web Services*) [1] is an XML-based language to specify business processes and business interaction protocols. BPEL4WS is process-oriented. It defines a business process as coordinated sets of web service interactions, in which a given task is achieved by a subset of services. BPEL4WS merges the concepts of XLANG [22](Web Services for Business Process Design) and WSFL [19] (Web Services Flow Language) as a standard for web services flow specification, enabling the creation of compositions of web services.

BPEL4WS supports the modeling of executable and abstract processes. An executable business process models the actual behavior of a participant in a business interaction. An abstract process is a partially specified process that is not intended to be executed.

Typically, a BPEL business process invokes the involved web services to fulfil user requirement. A BPEL process consists of steps; each step is called an “activity”. BPEL supports both primitive and structured activities. Primitive activities are used for common tasks, including

- `<invoke>` - Invoking other web services.

- `<receive>` - Waiting for the client to invoke the business process by sending a message.
- `<reply>` - Generating a response for synchronous operations.
- `<assign>` - Manipulating data variables.
- `<throw>` - Indicating faults and exceptions.
- `<wait>` - Waiting for some time.
- `<terminate>` - Terminating the entire process.

These primitive activities can be combined to form complex activities. BPEL supports several structure activities to combine primitive activities. Typical structure activities are

- `<sequence>` - for definition of a set of activities that will be invoked in an ordered sequence.
- `<flow>` - for defining a set of activities that will be invoked in parallel.
- `<switch>` - for implementing branches.
- `<while>` - for defining loops.
- `<pick>` - for selecting one of several alternative paths

Each BPEL process will also define partner links, using `<partnerLink>`, and declare variables, using `<variable>`.

```
<process name="loanApprovalProcess"
          targetNamespace="http://acme.com/simpleloanprocessing"
          xmlns="http://schemas.xmlsoap.org/w s/2002/07/business-process/"
          xmlns:lns="http://loans.org/w sdl/loan-approval"
          xmlns:loandef="http://tempuri.org/services/loandefinitions"
          xmlns:apns="http://tempuri.org/services/loanapprover">
  <partners>
    <partner name="customer"
              serviceLinkType="lns:loanApproveLinkType"
              myRole="approver"/>
    <partner name="approver"
              serviceLinkType="lns:loanApprovalLinkType"
              partnerRole="approver"/>
  </partners>
  <containers>
    <container name="request" messageType="loandef:CreditInformationMessage"/>
    <container name="approvalInfo"  messageType="apns:approvalMessage"/>
  </containers>
    <invoke name="invokeapprover"
             partner="approver"
             portType="apns:loanApprovalPT"
             operation="approve"
             inputContainer="request"
             outputContainer="approvalInfo">
    </invoke>
    <reply name="reply" partner="customer" portType="apns:loanApprovalPT"
            operation="approve" container="approvalInfo">
    </reply>
  </sequence>
</process>
```

Figure 2.8: The *loan approval* process

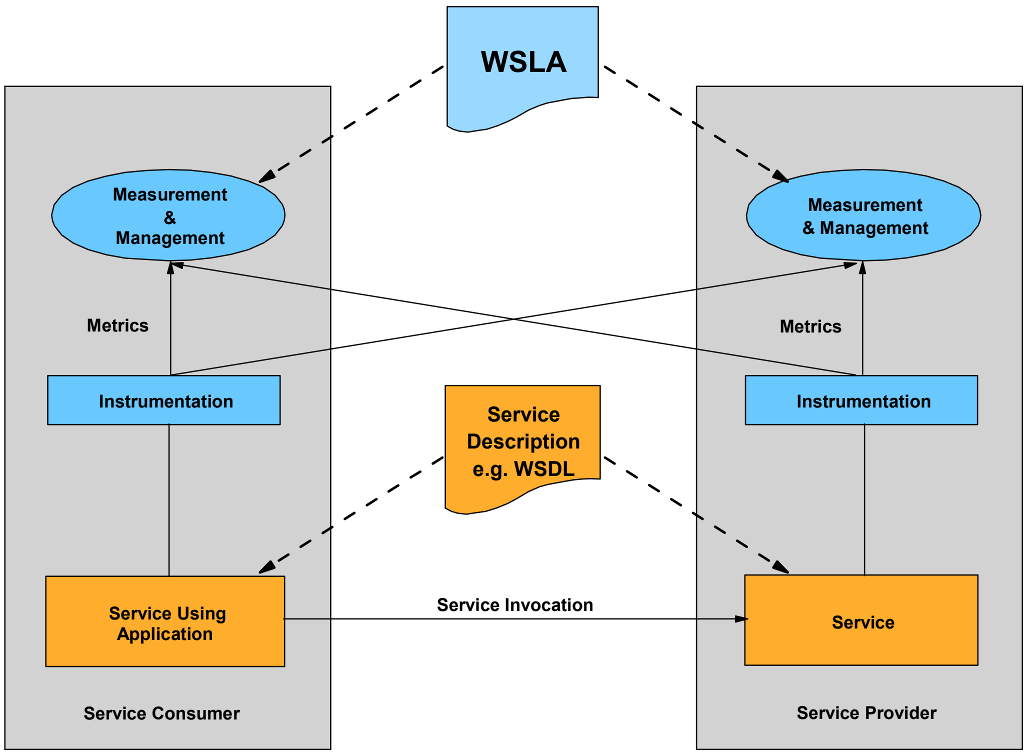


Figure 2.9: Role of a Web Service Level Agreement.

In order to demonstrate how a business process can be created by BPEL4WS, we describe a simple example taken from [76], which shows a loan approval process. In this example, a customer sends a request for a loan, then the financial institution's web service is invoked, and finally replies to the customer. The running flow of the loan approval process is shown in Figure 2.8.

*WSLA*

The *Web Service Level Agreement language* (WSLA) [16], presented by IBM, is a framework for specifying and monitoring Service Level Agreements (SLA) for web services. A WSLA document is an XML document which defines assertions of a service provider to perform a service according to agreed guarantees for IT-level and business process-level service quality parameters such as response time and throughput, and measures to be taken in case of deviation and failure to meet the asserted service guarantees, for example, a notification of the service customer. The assertions of the service provider are based on a detailed definition of the service parameters including how basic metrics are to be measured in systems and how they are aggregated into composite metrics for web service composition. In addition, a WSLA expresses which party monitors the service, third parties that contribute to the measurement of metrics, supervision of guarantees or even the management of deviations of service guarantees. Interactions among the parties supervising the WSLA are also defined. WSLA can be used by both service provider and service customer to configure their respective systems to provide and supervise their service. See Figure 2.9 [16] for the role of a web service level agreement in web service environments.

As far as web service composition is concerned, its Quality of Service (QoS) issue can be solved by the WSLA language. With WSLA, we can specify the QoS property, requirement and agreement.

## *2.3 Query and Matching Techniques for Web Services Discovery*

In the previous sections, we introduced various web service specifications and models, including WSDL, UDDI, SOAP, XML schema and BPEL. We also introduced the models and research issues of web service composition. In this section, we will focus on important related techniques and backgrounds for web services discovery. In the first part of this section, we illustrate tModel, which serves as an important tool to facilitate discovery of web services. Then we present some classic clustering algorithms, which lay basis for our web services discovery, clustering and ranking approaches proposed in the thesis. After that, we illustrate the tree edit distance algorithm, which is mainly used to calculate the similarity degree of two web service messages represented as tree structures.

### *2.3.1 tModel*

We have known that UDDI can be seen as an online "yellow book", where service providers register their web services, and service consumers try to find the desired service descriptions from this online registry. tModel is an important tool to facilitate businesses to discover desired web services through UDDI [5][120]. A tModel is a single interface of the web service that we are going to develop and register, i.e. a technical specification and a data structure representing a service type (a generic representation of a registered service) in the UDDI registry. Each tModel consists of a name, a service description, and a Universal Unique Identifier (UUID). Via tModel, service providers can register their web services according to a predefined list of service types, and the tModel organizes the service type's information and makes it accessible in the registry database. Therefore, businesses can search the registry's service types list to find service providers in a much easier way.

### *2.3.2 Clustering Algorithms for Web Services Discovery*

The Web Service Description Language (WSDL) specification provides a standard way for describing web services in natural language. A WSDL file shows the relevant information about a web service, such as service name, service operations and service description. Our web services discovery methods are based on unsupervised clustering for web services. They mainly focus on the textual features of WSDL files and depend on analyzing the frequency of terms to make service matching. Furthermore, by schema matching, which describe the grammatical XML structure used by web services to send and receive messages, we try to compute the similarity between two web services, and find any related matches. The following sections briefly describe various clustering algorithms employed in web services discovery processes of users, given a textual description of desired services.

*Clustering Analysis*

Clustering analysis is a widely used data mining algorithm for many data management applications. Clustering is a process of partitioning a set of data objects into a number of object clusters, where each data object shares high similarity with the other objects within same cluster but is quite dissimilar to objects in other clusters [135]. Different from classification algorithm that assigns a set of data objects with various labels previously defined via a supervised learning process, clustering analysis is to partition data objects objectively based on measuring the mutual similarity between data objects, i.e. via a unsupervised learning process. Due to the fact that the class labels are often not known before data analysis, for example, in case of being hard to assign class labels in large databases, clustering analysis is sometimes an efficient approach for analyzing such kind of data. To perform clustering analysis, similarity measures are often utilized to assess the distance between a pair of data objects based on the feature vectors describing the objects, in turn, to help assign them into different object classes/clusters. There are a variety of distance functions used in different scenarios, which are really dependent on the application background. For example, cosine function and Euclidean distance function are two commonly used distance functions in information retrieval and pattern recognition [61]. On the other hand, assignment strategy is another important point involved in partitioning the data objects. Therefore, distance function and assignment algorithm are two core research focuses that attract a lot of efforts contributed by various research domain experts, such as from database, data mining, statistics, business intelligence and machine learning etc.

The main data type typically used in clustering analysis is the matrix expression of data. Suppose that a data object is represented by a sequence of attributes/features with corresponding weights, for example, in the context of web services discovery, a web service is modeled as a weighted keyword sequence. Like what we discussed above, this data structure is in the form of object-by-attribute structure, or $n \times m$ matrix where $n$ denotes the number of data objects and $m$ represents the number of attributes. In addition to data matrix, similarity matrix where the element value reflects the similarity between two objects is also used for clustering analysis. In this case, the similarity matrix is expressed by an $n \times n$ table. For example, an adjacency matrix addressed in the web services discovery is actually a similarity/relevance matrix. In this work, we adopt the first data expression, i.e. data matrix to address web service clustering and discovery.

*Cosine-Based Similarity Metric*

The cosine coefficient is to measure the cosine function of angle between two feature vectors. Cosine function is widely used in information retrieval research. The cosine coefficient can be calculated by the ratio of the dot product of two vectors with respect to their vector norms. Given two vectors $A$ and $B$, the cosine similarity is defined as:

$$sim(A,B) = \cos(\overrightarrow{A}, \overrightarrow{B}) = \frac{\overrightarrow{A} \cdot \overrightarrow{B}}{\left|\overrightarrow{A}\right| \times \left|\overrightarrow{B}\right|}$$

where “$\cdot$” denotes the dot operation and “$\times$” denotes the norm form.

*Tree-based Similarity Metric*

Each web-service operation can be modeled as a multi-input-multi-output function of the form $f : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$, where $s_i$ and $t_j$ are data types in according with XML schema specification. Intuitively, we consider two web-service operations similar if they have similar input/output data types. Since XML schemas are usually modeled as trees, the problem of web-service operation matching is converted to the problem of schema tree matching.

*Tree edit distance* [118][146] is one of the efficient approaches to describe the similarity between two trees. Tree edit distance between two trees $A$ and $B$ is the minimum cost with the set of tree edit operations needed to transform $A$ into $B$. Classic tree edit operations include node removal, node insertion, and node relabeling. A cost is assigned to each tree edit operation, so the tree edit distance can be derived by finding a minimum-cost mapping between two trees. Many algorithms have been proposed to address the problem of finding the minimum-cost mapping, for example [118][147][126], etc, but their computing complexity are usually high.

There are also some restricted tree matching algorithms, in which *simple tree matching (STM)* [137] was proposed first and it does not allow node relabeling operation and level crossing. STM evaluates the similarity of two trees by computing the maximum matching using dynamic programming. We briefly explain how STM [146] works in the following paragraph.

Let $A$, $B$ be two trees and $i$, $j$ are two nodes belonging to $A$ and $B$ respectively. A matching between two trees is defined as a mapping $M$, such that for each non-root node pair $(i, j) \in M$, we have $(parent(i), parent(j)) \in M$. A matching is maximum if it has the

```
input  : A,B
output: The similarity of A,B
if the roots of the two trees A and B contain distinct symbols then
    return (0);
else
    $m$ := the number of first-level sub-trees of $A$;
    $n$ := the number of first-level sub-trees of $B$;
    $M[i,0] := 0$, for $i = 0,.,.,m$;
    $M[0,j] := 0$, for $j = 0,.,.,n$;
    for $i \leftarrow 1$ to $m$ do
        for $j \leftarrow 1$ to $n$ do
            $M[i,j] := max(M[i,j-1], M[i-1,j], M[i-1,j-1] + W[i,j])$;
            where $W[i,j] =$ Simple-Tree-Matching$(A_i, B_j)$;
        end
    end
    return $(M[m,n] + 1)$;
end
```

**Algorithm 2.1**: The **Simple-Tree-Matching** algorithm

maximum number of pairs. STM compares the roots of $A$ and $B$ first. If the roots contain distinct labels, then $A$ and $B$ do not match; otherwise the roots contain same labels, and the algorithm recursively searches the maximum matching between first-level sub-trees of $A$ and $B$ and store it as a matrix $W$. Based on $W$, the number of pairs in a maximum matching between $A$ and $B$ is found using a dynamic programming method. The *Simple-Tree-Matching* algorithm is shown in Algorithm 2.1.

*The k-means Algorithm*

To date, there are a large number of approaches and algorithms having been developed for clustering analysis in the literature, such as [62, 61, 24, 39], etc. Based on the operation targets and procedures, the major clustering methods can be categorized as: Partitioning methods, Hierarchical methods, density-based methods, grid-based methods, Model-based methods, High-dimensional clustering and constraint-based clustering [62, 71]. Partitioning method is to assign $n$ objects into $k$ predefined groups, where each group represents a data segment sharing the highest average similarity in comparison to other groups. The well-known $k$-means is one of the most conventional partitioning clustering algorithms. The algorithm is expressed as four steps:

1. Arbitrarily choose $k$ data points as initial cluster mean centers;

2. Assign each data to the cluster with the nearest centers, and update each mean center of cluster;

3. Repeat step 2 until all centers don't change and no reassignment is needed;

4. Finally, output subject clusters and their corresponding centers.

## *2.4 Relaxation Technique for QoS-driven Service Selection*

In this section, we survey some related techniques about our QoS-driven web service selection strategy, which are used to select web services with the best quality among those that are able to satisfy users' requirements or needs. Depending on user context, different users may have different preferences about the quality of services they need. In order to fulfill users' personalized preferences and requests, we use skyline model to relax users' constraints and try to find the most possible services which meet users' requirements. The QoS-driven service computation and policing can be achieved through skyline computation.

In the first part of this section, we illustrate the concept of skyline with an example. Then we present some classic algorithms of skyline computation.

### *2.4.1 Introduction to Skyline*

We use the common example in the literature. Suppose we have a group of candidate hotels and we are looking for a hotel that is both cheap and close to the beach. Unfortunately, this goal is hard to reach as the hotels near the beach tend to be more expensive. However, we can at least pick those interesting hotels, which are not worse than any other hotel in both price and distance. The set of interesting hotels chosen by us are called *skyline* [37].

The skyline has its graphical presentation, as shown in Figure 2.10 [14], in which each point stands for one hotel, $x$ axis shows its distance from the beach and its price is represented by $y$ axis. Formally, the Skyline is defined as those points which are not dominated by any other point. A point dominates another point if it is as good or better in all dimensions and better in at least one dimension. For example, suppose we have three hotels,

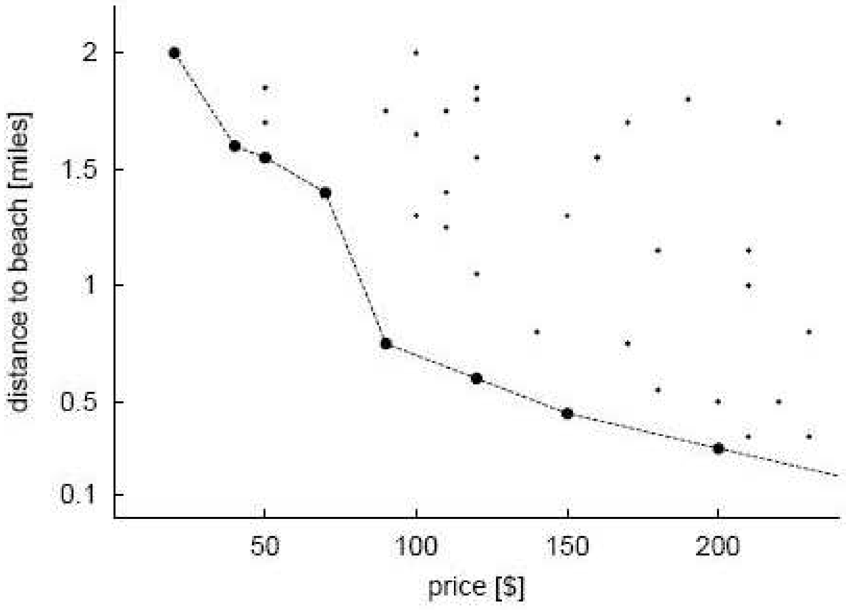


Figure 2.10: Skyline of hotels

where $a = (0.4\texttt{mile}, \$200)$, hotel $b = (0.4\texttt{mile}, \$180)$ and hotel $c = (1.2\texttt{mile}, \$200)$. According to the definition of skyline, we conclude hotel $a$ dominates $c$ and $b$ dominates $a$.

Generally, given a set of $d$ dimensional points $P$, we say that one point $p_1$dominates another point $p_2$ if and only if:

- $p_1$ is better than or equal to $p_2$ on all dimensions, and

- $p_1$ is better than $p_2$ on at least one dimension.

The skyline points are those that are not dominated by any other points in $P$.

### *2.4.2 Algorithms of Skyline Computation*

Computing the skyline is known as the maximum vector problem [79]. Given a set of $d$ dimensional points $P$, for each data point in $P$ a naïve approach is to make dominance checking with all the other points in the dataset. However, the naïve strategy is expensive since every two data points needs to be compared. If we consider the large number of data points, the cost of the naïve approach would be even more expensive. Recently, there are a lot of improved approaches and algorithms [98] having been developed for variants of this problem, such as *divide-and-conquer* [83], *index* [119], *nearest-neighbor* [77], *branch-and-bound skyline* [97], etc.

#### *Divide-and-Conquer*

The divide-and-conquer approach divides the dataset $P$ into several partitions so that each partition fits in memory. Then, the partial skyline of the points in every partition is computed. Finally, all the partial skylines are merged to obtain the full skyline. The shortcomings of the divide-and-conquer approach is that it does not suit large dataset. If the data set is large, the partitioning process requires reading and writing entire data set at least once, resulting in high I/O cost.

#### *Index*

The Index approach divides the data points of $P$ into $d$ lists, where a point $p = (p_1, p_2, ..., p_d)$ is assigned to the $i$th list $(1 \leq i \leq d)$ when $p_i$ is the smallest. Points in each list are sorted in ascending order of their minimum coordinate and indexed by a B-tree. The $i$th list is grouped into some batches, such that each batch consists of points having the same $i$th coordinate. The algorithm begins with loading the first batch of each list and then deal with the one with the minimum coordinate. It computes the skyline inside a batch and, within the computed points, adds the ones not dominated by any of the already-found skyline points into the skyline list. The index method has high I/O cost, because it involves a lot of scan operations to find the best data point for merging. Also, it cannot be used to retrieve the skyline on sub-dimensions of data points.

*Nearest Neighbor*

In the Nearest Neighbor (NN) algorithm, dataset $P$ is indexed by a R-tree. Firstly, NN performs a nearest-neighbor query on the R-tree, to find the point with the minimum distance from the beginning of the axes. Distances are computed according to $L_1$ norm, i.e., the mindist of a point $p$ from the beginning of the axes equals the sum of the coordinates of $p$. All the points in the dominance region are abandoned without further consideration. According to the results of NN search, the dataset $P$ is partitioned recursively. After finding a skyline point, the set of relevant partitions are inserted in a to-do list. NN removes one of the partitions from the list and repeats the process recursively until the to-do list is empty.

*Branch-and-Bound Skyline Algorithm*

The branch-and-bound skyline (BBS) algorithm is also based on nearest-neighbor search on the R-tree created from the data set $P$. In the R-tree, an intermediate entry $e_i$ corresponds to the *minimum bounding rectangle* (MBR) of a node $N_i$ at the lower level, while a leaf entry corresponds to a data point. Similar to the NN algorithm, distances are computed according to $L_1$ norm, i.e., the mindist of a point equals the sum of its coordinates and the mindist of a MBR (i.e., intermediate entry) equals the mindist of its lower-left corner point.

BBS starts from the root node of the R-tree and inserts all its entries in a heap sorted according to their mindist. If an entry is not dominated by some already-found skyline point, then we expand it. That is to say it is removed from the heap and all its children are inserted into the heap. This process repeats until the heap is empty. The details of BBS algorithm can be found in [97].

## *2.5 Modeling and Verification of Web Service Composition*

In this section, we present some models, methods and tools related to our work on verification of transactional properties to ensure a correct composition and a reliable execution of a composite service, given a user's transactional requirements. These related works are used to formally represent and model composite web services.

### *2.5.1 Models*

Recently, numerous formal models and approaches are applied into web service composition. Typical models are *Petri nets*, *Automata*, *Temporal Logic*, and so on.

#### *Petri nets*

A Petri net is a graphical tool for the description and analysis of concurrent processes which arise in distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent places, transitions, and directed arcs[99][91]. Arcs only run between places and transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. Places can have a number of tokens. If there is a token at the end of all input arcs, a transition may fire, which will consume these tokens and place tokens at the end

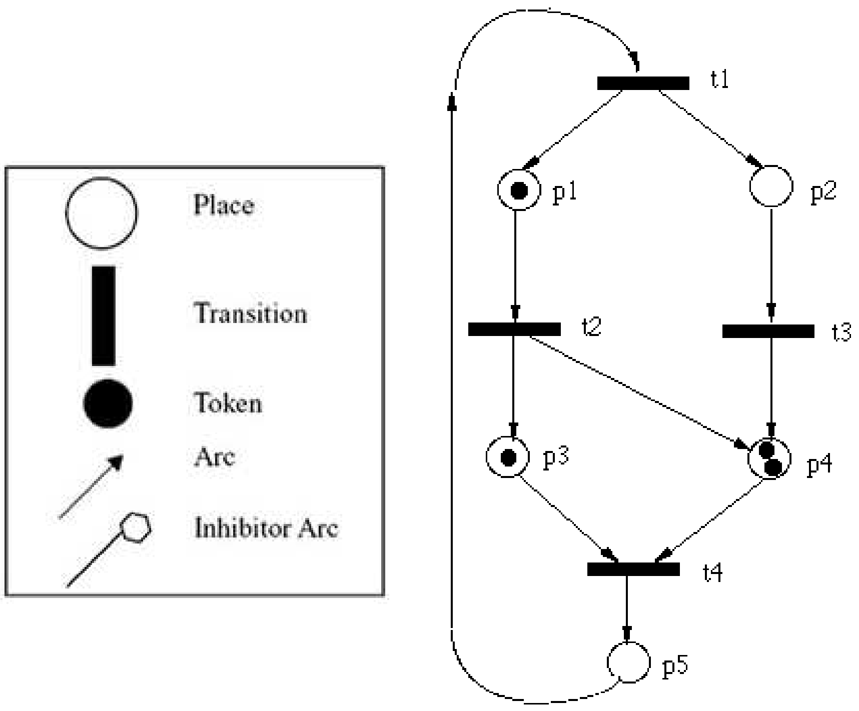


Figure 2.11: Example of a Petri Net

of all output arcs. A firing is atomic, which means it is not interruptible. We borrow the examples from [7] and [12] to show the intuitive meaning of Petri nets, as shown in Figure 2.11.

Just as some existing industry standards (e.g. UML), Petri nets can graphically present choice, iteration, and concurrent execution structures in a process. More importantly, the execution semantics of Petri nets have an exact mathematical definition and a well-developed mathematical theory basis, so if we map a composite web service to a Petri net, the formal model of the composite service can be derived. Moreover, we can apply the verification techniques and tools developed for Petri nets to the context of composite services.

### *Markov Decision Processes for web service composition*

Web service composition can be viewed as a goal-driven problem. *Markov decision processes* (MDPs) are decision-theoretic planning formalisms that model the inherent stochastic nature of web services. Because the number of tasks defined in a composite web service is usually finite, we can use the finite horizon MDP to model the composition problem. The solution of an MDP is a optimal policy that guides a web service composition towards its goal [50][102][58]. An MDP is formally defined as below:

(**Markov Decision Process (MDP)**) *A Markov decision process is a tuple,* $M = (S, A, T, C, H)$ *where*

- $S$ is the set of all possible states;

- $A$ is the set of all possible actions;

- $T$ is a transition function, $T : S \times A \rightarrow \Delta(S)$, which specifies the probability distribution over the next states given the current state and action;

- $C$ is a cost function, $C : S \times A \rightarrow \pi$, which specifies the cost of performing each action from each state; and

- $H$ is the period of consideration over which the plan must be optimal, also known as the horizon, $0 < \mathrm{H} \leq \infty$

We regard a composite web service as a state machine whose set of *states* is the product of the sets of states of the state machines for each service node. We assume that a composite service $C$ is composed of $n$ service nodes, then a system state of $C$ is written as a $N$-tuple: $< s_1, s_2, ..., s_i, ..., s_n > (1 \leq i \leq n)$, where $s_i$ represents the current state of service $i$. *Actions* are web service invocations. The set of invoked web services are bound to the current active service nodes. The *transition* function is defined as the probability of reaching the next state, depending on the current state and action. The *cost* function $C$ defines the cost associated with web service invocation, for example response time, price, etc. We let $0 < \mathrm{H} \leq \infty$ imply that the goal of our web service composition is getting the most optimal workflow policy, for example, by minimizing the total expected response time, etc.

*Modeling Composite Services with Temporal Logic*

In [29] authors modeled workflows as a set of inter-task dependencies. Applying to the context of web service composition, the tasks in a composite service are participant services. The events related to tasks include starting, terminating, and execution of specific functions of services, etc. Every time an event is received for execution, all dependencies are checked so as to accept, reject, or delay the event.

The dependencies are formalized in Computational Tree Logic (CTL) [52], a branching temporal logic by Edmund Clarke and E. Allen Emerson. For example, the order dependency, $e_1 < e_2$, requiring that if $e_2$ occurs then $e_1$ will never occur later at any following path, is specified in CTL as $A \diamond (e_2 \rightarrow A \diamond \neg e_1)$. For each dependency D, a finite state machine $A_D$ is created to enforce $D$. Formally, $A_D$ is a tuple $< s_0, S, \sum, \rho >$, where $S$ is a set of states, $s_0$ is the initial state, $\sum$ is the alphabet, and $\rho \subseteq S \times \sum \times S$ is the transition relation. The elements of $\sum$ can have one of the following forms:

- $a(e_1, e_2, ..., e_n)$: This indicates that events $e_1$,...,$e_n$ are accepted by $A_D$ and scheduled for execution.
- $r(e_1, e_2, ..., e_n)$: This expression says that events $e_1$,...,$e_n$ are rejected by $A_D$.
- $\sigma_1 \parallel ... \parallel \sigma_n$. This says $\sigma_1$,..., $\sigma_n$ are run concurrently in an interleaved fashion.
- $\sigma_1 ; ... ; \sigma_n$. This says $\sigma_1$,..., $\sigma_n$ are run in sequence.

A legal sequence of events must be accepted by every finite state machine. More detailed information can be found in [29] and [89].

### *2.5.2 Tools*

In this section, we introduce two prominent public domain model-checking tools which can also be used to model and verify composite web service and workflows, namely SPIN [65] and CPN tools [73].

### *SPIN*

SPIN is an automata-based model checker for verifying the correctness of distributed software. It verifies systems that are described in Promela (Process Meta Language), which supports modeling distributed algorithms as non-deterministic automata. The properties to be verified by SPIN are described by Linear Temporal Logic (LTL) formulas.

In order to use SPIN to verify composite services, the execution process of composite services, for example expressed by BPEL, needs to be translated into Promela. A lot of works have focused on the translation of BPEL into Promela, such as [55][56][57] etc. We omit their details here.

### *CPN Tools*

CPN Tools is a tool for editing, simulating and analyzing colored Petri Nets [73][90]. Colored Petri Nets (CP-nets or CPN) is a graphical oriented language for design, specification, simulation and verification of systems. It extends the classical Petri Nets with colors (to model data), time (to model durations), and hierarchy (to structure large models). Similar to classical Petri Nets, CPNs uses transition, place, and token as well. Because of these characteristics, CPNs is in particular well-suited for modeling and verifying composite web services. Related works are [138][73][47], etc.

# Chapter 3

# EFFICIENT IR-STYLE SEARCH OVER WEB SERVICES

## *3.1 Introduction*

A web service is programmatically available application logic exposed over the Internet. It can be accessed and invoked via standards such as XML, WSDL, UDDI and SOAP. With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web [34]. Then they can combine individual services into more complex, orchestrated services. As the number of web services and Service Oriented Computing applications increases, there is a growing need for mechanisms for discovering services efficiently. Effective mechanisms for web services discovery and ranking are critical for organizations to take advantage of the tremendous opportunities offered by web services, to engage in business collaborations and service compositions, to identify potential service partners, and to understand service competitors and increase the competitive edge of their service offerings [43].

Web services discovery introduces many new challenges. First, current web services discovery methods are mostly based on the UDDI-registry. To find a service in UDDI, a user needs to browse the relevant UDDI category to locate relevant web services. Considering a large amount of service entries, this process is time consuming and frustrating. So, we need an effective mechanism for automatic web services discovery. Second, a user's requirement for desired web services may not always be precise and a service discovery mechanism can potentially return a large number of results to satisfy the user's requirement, especially when a large service repository is available. Consequently, an important requirement for web services discovery is to rank the discovered results so that the most relevant services appear first. Finally, a good web services discovery mechanism should also be able to assist users in selecting relevant services and combining them. For example, a typical strategy would allow users to see the services before they can start composing their application. Consider the four services shown in Figure 3.1. The second and the third services can process the order information for one transaction provided that a buyer's order has been generated, whereas the first and the fourth services provide the buyer's order according to her requirement. Obviously, it is reasonable to say the first and the fourth services are more important than the others since they contribute indispensable information for the other two services to be invoked. So, an ideal ranking strategy should put the first or the fourth service on top. Also, as we can see from Figure 3.1, there are two links between the first and the second and the third services respectively, in which the output of *CreateOrder* service, *BuyingOrder*, is also the input of both *ProcessPayment* service and *TransportOrder* service. The two links apply to the fourth service, too. This form of link potentially involves more web services and thus is particular useful in web service composition.

To address the problems above, in this chapter we propose a novel IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. The contribution of the work reported here is summarized as follows:

1. We introduce the notion of *preference degree* for web services and then we define

| | |
|---|---|
| WS1: Web Service: CreateOrder | |
| Operation: *OrderBuilder* | |
| Input: UserID | DataType: *int* |
| Requirement | DataType: *ItemList* |
| **Output**: ProductsList | DataType: *BuyingOrder* |
| WS2: Web Service: ProcessPayment | |
| Operation: *CheckoutOrder* | |
| **Input**: UserProducts | DataType: *UserOrder* |
| Output: PaymentConfirmation | DataType: *bool* |
| WS3: Web Service: TransportOrder | |
| Operation: *ShippingOrder* | |
| **Input**: Cargo | DataType: *Order* |
| Output: PickupTime | DataType: *TimeLimit* |
| WS4: Web Service: OrderGeneration | |
| Operation: *GetOrder* | |
| Input: UserName | DataType: *String* |
| **Output**: MyProducts | DataType: *PurchaseOrder* |

Figure 3.1: Sample web-service operations

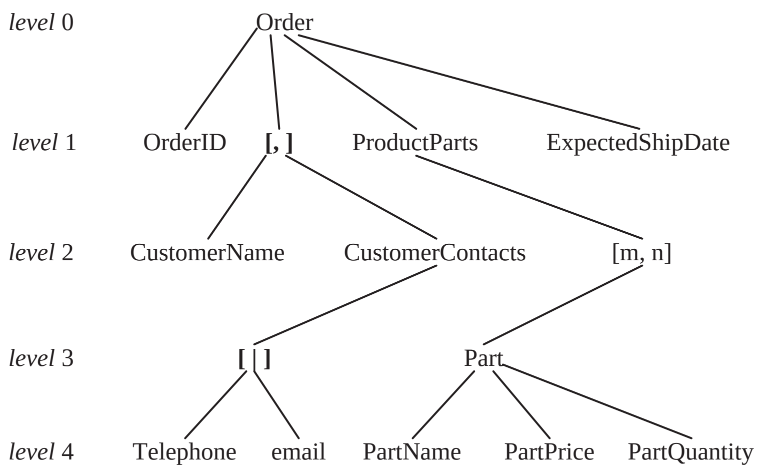


Figure 3.2: XML schema tree of *Order* type

*service relevance* and *service importance*, respectively as two desired properties for measuring the preference degree.

2. We design novel algorithms for computing the relevance and importance degree of services. Our algorithms take into account both textual and structural information of web services.

3. We define *service connectivity*, a novel metric to evaluate the importance of services. In addition, we also develop a new schema tree matching algorithm to measure the service connectivity.

4. We present an approach to identify associations between web-service operations based on service operations matching. This approach uses the concept of *attribute closure* to obtain sets of operations. Each set is composed of associated web-service operations.

5. We do various experiments to search for desired web services. Initial results show that the proposed IR-style search strategy is efficient and practical.

The rest of this chapter is organized as follows. Section 3.2 reviews the related work. Section 3.3 introduces the conception of preference degree for service ranking. Section 3.4 and Section 3.5 present models and definitions for service relevance and service importance, followed by Section 3.6, in which we present algorithms for ranking web services. An approach to identify associations between web-service operations is presented in Section 3.7 based on service operation matching. In Section 3.8 we describe our experimental evaluation. Section 3.9 gives some concluding remarks.

## *3.2 Related Work*

Finding similar web-service is closely related to software components matching. In [142], signatures are used to describe a component's type information (which is usually statically checkable), and formal specifications are defined to describe the component's dynamic behavior. Two components match if their signatures and specifications match. However, the formal specifications used are function's post conditions, which are not available in web services.

Several approaches use text or structural matching to find similar web services for a given web service. The earlier technique tModel presents an abstract interface to enhance service matching process. But the tModel needs to be defined while authors publishing in UDDI [36]. In [108], the authors propose an SVD-Based algorithm to locate matched services for a given service. This algorithm uses characteristics of singular value decomposition to find relationships among services. But it only considers textual descriptions and can not reveal the semantic relationship between web services. Wang etc. [129] discover similar web services based on structure matching of data types in WSDL. The drawback is that simple structural matching may be invalid when two web-service operations have many similar substructures on data types. Woogle [49] develops a clustering algorithm to group names of parameters of web-service operations into semantically meaningful concepts. Then these concepts are used to measure similarity of web-service operations. However, it relies too much on names of parameters and does not deal with composition problem. Further more, it does not provide efficient ranking mechanism to facilitate user browsing services.

Recently, some methods have been proposed to annotate web services with additional semantic information. These annotations are used to match and compose services. For example, in [117] the authors extended DAML-S to support service specifications, including behavior specifications of operations. The Web Service Modeling Ontology (WSMO) [106] is a conceptual model for describing web services semantically, and defines the four main aspects of semantic web service, namely Ontologies, Web services, Goals and Mediators. However, most of existing web services currently use WSDL specifications, which do not contain semantics. Annotating the collection of services requires much effort, and it is infeasible in web services discovery. [113] formally defines a behavior model for web service by automata and logic formalisms. However, the behavior signature and query statements need to be constructed manually, which can be very hard for common users. Also, ranking mechanisms are lacking in these methods.

Other approaches include P2P-based service discovery [63], QoS-based discovery and ranking [26], service crawler [28]. Although these work provide ranking strategies, they overlook user's semantic preferences and can not identify the semantic relationship between services.

Some of our algorithms to be presented in this chapter are related to keyword search in databases or Internet. For example, Discover [66, 67] and DBXplore [25] operate on relational databases and facilitate information discovery on them by allowing users to issue keyword queries without any knowledge of the database schema. They return sets of tuples that are associated by joining on their primary and foreign keys. Google PageRank [38] uses the Internet's link structure as an indication of each web page's importance value. Also, our work is inspired by the work on XML schema. For instance, in [101] authors propose a syntactic approach to web service composition, given only the input-output schema types of web services available in their WSDL descriptions. [139] introduces the concept of schema summary and suggests importance and coverage as two relevant properties by which to judge the quality of a schema summary. Inspired by these methods, we model each web-service operation as a dependency (schema) according to its data types (attributes), by which we design an IR-style ranking mechanism for web services and find associations between them.

## *3.3 Desired Properties for Service Rank*

Our goal is to find services in a more automatic and IR-style way, given a potentially partial specification of the desired service. We need an efficient mechanism to select the preferred services from available ones to satisfy the user's requirement. A natural idea is, firstly, to evaluate the user's preference for available services with respect to the textual service requirement, rank them according to the degree of preference, and then return the top services as search results.

But, what makes a good service to the user? What does "preference degree"mean and how do we compute it? In [38], authors pointed that the final rank of a web page appearing in search results pages is determined by both the goodness of the match of the search terms on the page itself (*relevance* of the page) and this page's PageRank (*importance* of the page). Extending this idea to our context, we consider two factors for the user's preference degree for a service, in other words, the rank of the service. First, a good web service should be relevant to the user's requirement, i.e., to a certain extent, similar to the service requirement. Second, we should select services that are important. Intuitively, a service is important if it is employed by many other services in service composition; therefore, services that can be employed by as many as services are worth looking at.

Having seen what we consider to be desired properties for ranking web services, in the next two sections we will define the *service relevance* metric and *service importance* metric respectively, and then we calculate the user's preference degree for available services in reference to a textual description of desired web services provided by the user.

## *3.4 Service Relevance*

Let $q$ be a natural language description of the desired web services, $S = \{s_1, s_2, ..., s_k\}$ be the set of all available services published through UDDI, and $D = \{D_1, D_2, ..., D_k\}$ be a document collection containing WSDL specifications for all the services in $S$, where each WSDL document $D_i$ corresponds to service $s_i$. Suppose there are $N$ distinct words in $D$ after a pre-processing step, including word stemming, removing stop words and expanding abbreviations and acronyms into the original forms. Applying the vector-space model to the web services context, we describe each service $s_i$ as an $N$-dimensional vector $\overrightarrow{D}_i$ containing all terms in its specification $D_i$, denoted as $\overrightarrow{D}_i = \{(t_1, w_{i1}), (t_2, w_{i2}), ..., (t_N, w_{iN})\}$, where each term is assigned a weight. A well-known weighting method is TF/IDF, namely the normalized *term frequency* (TF) and *inverse document-frequency* (IDF). Typically, the weight for each term $t_j$ in document $D_i$ is given by $w_{ij} = tf_{ij} \times idf_j = tf_{ij} \times \log(k/n_j)$, where $k$ is the total number of available web services and $n_j$ is the number of corresponding WSDL documents in which the term $t_j$ appears. For more details, interested readers are referred to see [109].

Given the weighted vector $\overrightarrow{q}$ for the user's description of desired services and the weighted vector $\overrightarrow{D}_i$ for a service $s_i$'s WSDL specification document, we adopt the cosine distance metric to compare their similarity. Formally, the *service relevance* can be defined as follows:

**Definition 1.** (`Service Relevance` ). *The relevance of a service $s_i$, denoted as $R_{si}$, with respect to the user's natural language description of desired web services, written as $q$, is* *defined as:*

$$R_{s_i} = \frac{\sum_{k=1}^{N} w_{ik} \times w'_{ik}}{\sqrt{\sum_{k=1}^{N} (w_{ik})^2 . \sqrt{\sum_{k=1}^{N} (w'_{ik})^2}}} \tag{3.1}$$

where $w_{ik}$ is the weight for term $k$ in $\overrightarrow{s}_i$ and $w'_{ik}$ is the weight for term $k$ in $\overrightarrow{q}$. $R_{s_i}$ ranges from 0 to 1. The higher score $R_{s_i}$ is, the more relevance service $s_i$ is with respect to $q$, indicating a closer similarity between the user's description of request $q$ and the available web service $s_i$.

## *3.5 Service Importance*

A web service is in some way not different from a software component or module. Like in a software library where different functions or modules have different tier, not all services have equal importance in a web service repository. For example, consider the services in Figure 3.1. Although all their WSDL descriptions contain terms provided by the user to search for desired services, most people would agree that service *CreateOrder* is more important than both the *ProcessPayment* service and the *TransportOrder* service, since the output of the first service, *BuyingOrder*, is also the input of the two services, and thus a required prerequisite for the other two services to be invoked. Based on this observation, we argue that a service can have great importance if there are many other services that employ it, or if there are some services that employ it and have a great importance. From this point of view, a service having low relevance may be more important than a service showing high relevance.

In order to reveal the nature of the importance of a service, we need to identify the relationships between the service and other services. Furthermore, we need an appropriate metric to capture how well a service can be used by other services. In the following subsections, we describe models and algorithms to evaluate service importance; in particular, we show how to measure *connectivity* between two web-service operations based on schema matching, and how by connectivity we can achieve the importance of a web-service operation, which contributes to part of the importance of the service it belongs to.

### *3.5.1 Web-service Operation Modeling*

**Definition 2.** *A web service is a triple $ws = (TpSet, MsgSet, OpSet)$, where TpSet is a set of data types, MsgSet is a set of messages (parameters) conforming to the data types defined in TpSet, and $OpSet = \{op_i(input_i, output_i) | i = 1, 2, ..., n\}$ is a set of operations, where $input_i$ and $output_i$ are parameters (messages) for exchanging data between web-service operations.*

Figure 3.1 has given four web-service operations used as examples in this chapter. According to Definition 2, a web service can be briefly described as a set of operations.

**Definition 3.** *Each web-service operation is a multi-input-multi-output function of the form $f : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$, where $s_i$ and $t_j$ are data types in according with the W3C XML schema specification. We call f a* ***dependency*** *and $s_i / t_j$ a* ***dependency attribute****.*

A dependency attribute can be a complex data type or a primitive data type. Complex data types, such as *BuyingOrder* and *UserOrder* in Figure 3.1, define the structure, content, and semantics of parameters, whereas primitive data types, like *int* and *bool*, are typically too coarse to reflect semantic information. Since parameters usually can be regarded as data types, we can convert primitive data types to complex data types by replacing them with their corresponding parameters. For example, in Figure 3.1 *bool* is converted into *PaymentConfirmation* type while *int* is converted into *UserID* type. Both *PaymentConfirmation* and

*UserID* are considered as complex data types with semantics. Therefore, now each data type defined in a web-service operation can carry a semantic meaning, in according with XML schema specification at the same time.

An XML schema can be modeled as a tree of labeled nodes. We categorize a node *n* by its label:

1. **Tag node**: Each tag node *n* is associated with an element type *T*. *T* is also the tag name of node *n*.

2. **Constraint node**:

   - **Sequence node**: A sequence node indicates its children are an ordered set of element types. We use [ , ] to denote a sequence node.
   - **Union node**: A union node represents a choice complex-type, that is, the instance of which can only be one of the children types in accordance with the XML Schema specification. We use [ | ] to denote a union node.
   - **Multiplicity node**: Each node may optionally have a multiplicity modifier [*m*, *n*] indicating that in the instance, its occurrence is between *m* and *n*. This corresponds to the *minOccurs* and *maxOccurs* constraints in an XML Schema. We use [*m*, *n*] to denote a multiplicity node.

As an example, the schema tree of data type *Order* is shown in Figure 3.2.

### *3.5.2 Connectivity of Web Service Operations*

As we can see, data types defined in web-service operations carry semantic information. Intuitively, we can consider two web-service operations, say *A* and *B*, connected if the data types of the output parameters of $A$ are the same as the data types of the input parameters of $B$, so service $B$ could directly employ service $A$'s output result and they can potentially collaborate in a user's web-service composition process. Obviously, however, requiring that $A$'s output and $B$'s input are the same so as to be connected is too strict and not practical in many cases. Generally, the connectivity relationship between two web-service operations can be defined formally as follows.

**Definition 4.** (`Web-service Operation Connectivity`). *Given two web-service operations* $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$ *and* $op_2 : u_1, u_2, ..., u_l \rightarrow v_1, v_2, ..., v_k$*, let* $X = \{t_1, t_2, ..., t_m\}$ *and* $Y = \{u_1, u_2, ..., u_l\}$*. The connectivity of* $op_1$ *with respect to* $op_2$ *can be measured as the similarity degree between X and Y, denoted as* $Con_{op_2 \rightarrow op_1} = sim(X, Y)$.

Service operation $op_1$ is said to be *connected* to $op_2$ if the connectivity degree $Con_{op_2 \rightarrow op_1}$ is greater than some threshold value $\lambda (0 < \lambda < 1)$. If $op_1$ has exactly the same output data type as $op_2$'s input, we will have $Con_{op_2 \rightarrow op_1} = sim(X, Y) = 1$, indicating the highest possible degree of connectivity of $op_1$ regarding $op_2$. In this case, we say $op_1$ is *well connected* to $op_2$. On the contrary, if the output of $op_1$ is totally different from $op_2$'s input, we will have $Con_{op_2 \rightarrow op_1} = sim(X, Y) = 0$, indicating the lowest possible degree of connectivity of $op_1$ regarding $op_2$.

As we have known, a data type used in web service operations is a structure that is presented by a schema tree, so $X$ and $Y$ are actually two groups of schema trees. Therefore, we can convert the problem of measuring connectivity between two web-service operations to the problem of schema tree matching. Section 3.6 will detail the algorithms for deriving the connectivity of a web service operation.

### *3.5.3 Importance of Web Service Operations*

Based on the notion of connectivity, the web-service operation importance is given by an iterative equation below, similar to the technique used in PageRank [38] algorithm:

**Definition 5.** (`Web-service Operation Importance`). *The importance of a web-service operation op, written as $I_{op}$, is calculated as the following iterative formula until convergence is reached:*

$$I_{op}^{r} = (1-p) + p * \sum_{j \in F_{op_j}} Con_{op_j \rightarrow op} * I_{op_j}^{r-1} * 1/N_{op_j} \tag{3.2}$$

where $Con_{op_j \rightarrow op}$ is the connectivity degree of $op$ with respect to service operation $op_j$, $r$ denotes the number of iterations, $F_{op_j}$ is the set of service operations connected by $op_j$, $N_{op_j} = |F_{op_j}|$ is the number of operations in $F_{op_j}$, and $0 \leq p \leq 1$ is a tuning parameter indicating how well the importance of a web service operation is affected by that of others. For all web-service operations, the initial importance $I_0$ is set to $1/N$, where $N$ is the total number of available web-service operations. The computing process of the iterative equation above is shown by the *CompImp* algorithm in Section 3.6.2.

## *3.6 Algorithm for Ranking Web Services*

We now turn to the main focus of this chapter, which is efficiently ranking web services. Recall that in Section 3.3, two factors are considered for the rank of a service: service relevance and service importance. Since service relevance has been discussed in Section 3.4, now the key issue remaining is how to compute the importance of services. We start with computing the connectivity of web-service operations by a new schema matching strategy, then iteratively compute the importance of operations by using the *CompImp* algorithm

(Algorithm 3.3). Finally, we combine these two factors to achieve the final rank scores for all web-service operations.

### *3.6.1 Computing Connectivity Using Schema Tree Matching*

In this section, we propose a new schema tree matching algorithm to measure the connectivity of a web-service operation, which is also a key step for evaluating its importance.

#### *Tree Edit Distance*

Many works have been done on the similarity computation on trees. Among them *tree edit distance* is one of the efficient approaches to describe difference between two trees. We introduce tree edit operations first. Generally, the tree edit distance operations include: (a) *node removal*, (b) *node insertion*, and (c) *node relabeling*. Such a set of operations can be represented by a mapping with minimum cost between the two trees. The concept of mapping is formally defined as follows [105]:

**Definition 6.** *Let $T_x$ be a tree and let $T_x[i]$ be the* i*th node of tree $T_x$ when traversing it in preorder. A mapping between a tree $T_1$ and a tree $T_2$ is a set* M *of ordered pairs $(i, j)$, satisfying the following conditions for all $(i_1, j_1)$,$(i_2, j_2) \in M$*

1. *$i_1 = i_2$ iff $j_1 = j_2$;*
2. *$T_1[i_1]$ is on the left of $T_1[i_2]$ iff $T_2[j_1]$ is on the left of $T_2[j_2]$;*
3. *$T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$.*

Figure 3.3 gives an example of tree mapping. This mapping also shows the way of transforming the left tree to the right one. A dotted line from a node of $T_1$ to a node of $T_2$

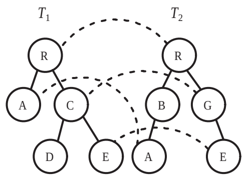


Figure 3.3: An example of tree mapping

indicates that the node of $T_1$ should be changed if the corresponding nodes are different, remaining unchanged otherwise. Nodes of $T_1$ not connected by dotted lines are deleted, and nodes of $T_2$ not connected are inserted. Each of these operations is assigned a cost. The tree edit distance between two trees is defined as the minimal set of operations to transform one tree into the other.

Our schema matching algorithm is based on tree edit distance. However, the problem in our case is more complex than the traditional tree edit distance for the following reasons:

1. The labels of an XML Schema tree can carry complex type information (e.g., union, multiplicity) which makes simple relabeling operations inapplicable. For instance, let $T_1$ and $T_2$ be the schema trees of *Order* and *BuyingOrder* respectively. Let us assume there exits a mapping $M$ between $T_1$ and $T_2$, and there are two node-mapping pairs $(i_1, j_1)$,$(i_2, j_2) \in M$, where $T_1[i_1]$ =[*telephone*|*email*], $T_2[j_1]$ =*email*, $T_1[i_2]$=*price*, and $T_2[j_2]$=*quantity*. The edit operation of $(i_1, j_1)$ should have a smaller cost than that of $(i_2, j_2)$. But the existing works consider all tree edit operations to have same unit distance.

2. The labels of nodes carry semantic information. So a relabeling from one node to another unrelated node will have a bigger cost than a semantic related node. For

example, relabeling *part* to *item* is less costing than relabeling *price* to *email*.

3. We argue that tree edit operations on low-level nodes of a tree should have more influence than operations on high-level nodes. For example, in Figure 3.2, node *Order* is more important than node *PartPrice*, because *Order* denotes broader semantics information than *PartPrice*. So, if a *PartPrice* node of the first tree is mapped into an *Order* node of the second tree, the edit operation cost should not be zero. But traditional works on tree edit distance do not consider the difference and assign a unit cost to each edit operation.

In the next section, we present a new cost model to compute the cost of tree edit operations, by which the tree edit distance of two schema trees is achieved.

### *Cost Model*

Measuring similarity between two XML schema trees equals to finding a mapping with minimum cost. So, the cost of each edit operation involved in the mapping needs to be computed first. [93] proposed an algorithm for fast computing tree edit distance, but it assigns the same cost to each unit edit operation on all nodes and overlooks node difference. In [133], the authors introduced a summary structure for computing structural distance and took weight information into account for nodes in distance computation, but they did not consider the semantic difference or similarity. In this section we introduce a new cost mode based on tree edit distance as presented in [93] and [133]. The new cost model integrates weights of nodes and semantic connections between nodes. Let $T_1$ and $T_2$ be two schema trees and let $n$, $node_1$ and $node_2$ be tree nodes. Formally, the cost model is defined as

$$cost(\rho) = \begin{cases} weight(n)/W(T_1,T_2), & \text{if}\rho = insert(n) \\ weight(n)/W(T_1,T_2), & \text{if}\rho = delete(n) \\ \alpha \times wd(node_1,node_2) & \text{if}\rho \text{ relabels} \\ +\beta \times sd(node_1,node_2) & node_1 \text{to } node_2 \end{cases} \tag{3.3}$$

where $\rho$ indicates a tree edit operation. $weight(n)$ shows the weight of node $n$, which is defined in Definition 9. $wd(node_1,node_2)$ and $sd(node_1,node_2)$ give the weight and semantic difference of $node_1$ and $node_2$, respectively. $\alpha$ and $\beta$ are weights of *wd* and *sd*, satisfying $\alpha+\beta=1$. $W(T_1,T_2)$ is defined as $W(T_1,T_2) = weight(T_1) + weight(T_2)$, where $weight(T_i)$ is the sum of all node weights of tree $T_i (i = 1,2)$. $wd(node_1,node_2)$ is defined as

$$wd(node_1,node_2) = \frac{\|weight(node_1) - weight(node_2)\|}{W(T_1,T_2)} \tag{3.4}$$

where $node_1 \in T_1$ and $node_2 \in T_2$ .

In Equation 3.3, $weight(n)/W(T_1,T_2)$ explains the cost of inserting or deleting node $n$. For the relabel operation, both weight and semantics of $node_1$ and $node_2$ can be different, so we use the combination of weight and semantic difference as the relabel cost. All costs are normalized by $W(T_1,T_2)$, i.e. the sum of all node weights of tree $T_1$ and $T_2$.

In the next two sections, we propose a set of schema-tree transformation rules and a semantic similarity measure to compute *wd* and *sd*, i.e., the weight and semantic difference of nodes.

### *XML Schema Tree Transformation*

**Definition 7.** *The tag name of a node is typically a sequence of concatenated words, with the first letter of every word capitalized (e.g., ExpectedShipDate). Such a set of words is* *referred to as a* ***word bag****. We use* $\pi(n)$ *to denote the word bag of node n.*

**Definition 8.** *Two word bags* $\pi(n_1)$ *and* $\pi(n_2)$ *are said to be equal, only if they have the same words.*

Two nodes are considered different if they have different word bags. The word bag reflects the semantic meaning of a node. As we shall see later, using word bags we can measure the semantic similarity between two schema-tree nodes.

**Definition 9.** *Let* $level(n)$ *denote the level of node n in the schema tree* $T$*. The weight of node n is defined by the following weight function:*

$$weight(n) = 2^{depth(T)-level(n)}(\forall n \in T) \tag{3.5}$$

The weights of all nodes fall in the range of $[2, 2^{depth(T)}]$. Each weight reflects the importance of a node in schema tree $T$. From Section 3.6.1.1, it can be seen that traditional tree-edit-distance algorithm is not suitable for XML schema trees, as it does not deal with constraint nodes. We propose three transformation rules to solve this problem. These rules are used to transform constraint nodes, specifically, sequence nodes, union nodes and multiplicity nodes, to tag nodes. At the same time, the weights of nodes are reassigned according to the following rules:

1. *split*: This rule is applied to sequence nodes. A sequence node $l = [l_1, l_2, ..., l_s]$ is split into an ordered list of nodes $l_1, l_2, ..., l_s$, where $l_i (i = 1, 2, ..., s)$ is a child node of the sequence node $l$. After the split process, each sequence node is replaced by its child nodes. Each child node $l_i$ inherits the weight of its parent node $l$ as a new weight. Figure 3.4(a) gives an example of the split rule.

```
input  : schema tree $T$
output: transformed schema tree $T^*$
d = GetDepth(T);
for $i \leftarrow d$ to 0 do
    foreach node $p \in level_i$ do
        if p is a sequence node then
            weight(each of p's child nodes)=weight(p);
            add p's child nodes to p's parent's child list;
            delete p;
        end
        if p is a union node with s options $\{l_i | i = 1, ..s\}$ then
            merge p's child nodes into a new node q;
            add q to p's parent's child list;
            $weight(q) = weight(p) \times s$;
            $\pi(q) = \bigcup_{i=1}^{s} \pi(l_i)$ ;
            delete p;
        end
        if p is a multiplicity node $[m, n]$ then
            add p's child node to p's parent's child list;
            weight(p's child node)=$weight(p) \times (m + n)/2$;
            delete p;
        end
    end
end
```

**Algorithm 3.1**: Bottom-up-transformation

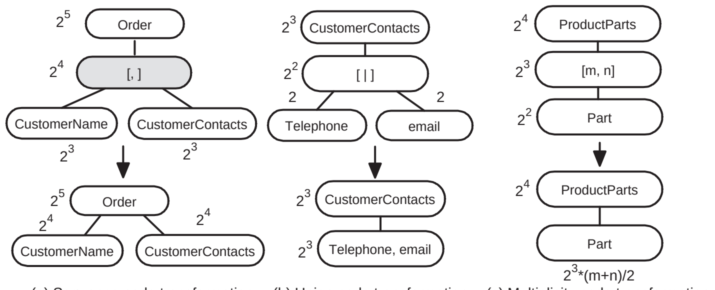


Figure 3.4: XML Schema tree transformation

2. *merge*: This rule is applied to union nodes. After the merge process, each union node is replaced by all its option nodes, i.e. all its child nodes. All child nodes of the union node $l = [l_1|l_2|...|l_s]$ are merged into a new node $l^*$, while the union node $l$ is deleted. The weight of node $l^*$ is $s$ times the weight of $l$. Each $l_i$'s$(i = 1, 2, ..., s)$ word bag is also merged into a new word bag. Formally, we have $weight(l^*) = weight(l) \times s$. Figure 3.4(b) gives an example of the merge rule.

3. *delete*: This rule is applied to multiplicity nodes. We delete a multiplicity node $l = [m, n](m, n \in N)$ and scale up the weight of each of its child nodes $l_i$. After the deletion process, each multiplicity node is replaced by its child nodes. We have $weight(l_i) = weight(l) \times (m+n)/2$. Figure 3.4(c) gives an example of the delete rule.

Note that the definition of complex types can be nested according to the XML schema specification. Thus, given a schema tree, we apply the three transformation rules to its nodes level by level, from bottom to top. This process is formally described as the *Bottom-up-transformation* algorithm (see Algorithm 3.1). The time complexity of Bottom-up-transformation is $O(n)$, where $n$ is the number of nodes in the XML schema tree.

*Semantic Measurement between Schema-tree Nodes*

After the bottom-up transformation, schema tree $T$ is converted into a new schema tree $T^*$. Each node $n$ of $T^*$ is a tag node, whose word bag may come from two or more word tags because of nodes merge by the merge rule. Formally, node $n$ can be regarded as a vector $(W,B)$, where $W$ is the weight of node $n$ and $B$ is the word bag of node $n$. As we can see, after the transformation the weight difference between two nodes can be computed by the new cost model. In this section, we present a strategy to determine the semantic similarity of two schema-tree nodes, i.e. the semantic distance between two word bags.

Our idea relies on a hypothesis that two co-occurrence words in a WSDL description tend to have the same semantics. We exploit the co-occurrence of words in word bags to cluster them into meaningful concepts. To improve accuracy of semantic measurement, we first carry out the pre-processing step before words clustering, which has been done in the service relevance computation.

Let $I = \{w_1, w_2, ..., w_m\}$ be a set of words. These words come from word bags of all schema-tree nodes to which a similarity measurement is applied. Let $D = \{D_1, D_2, ..., D_k\}$ be a document collection containing WSDL specifications for all available services. We introduce association rules to reflect the notion of word co-occurrence. An *association rule* is an implication of the form $w_i \rightarrow w_j$, where $w_i, w_j \in I$. The rule $w_i \rightarrow w_j$ holds in the descriptions set $D$ with *support* $s$ and *confidence* $c$, where $s$ is the probability that $w_i$ occurs in a web-service description and $c$ is the probability that $w_j$ occurs in an web-service description, given $w_i$ is known to occur in it. All association rules can be found by the A-Priori algorithm [75]. We are only interested in rules that have confidence above a certain threshold $t$.

We use the agglomeration algorithm [75] to cluster words set $I = \{w_1, w_2, ..., w_m\}$ into a concept set $C = \{C_1, C_2, ...\}$. There are three steps in the clustering process. It begins with each word forming its own cluster and gradually merges similar clusters.

1. Set up a confidence matrix $M_{m \times m}$. $M_{ij}$ is a two-dimensional vector $(s_{ij}, c_{ij})$, where $s_{ij}$ and $c_{ij}$ are the support and confidence of association rule $w_i \rightarrow w_j$, respectively.

2. Find the two-dimensional vector $M_{ij} = (s_{ij}, c_{ij})$ with the largest $c_{ij}$ in the confidence matrix $M$. If, for both of them, $c_{ij} > t$ and $s_{ij} > t$ then merge these two clusters and update $M$ by replacing the two rows with a new row that describes the association between the merged cluster and the remaining clusters. The distance between two clusters is given by the distance between their closest members. There are now $m-1$ clusters and $m-1$ rows in $M$.

3. Repeat the merge step until no more clusters can be merged.

Finally, we get a set of concepts $C$. Each concept $C_i$ consists of a set of words $\{w_1, w_2, ...\}$. To compute semantic similarity between schema-tree nodes, we replace each word in their word bags with its corresponding concept, and then use the TF/IDF measure. After finishing the schema-tree transformation and semantic similarity measure, the tree edit distance can be applied to match two XML schema trees by the new cost model.

*Obtaining Connectivity*

As it has been mentioned before, we use tree edit distance to match two schema trees. It is equivalent to finding the minimum cost mapping. Let $M$ be a mapping between schema trees $T_1$ and $T_2$, let $S$ be a subset of pairs $(i, j) \in M$ with distinct word bags. Let $D$ be the set of nodes in $T_1$ that are not mapped by $M$, and $I$ be the set of nodes in $T_2$ that are not mapped by $M$. The mapping cost is given by $C = Sp + Iq + Dr$, where $p$, $q$ and $r$ are the costs assigned to the relabel, insertion, and removal operations according to the cost model proposed in Section 3.6.1.2. We call $C$ the *match distance* between $T_1$ and $T_2$, denoted as $C = ED(T_1, T_2)$. Match distance reflects semantic similarity of two schema trees. Now let us see how to compute the connectivity of a web-service operation.

Given two web-service operations $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$ and $op_2 : u_1, u_2, ..., u_l \rightarrow v_1, v_2, ..., v_k$, let $X = \{t_1, t_2, ..., t_m\}$ and $Y = \{u_1, u_2, ..., u_l\}$. The connectivity of $op_1$ with respect to $op_2$ is $Con_{op_2 \rightarrow op_1} = sim(X,Y)$. To achieve $sim(X,Y)$, for each schema tree $\in X$, we find its corresponding schema tree $\in Y$ with the minimum match distance. We simply identify all possible matches between two lists of schema trees $X$ and $Y$, and return the source-target correspondence that minimizes the overall match distance between the two lists. It does not depend on whether the number of schema trees between $X$ and $Y$ is the same or not. This process is illustrated by Algorithm 3.2.

```
input  : $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$
         $op_2 : u_1, u_2, ..., u_l \rightarrow v_2, ..., v_k$
output: The connectivity of $op_1$ with respect to $op_2$

1 for $i \leftarrow 1$ to $m$ do
2     $S_i = min\{ED(t_i, u_j) | j = 1, 2, ..., l\}$;
3 end
4 $Con_{op_2 \rightarrow op_1} = \sum_{i=1}^{m} S_i$
```
**Algorithm 3.2**: Algorithm for computing web-service operation connectivity

### *3.6.2 The* CompImp *Algorithm*

Based on the strategies proposed in Section 3.5.3, we design the *CompImp* algorithm for automatically computing the importance of a set of given web-service operations $OP = \{op_1, op_2, ..., op_N\}$. The algorithm iteratively computes the importance values for all operations until convergence. It initializes the importance of each service operation to $1/N$ and then iteratively applying Equation 3.2 until the importance values converge, i.e., for each operation, the difference between the old and the new importance value is less than some threshold $c$ (typically, we can choose $c = 0.1\%$). The details of *CompImp* are shown in Algorithm 3.3.

Once the importance of all available web-service operations has been obtained, we simply define the importance of a web service as the average of the importance values of all operations in the service. The process is straightforward and not presented due to space limitations.

### *3.6.3 Combining Service Relevance with Importance*

Recall that in Section 3.4, each web service $s$ is assigned a relevance score by similarity measure. In order to reflect the two factors we proposed for characterizing the user's preference degree for $s$, we need to incorporate a service importance score into the relevance score of $s$. Then, we can rank $s$ according to its combination score, which is a weighted sum of its relevance score with a query $q$ and its importance score. Formally, we have

$$Ranking\_Score(q,s) = \begin{cases} w \times R_s + (1-w) \times I_s & \text{if} R_s > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

```
input  : A set of web-service operations $OP = \{op_1, op_2, ..., op_N\}$
output: An array $I[1:N]$ to store the importance values of $OP$
1  foreach service operation $op_i \in OP$ do
2      $I_i^{cur} = 1/N$;
3      $convergence[1:N]$=false;
4  end
5  repeat
6      foreach service operation $op_i \in OP$ do
7          calculate $I_i^{cur}$ using Equation 3.2;
8          if $|I_i^{new} - I_i^{cur}|/I_i^{cur} \leq c$ then
9              $convergence[i] = true$;
10         else
11             $convergence[i] = false$;
12             $I_i^{cur} = I_i^{new}$;
13         end
14     end
15 until convergence[1:N]=true ;
16 Return $I$;
```

**Algorithm 3.3**: The *CompImp* Algorithm

where $0 \leq w \leq 1$. Both $R_s$ and $I_s$ need to be normalized to between [0, 1]. A higher rank score indicates a more desirable web service, so the user's top-k search requirement can be satisfied.

## *3.7 Finding Associated Web-service Operations*

In order to assist users to compose web services, we need to identify potentially relevant web services given a textual description of services. In this section we propose an approach to explore associations between web-service operations based on service operations matching. This approach uses the concept of *attribute closure* to obtain sets of operations. Each set is composed of associated web-service operations.

### *3.7.1 Web-service operations matching*

Similar to the process of obtaining connectivity, now let us see how to match web-service operations. Given two web-service operations $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$ and $op_2 : x_1, x_2, ..., x_l \rightarrow y_1, y_2, ..., y_k$, for each schema tree of $op_1$, we find its corresponding schema tree of $op_2$ with the minimum match distance. We simply identify all possible matches between two lists of schema trees, and return the source-target correspondence that minimizes the overall match distance between the two lists, as shown in Figure 3.5. We formally describe this process in algorithm 3.4.

### *3.7.2 Clustering Web-service Operations*

Suppose $OP = \{op_1, op_2, ..., op_q\}$ is a set of web-service operations and each pair of operations $op_i$ and $op_j$ $(i, j = 1, 2, ..., q)$ match with the distance of $z_{ij}$. We classify $OP$ into a set of clusters $\{op_{c1}, op_{c2}, ...\}$. The clustering algorithm is described as below. It begins

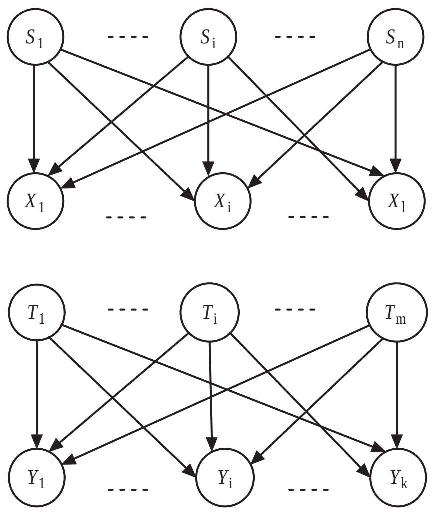


Figure 3.5: Matching web service operations

```
input  : $op_1 : s_1, s_2, ..., s_n \rightarrow t_1, t_2, ..., t_m$
         $op_2 : x_1, x_2, ..., x_l \rightarrow y_1, y_2, ..., y_k$
output: The match distance $Z$ between $op_1$ and $op_2$
1 for $i \leftarrow 1$ to $n$ do
2     $S_i = min\{ED(s_i, x_j) | j = 1, 2, ..., l\}$;
3 end
4 for $i \leftarrow 1$ to $m$ do
5     $T_i = min\{ED(t_i, y_j) | j = 1, 2, ..., k\}$;
6 end
7 $Z = \sum_{i=1}^{n} S_i + \sum_{i=1}^{m} T_i$
```

**Algorithm 3.4**: Algorithm for matching web-service operations

with each operation forming its own cluster and gradually merges similar clusters.

1. Set up a match matrix $M_{q\times q}$. $M_{ij}$ is the match distance of operation $op_i$ and $op_j$.

2. Find the smallest $M_{ij}$ in the match matrix $M$. If $M_{ij} <$ threshold $\delta$ then merge these two clusters and update $M$ by replacing the two rows with a new row that describes the association between the merged cluster and the remaining clusters. The distance between two clusters is given by the distance between their closest members. There are now $q-1$ clusters and $q-1$ rows in $M$.

3. Repeat the merge step until no more clusters can be merged.

Finally, a set of clusters $\{OPC_1, OPC_2, ...\}$ is obtained. Given a cluster $OPC_i$ and an operation $OPC_{ik} \in OPC_i$, $OPC_{ik}$ is called a *pivot* of $OPC_i$ if it minimizes the sum of match distances to all the other operations in $OPC_i$. We consider all operations in $OPC_i$ as *instances* of $OPC_{ik}$.

For example, in Figure 3.1 we give a clustering result. There are two clusters of web-service operations. One is $\{WS1, WS4\}$, and the others are $\{WS2\}$ and $\{WS3\}$. In cluster $\{WS1, WS4\}$ the pivot is *GetOrder* and the instrances of *GetOrder* are *GetOrder* and *OrderBuilder*. In cluster $\{WS2\}$ the pivot is *CheckoutOrder*, which is also an instance of itself.

### *3.7.3 Identifying Associations*

A set of web-service operations is said to be *associated* if they potentially contribute to a user's web-service composition. Clearly, given two web-service operations $op_1$ and $op_2$, if the output attributes of $op_1$ are similar to the input attributes of $op_2$ then $op_1$ and $op_2$

may participate in a user's service composition together. The objective of this step is to find all associations between web-service operations. To do this, we first find associations among clusters $\{OPC_1, OPC_2, ...\}$. Let $OPC_{ik}$, say $x_1, x_2, ..., x_k \rightarrow y_1, y_2, ..., y_j$ be a pivot of $OPC_i$. Let $X = \{x_1, x_2, ..., x_k\}$ and $Y = \{y_1, y_2, ..., y_j\}$.We first compute the *attribute closure* $X^+$ with respect to $X$, which is the set of attributes $A$ such that $X \rightarrow A$ can be inferred by transitivity. At the same time, a pivot set *PS* associated with $OPC_{ik}$ is computed. The overall process is shown as algorithm 3.5.

We perform a worst case time analysis of Algorithm 3.5. The repeat loop is executed at most $|S|$ times, where $|S|$ is the total number of pivots corresponding to all clusters. The calculation of $q$ takes time $|S| - |T|$, where $T$ is the number of pivots in the pivot set *PS*. Hence the total execution time takes in the worst case time $O(S^2)$.

We first choose a pivot $OPC_{ik}$ for each cluster $OPC_i$. For each pivot, we compute a pivot set. We eliminate duplicate pivot sets. If two pivots are in the same pivot set, then their corresponding instances are associated.

Each pivot set $PS = \{p_1, p_2, ..., p_k, ...\}$ can generate a set of operation groups in the form of $\{p'_1, p'_2, ..., p'_k, ...\}$, where $p'_i$ is an instance of $p_i$. Operations in the same group are associated. To obtain an operation group, we simply replace each pivot $p_i$ in *PS* with one of its corresponding instances. All possible operation groups are output as search results.

For example, a pivot set for the clusters given in Figure 3.1 is $\{$*GetOrder*, *ShippingOrder*, *CheckoutOrder*$\}$. It can generate two search results, one is $\{$*GetOrder*, *ShippingOrder*, *CheckoutOrder*$\}$ and the other is $\{$*OrderBuilder*, *ShippingOrder*, *CheckoutOrder*$\}$.

Recall that in Section 3.6, each web-service operation is assigned a ranking score combining both service relevance and service importance. Thus, each operation group can acquire a *group score* by counting the sum of operation scores in it. A higher group score indicates a more desirable search result for web service composition.

```
input  : A pivot $p : x_1, x_2, ..., x_k \rightarrow y_1, y_2, ..., y_j$
output: A pivot set $PS$ containing associated pivots
1 $X = \{x_1, x_2, ..., x_k\}; Y = \{y_1, y_2, ..., y_j\};$
2 $Closure = X;$
3 $PS = \{X \rightarrow Y\};$
4 repeat
5     if there is a pivot $q : U \rightarrow V$ such that the match distance of $U$ and
      $Closure$ is less than threshold $\delta$ then
6         set $Closure = Closure \bigcup V;$
7         set $PS = PS \bigcup q;$
8     end
9 until there is no change ;
```

**Algorithm 3.5**: Algorithm for computing attribute closure and pivot set

## *3.8 Experiments and Evaluations*

We have implemented a prototype system, called IRWService, to evaluate the techniques presented in this chapter. The system architecture is shown in Figure 3.6. First, we developed a service index structure used in our experiments. Then the performance of building the service index and searching with it was evaluated; finally, we evaluate the precision and recall ratio of IRWService search compared with three other methods.

The experiments were conducted on a P4 Windows machine with a 2GHz Pentium IV and 512M main memory. The data set used in our tests is a web service repository collected from [134, 35, 130]. Their WSDL specifications are available so we can obtain the textual descriptions and the XML schemas of input/output data types. The data set contains 525 web services, including 3873 web-service operations. Each web service has an average number of operations of 7.

The web service repository is composed of the WSDL files for every service. In order to improve the performance of searching for desired web services, we design a *service index* for the repository. The service index keeps information about each WSDL document. Ordered by ServiceID, all WSDL documents occupy a fixed width and can be accessed sequentially. The information stored in each item includes various statistics about its corresponding web service, including a *relevance entry* and an *importance entry*. The relevance entry is TF/IDF information, including a list of words occurring in this service, where each word is accompanied by the number of its occurrences in the document and all the service IDs it appears in; whereas the importance entry includes a list of service operations, where each operation is combined with its importance and a list of connectivity it has with respect to other operations. The details of the service index are shown in Figure 3.7. We build the service index as follows. The relevance entry carrying TF/IDF information is implemented using the publicly available Rainbow [68] classification tool, and the importance entry is implemented by our proposed schema tree matching algorithm, based on an existing tree edit distance method [93]. We update the service index when a new web service is added into the repository.

We first evaluated the efficiency of building the service index. The time performance of our algorithm is tested with the increase of the number of web services. Taking CPU time as the standard measure, we get time costs of building the service index in Figure 3.8(a), in which the time includes the costs of constructing relevance entry for all services, and

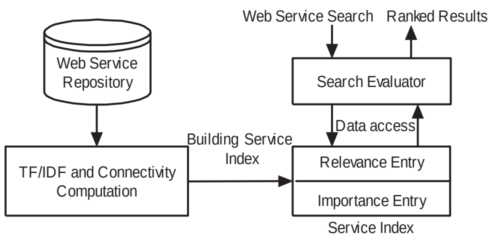


Figure 3.6: System Architecture

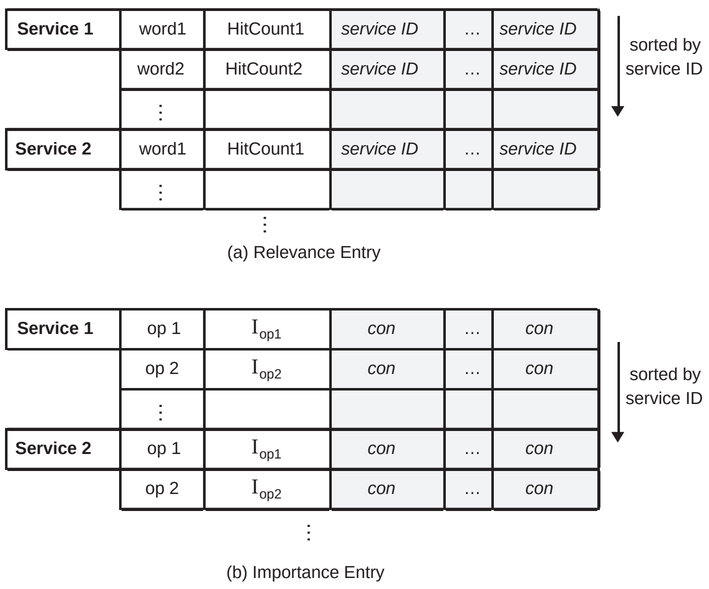


Figure 3.7: Service Index Structure

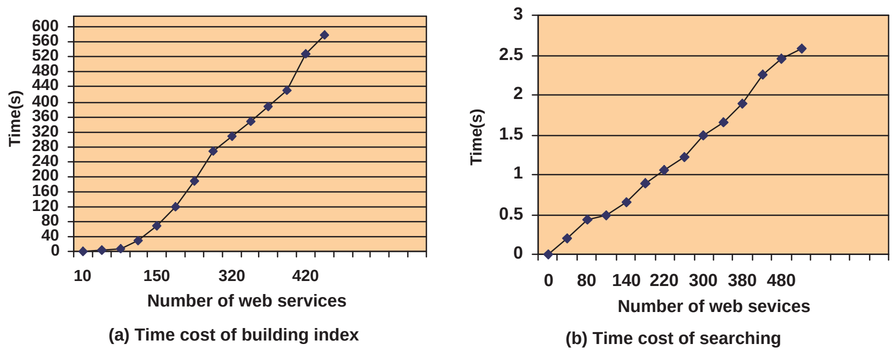


Figure 3.8: Time cost of building index and searching

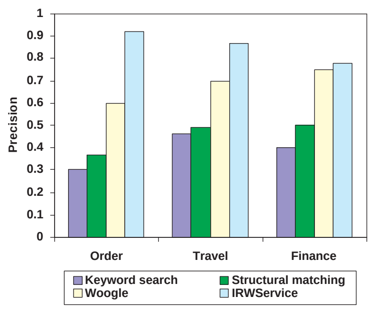
the computation of the importance entry as well. Figure 3.8(a) shows that, as the number of services increases, the time cost of building the service index increases rapidly. This indicates that, by adding web services, the number of schema trees increases, leading to more cost of schema tree matching. To overcome this drawback, we take the linear tree edit distance algorithm from [93] to reduce the time cost of building service index. The reason for choosing Nierman and Jagadish's method [93] as our basis for computing tree edit distance is that it can find the edit distance between pairs of trees of various sizes in an almost linear cost with tree size, considering schema tree matching is time consuming and constitutes the main cost for building the importance entry in the service index.

We then evaluated the efficiency of searching for desired web services. The time cost is given in Figure 3.8(b). It is can be seen that the time increases almost in a linear way with respect to the number of web services. This demonstrates that by building the service index, our searching performance is rather high, although building index is a bit time costing. But considering the fact that the web service repository does not change frequently comparing with a user's query request, the service index is effective and practical.

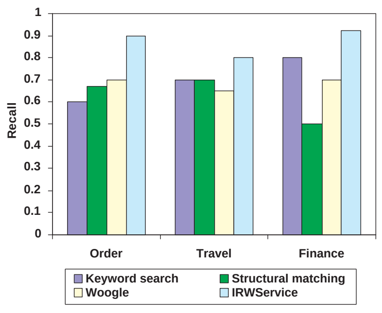
Finally, we used *recall* and *precision* ratio to evaluate the effectiveness of our approach. The precision($p$) and recall($r$) are defined as $p = \frac{A}{A+B}, r = \frac{A}{A+C}$ where $A$ stands for the number of returned relevant services, $B$ stands for the number of returned irrelevant services, $C$ stands for the number of missing relevant services, $A+C$ stands for the total number of relevant services, and $A+B$ stands for the total number of returned services. Specially, the top 50 search results were considered in our experiments for each web service search.

We evaluated IRWService by comparing the recall and precision of search with three other methods: keyword searching method, structure matching [129] and Woogle [49]. We chose 7 web services from three domains: *order*(3), *travel*(2) and *finance*(2). Each web service was used as the basis for desired services. We computed the recall/precision ratio manually and plotted them in Figure 3.9(a) and Figure 3.9(b), respectively. As can be seen, the precisions of IRWService are 92%, 87% and 78% respectively, almost always outperforming that of keyword, structure and Woogle. The precision is higher on *order* services but lower in *finance* services because *order* services have more complex structures and richer semantics in input/output data types. This indicates that, by combining structural and semantic information, the precision of IRWService improves significantly, compared to the results obtained with structural or semantic information only. It also can be seen that by keyword method the precision is rather low whereas the recall is rather high. This demonstrates textual description of web services contains much useful information but also noise at the same time.

Then, we labeled the associated web-service operations in data set manually. The average recall/precision curve is used in Figure 3.10 to evaluate the performance of IRWService on identifying associated operations. This figure illustrates that IRWService can achieve good recall and precision by integrating structural and semantic measurements.



(a)



(b)

Figure 3.9: Precision and recall of IRWService

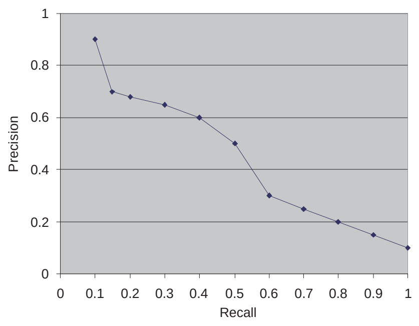


Figure 3.10: Performance of identifying associated operations

## *3.9 Conclusions*

In this chapter, we have presented a novel IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. We have introduced the notion of preference degree for a web service, and suggested *relevance* and *importance* as two desired properties for measuring its preference degree. Also, various algorithms were given to obtain service relevance and importance. The key part for computing service importance is a new schema tree matching algorithm, which catches not only structures, but even better semantic information of schemas defined in web services. Moreover, we developed an approach to identify associations between web-service operations based on service operations matching. This approach uses the concept of attribute closure to obtain sets of operations. Each set is composed of associated web-service operations. Experimental results show the proposed IR-style search strategy is efficient and practical.

As part of on-going work, we are interested in improving efficiency of the connectivity computation algorithm in terms of running time, since the computation of extended tree edit distance is costly. Second, our proposed technique assumes that structures of XML schema are trees. However, their structures may also be graphs and contain cycles. In the future, we plan to extend our algorithm to support graph matching. Finally, in order to further understand the semantics of web services descriptions and integrate more semantic information to our system, we also plan to use WordNet to handle word stems and synonyms to improve the precision of our algorithm.

# Chapter 4

# A RELAXATION-BASED QOS MODEL AND COMPUTATION FRAMEWORK FOR SERVICE SELECTION

## *4.1 Introduction*

A web service is programmatically available application logic exposed over the Internet. It has a set of operations and data types. The current set of web service specifications defines how to specify reusable operations through the Web-Service Description Language(WSDL), how these operations can be discovered and reused through the Universal Description, Discovery, and Integration(UDDI) API, and how the requests to and responses from web-service operations can be transmitted through the Simple Object Access Protocol API(SOAP). With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can combine individual services into more complex, orchestrated services.

Recently, the process-based approach to web service composition has gained considerable momentum and standardization [30]. In this scenario, a service composition can be regarded as a process model containing abstract service specifications, without specifying actual services needed to be bound to the process model. With the increasing number of web services having equivalent functionality, the binding procedure is driven by some non-functional, Quality of Service (QoS) criteria, such as the money cost, response time, reputation, reliability or a trade-off between them [41]. Thus, an important problem is, given QoS constraints, how to aggregate and leverage individual services' QoS information to derive the optimal QoS of the composite service. Actually, dynamic binding of service compositions constitutes one of the most interesting challenges for service-oriented architectures.

A lot of methods have been proposed to solve this problem, including linear programming [144], reduction rules method [42], utility function strategy [140], CP-nets [128] and AND/OR graph method [125], etc. However, most approaches only focus on obtaining optimized service selection under user requirements and overlook user preferences. As an example, consider a user who is searching for the *postcode* of a given city. Table 4.1 gives a collection of candidate web services with the same function. All these services can retrieve the postcode of a given city, although they have different non-functional properties, e.g. *Pri.*(i.e. price($)), *Res.*(i.e. response time(s)), *Rep.*(i.e. reputation score(points/100)) and *Rel.*(i.e. reliability probability). A typical form of QoS query $p$ issued by the user is $M\theta v$, where $M$ is a QoS metric, $v$ is a constant value, and $\theta$ is a comparison operator such as $=,<,>,\leq$ *or* $\geq$. Examples are $p_1$ : *price* $<$ \$10 , $p_2$: *response time*$<11s$, $p_3$: *reputation score*$\geq 70$ and $p_4$:*reliable probability*$> 0.5$, etc. These conditions can also be combined by an "AND" operator, denoted as *price* $<$ \$10 AND *response time* $< 11s$ AND $reputation \geq 70$ AND $reliability > 0.5$, forming a QoS query vector $p = (p_1, p_2, p_3, p_4)$. In order to fulfil the user's request, existing methods need to seek services satisfying all the four conditions above simultaneously and then return them. However, several problems may arise when applying these techniques: firstly, these methods can not solve the *empty result* problem. Since the user issues his requirements without knowing all the detailed QoS properties of all available services, maybe there are no fully satisfying services, i.e. services satisfying all the conditions at one time. In such cases, obviously relaxing

| *Web services* | *Quality of Service(QoS)* | | | |
|---|---|---|---|---|
| | *Pri.* | *Res.* | *Rep.* | *Rel.* |
| $s_1$ | 26 | 5 | 88 | 0.1 |
| $s_2$ | 14 | 10 | 70 | 0.9 |
| $s_3$ | 35 | 9 | 36 | 0.3 |
| $s_4$ | 5 | 1 | 90 | 0.6 |

Table 4.1: Sample web services. Each web service provides the same postcode information.

the user's request and returning some approximate results is a much better idea than just reporting an empty result; secondly, these methods lack flexibility and the personalization problem has been overlooked. Depending on their context, different users may have different preferences about the services they need. For instance, a user may say service $s_2$ is better than service $s_4$, because although $s_2$ is more expensive than $s_4$ and exceeds the user's financial requirement a bit, it has much better reliability, which is crucial to her because of her strict reliability requirements. Similarly, another user running out of money prefers a cheaper service, while a cautious user may prefer services with "excellent" reputation, etc. So, whether a service is good or not in QoS depends on the user's context and preference. It is essential to model the user's personalized preferences and requests.

In this chapter, we present novel techniques to solve the challenges above, and give experimental evidence that shows our methods are efficient and practical. In particular, the main contributions of this chapter are listed as follows:

1. We propose a novel QoS evaluation model for performing flexible and adaptable service selection from the point of view of users. The key idea of the model is to

relax the user's QoS constraints and try to find the most possible services which meet the user's QoS requirements.

2. Based on the proposed QoS framework, we develop various algorithms for making service selection on individual and composite services, respectively. We also introduce a top-*k* ranking strategy to reflect a user's personalized requirements

3. We present the experimental result of a thorough evaluation. Experimental evaluation shows the proposed QoS model is efficient and practical.

The rest of this work is organized as follows. Section 4.2 gives the QoS computing model. Section 4.3 presents different approaches for answering a user's QoS query. Section 4.4 presents a personalized Service Selection strategy. In Section 4.5 we report our experimental results. Section 4.6 discusses related works. Concluding remarks are given in Section 4.7.

## ***4.2 QoS Computing Model***

In this section, we formulate the QoS computing model for individual services and composite services, respectively.

### *4.2.1 Computing Model for Individual Services*

#### *QoS Metrics*

Many QoS metrics have been proposed in the literature. Typical criteria include response time, throughput, price, reputation, reliability, transactional properties and security, etc. In this chapter, we only consider four basic metrics, which are available for almost all web services [82]: price, duration, reputation and reliability.

1. **Price**. This is the amount of money that a service requester has to pay for executing a task. The task may be either an element web service or a composite web service. Users can get execution price via online advertisement or inquiry entry available through service providers. Given a service $s$, we use $price(s)$ to denote the price for executing $s$.

2. **Duration**. The execution duration measures the response time from the submission of a request to the receiving of the response. The average response time can be estimated based on observation of past executions. Given a service $s$, we use $duration(s)$ to denote its execution duration.

3. **Reputation**. The reputation of a service $s$ reflects its trustworthiness. It mainly depends on usage evaluation carried out by end users. Usually, users are given a range, for example from 1 point to 100 points, to rank a web service. We define the reputation of service $s$ as the average ranking given by users during or up to a particular period, denoted as $reputation(s)$.

4. **Reliability**. The reliability of a service is the probability that the service can successfully complete within a time limit. Its value is computed from past invocations history. We use $reliability(s)$ to describe the reliability of a service $s$.

Given the QoS metrics above, the quality of a web service $s$ is defined as a four-dimensional vector: $q(s) = (price(s), duration(s), reputation(s), reliability(s))$. Let $S = \{s_1, s_2, ..., s_n\}$

be a group of web services with the same functional properties. For each service $s_i \in S$, its quality vector is represented as $q(s_i) = (q_i(1), q_i(2), q_i(3), q_i(4))$, corresponding to price, duration, reputation and reliability in order. Given a user's QoS constraints, our algorithms aim to determine the best services in $S$ satisfying the user's requirement. It is worth noting that although the number of metrics criteria is limited in this chapter, our model is extensible. New metric factors can be easily added without fundamentally changing the algorithms for QoS computation. As the quality of a web service $s$ is represented as a four-dimensional vector, we will use *dimension* and *metric* interchangeably in the remainder of the chapter.

*Normalization of QoS Values*

The QoS metrics proposed in Section 4.2.1.1 are not consistent with each other. As we can see, the higher the value of price or duration is, the lower the quality; whereas the higher the reputation or reliability is, the better the quality is. In order to provide a uniform representation of QoS metrics, we need to normalize QoS values before we do QoS computation. Suppose we have the quality vectors for all services $\in S$. For each service $s_i$'s quality vector $q(s_i) = (q_i(1), q_i(2), q_i(3), q_i(4))(1 \leq i \leq n)$, we normalize its values to obtain $q'(s_i) = (q'_i(1), q'_i(2), q'_i(3), q'_i(4))$, where

$$q'_i(j) = \frac{q^j_{max} - q_i(j)}{q^j_{max} - q^j_{min}}, j = price(1), duration(2) \tag{4.1}$$

$$q'_i(j) = \frac{q_i(j) - q^j_{min}}{q^j_{max} - q^j_{min}}, j = reputation(3), reliability(4) \tag{4.2}$$

In the equations above, we can safely assume $q^j_{max} \neq q^j_{min}$, where $q_i(j)$ is the QoS of $s_i$ on metric $j$, $q^j_{max} = Max\{q_i(j)\}, 1 \leq i \leq n$ and $q^j_{min} = Min\{q_i(j)\}, 1 \leq i \leq n$. When a user needs to find services satisfying his or her quality requirements, for consistency, it is also

| *Web* | *Quality of Service(QoS)* | | | |
|---|---|---|---|---|
| *services* | *Pri.* | *Res.* | *Rep.* | *Rel.* |
| $p$ | $> 0.8$ | $> 0.2$ | $> 0.3$ | $> 0.8$ |
| $s_1$ | 0.3 | 0.6 | 0.9 | 0 |
| $s_2$ | 0.7 | 0 | 0.6 | 1 |
| $s_3$ | 0 | 0.1 | 0 | 0.3 |
| $s_4$ | 1 | 1 | 1 | 0.6 |

Table 4.2: Normalization of QoS values of services in Table 4.1

needed to include the QoS query vector $p$ in the normalization process, so we have $q^j_{max} = Max\{\{q_i(j)|i = 1,2,..\} \cup \{p_j\}\}, 1 \leq i \leq n$ and $q^j_{min} = Min\{\{q_i(j)|i = 1,2,..\} \cup \{p_j\}\}, 1 \leq i \leq n$.

**Example 1**. Suppose there are four web services in $S$ and that their values of QoS metrics are given in Table 4.1. We also assume that a user's QoS query vector is $p = (p_1, p_2, p_3, p_4)$, where $p_1$ is *price* $<$ \$10, $p_2$ is *response time* $< 8s$, $p_3$ is *reputation* $> 50$, and $p_4$ is *reliability* $> 0.7$. Using Equation 4.1 and Equation 4.2, we have the normalized Table 4.2.

After normalization, the quality on each metric is monotonic increasing with its corresponding metric value, i.e. the greater the QoS value on one metric is, the better the quality of the service on the metric. So, users only need to issue a QoS query vector using the form of: $M > v$, where $M$ is a QoS metric and $v$ is a constant value related to $M$. For instance, the query vector $p$ in example 1 can be converted into $p' = (p'_1, p'_2, p'_3, p'_4)$, where $p'_1$ is *price* $> 0.8$, $p'_2$ is *response time* $> 0.2$, $p'_3$ is *reputation* $> 0.3$, and $p'_4$ is *reliability* $> 0.8$.

*Answering a User's QoS Query*

In this section, we focus on how to return desirable services from a group of services $S$ with the same function, given a QoS query. Before giving the definition of query result, we first define the quality difference of a user's QoS query $p$ with regard to a service $s_i$'s quality vector $q(s_i)$ as follows:

**Definition 1.** *Suppose $p = (p_1, p_2, p_3, p_4)$ is a four-dimensional QoS query vector, such that $p_k$ is of the form of $M_k > v_k$ ,where $k \in \{1,2,3,4\}$, $M_k \in \{price, duration, reputation, reliability\}$, and $v_k$ is the QoS value of $M_k$. $q(s_i) = (q_i(1), q_i(2), q_i(3), q_i(4))$ is an individual service $s_i$'s quality vector, where $q_i(1)$, $q_i(2)$, $q_i(3)$ and $q_i(4)$ are the QoS value on price, duration, reputation and reliability, respectively. Let $\delta_{ik} = v_k - q_i(k)(k = 1,2,3,4)$. We say $D(p, q(s_i)) = (\delta_{i1}, \delta_{i2}, \delta_{i3}, \delta_{i4})$ is the* individual quality difference *between the user's QoS query vector $p$ and the individual service $s_i$'s quality vector $q(s_i)$.*

Based on the monotonic increasing property of each QoS metric, it is straightforward to draw the following property:

**Property 1.** *The less $\delta_{ik}$ is, the better the quality of service $s_i$ is on metric $k$.*

In order to describe the quality of service $s_i$ relative to a query vector $p$, by Property 1, we say that $s_i$ satisfies $p$ strictly if $\forall k \in \{1,2,3,4\}, \delta_{ik} \leq 0$. $s_i$ is said not to satisfy $p$ strictly if $\exists k \in \{1,2,3,4\}, \delta_{ik} > 0$. Generally, $\forall i \neq j, D(p, q(s_i)) \neq D(p, q(s_j))$, which means service $s_i$ may be better or worse than $s_j$ on some metrics. But, how can we compare these two services as a whole? To formally describe this kind of relationship, we employ dominance checking between the *individual quality difference* of $p$ with regard to different services in $S$.

**Definition 2.** *Given a QoS query vector $p$ and two services $s_i, s_j \in S$, if the value of $D(p, q(s_i))$ on each dimension is not larger than that of $D(p, q(s_j))$ and strictly smaller on at least one dimension, then we say service $s_i$ dominates service $s_j$ with respect to the query vector $p$, denoted as $s_i \succ s_j$. In other words, we can say service $s_i$ is better than service $s_j$.*

For example, consider the running example in Table 4.2. According to Definition 1, we have $D(p,q(s_1)) = (0.5, -0.4, -0.6, 0.8)$, $D(p,q(s_2)) = (0.1, 0.2, -0.3, -0.2)$, $D(p,q(s_3)) = (0.8, 0.1, 0.3, 0.5)$, and $D(p,q(s_4)) = (-0.2, -0.8, -0.7, 0.2)$. $D(p,q(s_4))$ is smaller than $D(p,q(s_3))$ and $D(p,q(s_1))$ on every dimension, but its fourth metric is larger than $D(p,q(s_2))$. Hence, $s_4 \succ s_1$ and $s_4 \succ s_3$.

**Definition 3.** *The query result of a QoS query vector $p$ on a group of services $S$, denoted as $R(p,S)$, is the set of all the services $\in S$, each of which is not dominated by any other service $\in S$.*

In the example in Table 4.2, since $s_4 \succ s_1$ and $s_4 \succ s_3$, we can say $s_1$ and $s_3 \notin R(p,S)$. Also, $s_2$ is not dominated by any other service in $S$. Thus the query result of $p$ on $S$ is $R(p,S) = \{s_2, s_4\}$.

### *4.2.2 Computing Model for Composite Services*

A composite service is actually a business process integrating necessary individual services. So, the quality metrics for individual services are also applied to composite services. The QoS metric value of a composite service is determined by the QoS metric values of its individual services, and the workflow pattern capturing the control flow and dependencies between individual services. There are more than twenty different patterns [121] through which individual services can be integrated to form a composite service, but only four basic workflow patterns among them are essential: *sequential*, for defining an execution order; *parallel*, for parallel routing; *switch*, for conditional routing; and *while*, for looping. Figure 4.1 shows these four basic workflow patterns. Each pattern contains some nodes, which are also called tasks in this chapter. We deal with the sequential workflow pattern first. Later we will see its computing methods with QoS can also be applied to the other three patterns.

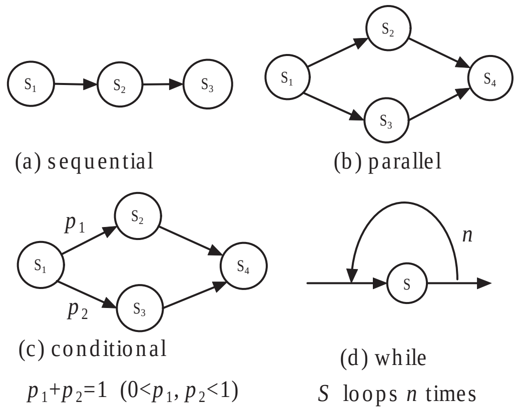


Figure 4.1: Workflow patterns

*Sequential Composition Model*

Now let us see how to compute the QoS metric values of a sequential composite service $s$ with the running pattern like Figure 4.1(a). Using the same quality metrics as individual services, the aggregation functions for $s$ are given below:

1. **Price**. The price of $s$ is the sum of each of its individual service $s_i$'s price. Formally, we have $price(s) = \sum_{i=1}^{n} price(s_i)$.

2. **Duration**. The duration of $s$ is the sum of each of its individual service $s_i$'s duration; that is $duration(s) = \sum_{i=1}^{n} duration(s_i)$.

3. **Reputation**. The reputation of $s$ is the average of each of its individual service $s_i$'s reputation point, denoted as $reputation(s) = (\sum_{i=1}^{n} reputation(s_i))/n$. In this chapter, we map $reputation(s) \times n$ to $reputation'(s)$, thus the function above becomes $reputation'(s) = \sum_{i=1}^{n} reputation(s_i)$.

4. **Reliability**. The reliability of $s$ is the probability product of each of its individual service $s_i$'s reliability probability: $reliability(s) = \prod_{i=1}^{n} reliability(s_i)$. We use the logarithmic function $\ln(x)$ to linearize this function. Let $\ln reliability(s) = \ln \prod_{i=1}^{n} reliability(s_i)$, and $reliability'(s_i) = \ln reliability(s_i)$, then we have a linear aggregation function to compute the reliability of $s$: $reliability'(s) = \sum_{i=1}^{n} reliability'(s_i)$.

Using the aggregation functions above, the quality of a sequential composite service $s$ can be unified as $q(s) = \sum_{i=1}^{n} q(s_i)$. Here $s$ and $s_i$ have been mapped and linearized.

*General Composition Model*

Besides sequential structures, real-world composite services often have loop operations, conditional operations, and parallel operations to run services simultaneously. These operations can be converted into sequential model according to a group of rules similar to [141], [72] and [144]:

1. We unfold a loop operation to a sequential structure by cloning the cyclic nodes $n$ times, where $n$ is the looping count of the loop structure.

2. Both conditional and parallel operations contain multiple branches. Each of the branches can be regarded as an independent sequential operation.

   (a) For a conditional operation, since only one branch is executed with a probability *p* at runtime, we calculate its QoS value by averaging the QoS values of all its branches.

   (b) In parallel structures, all branches are executed simultaneously at runtime, so we need to combine the QoS values of all branches. For price, reputation and reliability metrics, all branch QoS values are summed up as the overall QoS of the parallel operation; whereas for execution duration, the overall QoS value is defined as the maximum QoS value of all branches on this metric.

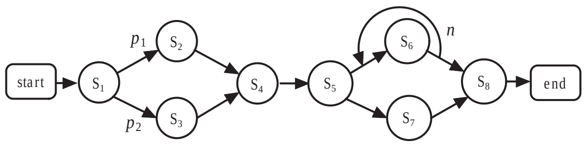


Figure 4.2: Composite service example

As an example, Figure 4.2 gives a composite service $E$ with the four workflow patterns above. $S_1$ is followed by either $S_2$ or $S_3$ with a probability of $p_1$ or $p_2$; $S_5$ is followed by both $S_6$ and $S_7$ in a parallel way. $S_6$ is iterated for at most $n$ times. Using the same notations as the sequential model, the overall *duration* and *price* of $E$ are formulated as follows:

$$\begin{aligned} dur(E) = dur(s_1) + p_1 * dur(s_2) + p_2 * dur(s_3) + dur(s_4) \\ + dur(s_5) + \max(dur(s_6) * n, dur(s_7)) + dur(s_8) \end{aligned} \tag{4.3}$$

$$\begin{aligned} pri(E) &= pri(s_1) + p_1 * pri(s_2) + p_2 * pri(s_3) \\ &+ \sum_{i=4,5,7,8} pri(s_i) + pri(s_6) * n \end{aligned} \tag{4.4}$$

Similar to *price*, the QoS value on *reliability* and *reputation* can be derived the same way. We omit their equations here due to space limit.

*Normalization*

As we have seen in individual services, in order to provide consistent QoS metrics for composite services, we need to normalize each of its individual service's QoS value before we do QoS computation, except that this time the normalization is within all candidate groups of services instead of a single group. Suppose we have a composite service $C$ with $k$ tasks in its execution flow. Each task is executed by a service $S_{ij}(i = 1,2,...,k; j = 1,2,...)$ from service group $S_i(i = 1,2,...,k)$, which consists of $|S_i|$ services having the same functionality as $S_{ij}$. For each service $S_{ij} \in S_i$, its quality vector $q(S_{ij})$ is normalized by the two equations below:

$$q'_{ij}(m) = \frac{q^m_{max} - q_{ij}(m)}{q^m_{max} - q^m_{min}}, m = price, duration \tag{4.5}$$

$$q'_{ij}(m) = \frac{q_{ij}(m) - q^m_{min}}{q^m_{max} - q^m_{min}}, m = reputation, reliability \tag{4.6}$$

where $q_{ij}(m)$ is the quality value of service $S_{ij}$ on metric $m$, $q^m_{max} = Max\{q_{ij}(m)\}, 1 \leq i \leq k, j = 0,1,...,|S_i|$ and $q^m_{min} = Min\{q_{ij}(m)\}, 1 \leq i \leq k, j = 0,1,...,|S_i|$.

Similar to individual services, we also include the user's QoS query vector in the normalization process. Before formally defining the QoS Query on composite services, we give a running example first.

**Example 2**. Figure 4.1 (a) shows a composite service with three sequential functions. Each function is executed by an individual service $S_{ij}$ from service group $S_i(i = 1,2,3)$.

| *Service group* | *Individual service* | *Quality of Service(QoS)* | | | |
|---|---|---|---|---|---|
| | | *Pri.* | *Res.* | *Rep.* | *Rel.* |
| $p$ | | $> 0.8$ | $> 0.2$ | $> 0.3$ | $> 0.8$ |
| $S_1$ | $s_{11}$ | 0.8 | 0.2 | 0.3 | 0.8 |
| $S_1$ | $s_{12}$ | 0.3 | 0.6 | 0.9 | 0 |
| $S_2$ | $s_{21}$ | 0.7 | 0 | 0.6 | 1 |
| $S_2$ | $s_{22}$ | 0 | 0.1 | 0 | 0.3 |
| $S_3$ | $s_{31}$ | 1 | 1 | 1 | 0.6 |
| $S_3$ | $s_{32}$ | 0.9 | 0.1 | 0.2 | 0.7 |
| $S_3$ | $s_{33}$ | 0.4 | 0.3 | 0.5 | 0.2 |

Table 4.3: Example of Composite Service

The QoS metric values of all individual services in each service group are shown in Table 4.3. Also, the QoS query vector for this composite service is given at the first row, i.e. $p = (p_1, p_2, p_3, p_4)$, where $p_1$ is *price* $> 0.8$, $p_2$ is *response time* $> 0.2$, $p_3$ is *reputation* $> 0.3$, and $p_4$ is *reliability* $> 0.8$. Without loss of generality, we assume all these values have been normalized and linearized, as shown in Section 4.2.2.1.

*Answering a User's QoS Query on Composite Services*

Following the definition of QoS query result for individual services, in this section we discuss how to return desirable services contributing to a composite service. In order to simplify the problem, we need to give several definitions first.

**Definition 4.** *We assume $C_i$ is a process with $k$ tasks, each of which is executed by a can-*

*didate service $S_i^{ki}$ from service group $S_i (i = 1,2,...,k)$. Then $C_i = \{S_1^{k1}, S_2^{k2}, ..., S_i^{ki}, ..., S_k^{kk}\}$ is called a* composite service. *We use C to denote the composite service set containing all composite services.*

**Definition 5.** *Given a QoS query vector $p = (p_1, p_2, p_3, p_4)$, where $p_k$ is of the form of $M_k > v_k, k \in \{1,2,3,4\}$, $M_k \in \{price, duration, reputation, reliability\}$ and $v_k$ is the QoS value of $M_k$; and given the quality vector $q(C_i) = (q_i(1), q_i(2), q_i(3), q_i(4))$ of a composite service $C_i$, where $q_i(1)$, $q_i(2)$, $q_i(3)$ and $q_i(4)$ is the QoS value of $C_i$ on price, duration, reputation and reliability, respectively. Let $\delta_{ik} = v_k - q_i(k) (k = 1,2,3,4)$ . We say $D(p, q(C_i)) = (\delta_{i1}, \delta_{i2}, \delta_{i3}, \delta_{i4})$ is the* composite quality difference *between the user's QoS query vector p and the composite service $C_i$'s quality vector $q(C_i)$.*

Clearly, according to the definition and the aggregation functions proposed in Section 4.2.2.1, we can conclude the composite quality difference between a user's QoS query vector $p$ and a sequential composite service $C_i$'s quality vector $q(C_i)$ can be calculated as the sum of all the individual quality difference of $p$ with regard to each candidate service contributing to $C_i$. We formulate this as

$$D(p, q(C_i)) = \sum_{i=1}^{k} D(p, S_i^{ki}) \tag{4.7}$$

where $S_i^{ki}$ is the service from service group $S_i$ contributing to $C_i$.

Similar to individual services, the dominance relationship between composite services is defined below:

**Definition 6.** *Given a QoS query vector p and two services $C_i, C_j$, if the value of $D(p, q(C_i))$ on each dimension is not larger than that of $D(p, q(C_j))$ and strictly smaller on at least one dimension, then we say service $C_i$ dominates service $C_j$ with respect to the query vector p, denoted as $C_i \succ C_j$.*

For example, in Table 4.3, consider a QoS query vector $p = \{0.8, 0.2, 0.3, 0.8\}$ and two sequential composite services $C_1 = \{s_{11}, s_{21}, s_{31}\}$, $C_2 = \{s_{12}, s_{22}, s_{33}\}$. By Definition 5, we have $D(p, q(C_1)) = D(p, q(s_{11})) + D(p, q(s_{21})) + D(p, q(s_{31})) = \{-0.1, -0.6, -1, 0\}$, and $D(p, q(C_2)) = D(p, q(s_{12})) + D(p, q(s_{22})) + D(p, q(s_{33})) = \{1.7, 0.5, -0.5, 1.9\}$. $D(p, q(C_1))$ is smaller than $D(p, q(C_2))$ on every dimension, so $C_1 \succ C_2$. In other words, we can say $C_1$ is better than $C_2$ for satisfying the query vector $p$.

Following the same way as Definition 3, the computing model for composite services is shown as follows:

**Definition 7.** *The query result of a QoS query vector p on a composite service set C, denoted as $R(p, C)$, is the set of all the composite services $\in C$, each of which is not dominated by any other composite service $\in C$.*

In the example of Table 4.3, there are a total of 12 candidate composite services, each with different composite quality difference with regard to query vector $p$. The number of possibilities goes up exponentially with an increasing number of service groups. So, developing efficient algorithms for computing the query result of $p$ on a composite service set is particularly challenging in our case. In this chapter, we design various algorithms to achieve this goal.

## *4.3 Algorithms for Answering a User's QoS Query*

In this section, we present different approaches of answering a user's QoS query on individual services and composite services, respectively.

### *4.3.1 Algorithms for QoS Query on Individual Services*

Recall the model we defined in Section 4.2.1.3. Our goal is to compute the query result of a QoS query vector $p$ on a group of services $S$, $R(p,S)$, i.e., the set of all the services $\in S$, each of which is not dominated by any other service $\in S$.

#### *A naïve Approach*

A naïve strategy for the QoS query is to compute the individual quality difference of every service in $S$ and then, for each quality difference, make dominance checking with all the other quality differences to find the satisfying services. However, the naïve strategy is expensive since every two quality differences are compared. If we consider the rapid increase of available services, the cost of the naïve approach would be even more expensive.

#### *The RC Algorithm*

The main cost the naïve approach is the duplicate comparison between quality differences. This algorithm, called ***R****educe* ***C****omparison* (RC) approach, tries to reduce the number of comparisons by using the comparing result already available. It traverses all the services in $S$ and maintains a set $R$ to keep the satisfying services obtained so far. For each service $s$, the RC algorithm checks if $s$ is dominated by a service in $R$ with respect to the query vector $p$. We discard $s$ if it is dominated; otherwise we insert it to $R$ and discard those services in $R$ dominated by $s$. When all services in $S$ have been traversed, the RC algorithm ends and returns $R$ as the QoS query result. Detailed steps of the RC algorithm are presented in Algorithm 4.1.

```
input  : A QoS query vector $p$, A group of services $S$
output: A query result set $R$ containing satisfying services
1  $R = \phi$;
2  foreach service $s \in S$ do
3      if there exists a service $s' \in R$ such that $s' \succ s$ with respect to $p$ then
4          discard $s$;
5      else
6          inset $s$ to $R$;
7          remove every service $s' \in R$ such that $s \succ s'$ with respect to $p$;
8      end
9  end
10 return $R$;
```

**Algorithm 4.1**: The RC algorithm for QoS query on individual services

*R-tree Based RC Algorithm (RCC)*

```
input  : A QoS query vector $p$, an R-tree $T$ indexing a group of services $S$
output: A query result set $R$ containing satisfying services
1  $R = \phi$;
2  insert into the heap all entries of the root node of $T$;
3  while heap not empty do
4      remove top entry $E$;
5      if there exists a service $s' \in R$ such that $s' \succ E$ with respect to $p$ then
6          discard $E$;
7      else
8          if $E$ is an intermediate entry then
9              foreach child $e_i$ of $E$ do
10                 if $e_i$ is not dominated by a service $s' \in R$ with respect to $p$ then
11                     inset $e_i$ into the heap;
12                 end
13             end
14         else
15             insert $E$ to $R$;
16             // $E$ is a leaf entry, i.e. the quality vector of an individual service;
17         end
18     end
19 end
20 return $R$;
```

**Algorithm 4.2**: The RCC algorithm for QoS query on individual services

The RC algorithm outperforms the naïve approach, as it avoids duplicate comparisons. However, the performance of RC algorithm may decrease as the number of services in $R$ increases. To overcome the problem, the R-tree based RC algorithm (RRC) is proposed to facilitate the retrieval of services. Following most methods in the relevant references, we index the metric values of each quality vector corresponding to each service in $S$. An intermediate entry $E_i$ corresponds to the *minimum bounding rectangle* (MBR) of a node $N_i$ at the lower level, while a leaf entry corresponds to a quality vector. The dominance relationship between services can be easily determined by bound check similar to BBS algorithm in [97]. In the RRC algorithm, the *maximum quality difference* between $p$ and a MBR, say $E$ (i.e., intermediate entry), denoted as $D_{max}(p, q(E))$, equals to the quality distance of its lower bound (e.g. the lower-left corner point in two dimension space) quality vector with respect to $p$, while the *minimum quality difference* between $p$ and $E$ , denoted as $D_{min}(p, q(E))$, equals to the quality distance of its upper bound (e.g. the higher-right corner point in two dimension space) quality vector with respect to $p$. We say a service $s$ dominates a MBR $E$, if the following equation holds: $D(p, q(s)) \leq D_{min}(p, q(E))$, and the inequality is strict on at least one QoS metric, as in this case service $s$ must dominate all services in $E$.

Based on the explanations above, we show how the algorithm works: it starts with the root node of the R-tree and inserts all its entries in a heap. If an entry is not dominated by the query quality vector $p$, then we expand it. That is to say it is removed from the heap and all its children are inserted into the heap. This process repeats until the heap is empty. Detailed steps of RCC algorithm are presented in Algorithm 4.2. As our experiments verified, the RRC algorithm is efficient and its performance is much better than RC.

### *4.3.2 Algorithms for QoS Query on Composite Services*

In this section, we propose methods to process the problem of QoS query on composite services. Recall Definition 7 in Section 4.2.2.3. Now the goal is to compute the query result of a QoS query vector $p$ on a composite service set $C$, denoted as $R(p,C)$, i.e., the set of all the composite services $\in C$, each of which is not dominated by any other composite service $\in C$.

#### *Straightforward Strategy*

Suppose each composite service $C_i \in C$ has $k$ tasks in its execution flow. Each task is executed by a candidate service $S_i^{ki}(i = 1,2,...,k)$ belonging to service group $S_i(i = 1,2,...,k)$, the cardinality of which is $|S_i|$. A straightforward strategy for the QoS query on composite service set $C$ is to enumerate all possible $C_i = \{S_1^{k1}, S_2^{k2},...,S_i^{ki},...,S_k^{kk}\}$ firstly and then compute their composite quality difference with respect to $p$, and finally return $R(p,C)$ by applying the proposed algorithms in Section 4.3.1 on $C$. Clearly, each service in $S_i$ can contribute to $C_i$, so the total number of composite services in $C$ is $\prod_{i=1}^{k} |S_i|$, which leads to much more expensive cost than QoS queries on individual services. If the number of services in service groups or the number of service groups increases, the cost of the straightforward strategy is even more expensive.

#### *Locally Pruning Method*

As we can see, the straightforward approach traverses all services in each service group then combines traversed individual services to find qualifying composite services, so the number of services in each service group is a key factor in the cost, and it would be useful if we can reduce the number of services in service groups. To achieve this, we need to identify and remove those services from each service group, which are impossible to appear in the final query result $R(p,C)$. In this section, the *locally pruning method*(LPM) is developed to address this issue. The intuition behind this approach is that a locally dominated service, i.e., it is dominated by some other service in the same group, does not appear in $R(p,C)$. This idea can be summarized by the following heuristic:

**Heuristic 1.** *Let $S_i$ be a group of services and $S_i^{ki}$ be an arbitrary service $\in S_i$. Service $S_i^{ki}$ can be safely pruned if there exists a service $S_i^{ki'} \in S_i$ such that $S_i^{ki'} \succ S_i^{ki}$.*

*Proof.* According to the four different workflow patterns in Figure 4.1, we give the proof separately.

Firstly, assume $S_i^{ki}$ is selected to a sequential structure $C_i = \{S_1^{k1}, S_2^{k2}, ..., S_i^{ki}, ..., S_k^{kk}\}$ of a composite service in $R(p,C)$. Now we modify $C_i$ by replacing $S_i^{ki}$ with $S_i^{ki'}$ to obtain another sequential structure $C_i' = \{S_1^{k1}, S_2^{k2}, ..., S_i^{ki'}, ..., S_k^{kk}\}$. According to Equation 4.7, we have $D(p,q(C_i)) = \sum_{l=1}^{k} D(p,S_l^{kl})$, and $D(p,q(C_i')) = \sum_{l=1}^{i-1} D(p,S_l^{kl}) + D(p,S_i^{ki'}) + \sum_{l=i+1}^{k} D(p,S_l^{kl})$.The dominance check between $C_i$ and $C_i'$ can be done using formula
$D(p,q(C_i')) - D(p,q(C_i)) = D(p,S_i^{ki'}) - D(p,S_i^{ki})$. Note that $S_i^{ki'} \succ S_i^{ki}$, so the QoS value of $D(p,q(C_i')) - D(p,q(C_i))$ on each metric is not larger than zero and strictly smaller on at least one metric. That is to say, the value of $D(p,q(C_i'))$ on each dimension is not larger than that of $D(p,q(C_i))$ and strictly smaller on at least one dimension. Therefore, $C_i'$ dominates $C_i$ and $C_i$ can not appear in $R(p,C)$, which is contradictory to the fact that $C_i \in R(p,C)$.

As for loop structures and conditional structures, since they can be converted into linear sequential structures according to the conversion rules in Section 4.2.2.2, the conclusion can be derived similarly.

Now we suppose $S_i^{ki}$ is selected to a parallel structure $C_i$ of a composite service in $R(p,C)$. Since $C_i$'s QoS values on price, reliability and reputation can be accumulated, effectively it can be regarded as a sequential structure and the deduction above still applies. Therefore, we only need to prove the heuristic rule works on *duration* metric as well. Without loss of generality, we suppose $C_i$ only contains two service groups $S_i$ and $S_j$, where $S_i = \{S_i^{ki'}, S_i^{ki}\}$, $S_i^{ki'} \succ S_i^{ki}$ and $S_j = \{S_j^{kj}\}$. If $dur(S_j^{kj}) \leq dur(S_i^{ki'})$, then $dur(C_i)$ is determined by $S_i^{ki}$ or $S_i^{ki'}$, so service $S_i^{ki}$ can be safely pruned; if $dur(S_j^{kj}) \geq dur(S_i^{ki})$, then $dur(C_i)$ is determined by $S_j^{kj}$, thus clearly service $S_i^{ki}$ can also be safely pruned; otherwise $dur(S_j^{kj})$ falls in between $dur(S_i^{ki'})$ and $dur(S_i^{ki})$ and we have the dominance relationship between two compositions: $\{S_i^{ki'}, S_j^{kj}\} \succ \{S_i^{ki}, S_j^{kj}\}$, therefore, service $S_i^{ki}$ can be safely pruned, too. $\square$

```
input  : A QoS query vector $p$
         $k$ service groups $S_1, S_2, ..., S_k$
output: A query result set $R$ containing satisfying composite services

1  $R = \phi$;
2  foreach service service group $S_i$ do
3      Locate $S_i^*$ for $S_i$;
4      Obtain $S_i^{'}$ by removing those services from $S_i$ dominated by $S_i^*$;
5  end
6  repeat
7      select one candidate service from each service group $S_i^{'}$;
8      form a composite service $C_i$;
9      if $\exists C_i' \in R$ such that $C_i' \succ C_i$ with respect to $p$ then
10         discard $C_i$;
11     else
12         insert $C_i$ to $R$;
13         remove every service $C_i' \in R$ such that $C_i \succ C_i'$ with respect to $p$;
14     end
15 until no more services can be traversed ;
16 Return $R$;
```

**Algorithm 4.3**: The LPM algorithm for QoS query on composite services

Based on the heuristic rule above, now we consider how to find such $S_i^{ki'}$s for service group $S_i$. For a service $S_i^{ki'}$ in $S_i$, its ability to prune and exclude other services from further consideration is determined by its own QoS value and the extent of the distribution of the QoS values of services in $S_i$. Suppose the value range on metric $k(k = 1, 2, 3, 4)$ is $[a_k, b_k]$ in $S_i$.

Then the pruning ability of $S_i^{ki'}$, denoted as $PA(S_i^{ki'})$, is evaluated as $PA(S_i^{ki'}) = \prod_{l=1}^{4}(b_l - q_{ki'}^l)$. We select the service from $S_i$ with the maximum pruning ability, termed $S_i^*$ to prune non-qualifying services. After pruning non-qualifying services for each service group $S_i$, the straightforward strategy is applied to compute the query result $R(p,C)$. Let $S_i^{'}$ represent $S_i$ from which some services have been pruned. The number of pruned services for $S_i$ is $|S_i| - |S_i^{'}|$. Therefore, the total number of pruned composite services in $C$ is $\prod_{l=1}^{k}(|S_l| - |S_l'|)$. Algorithm 4.3 shows the pseudo-code of the LPM algorithm.

## 4.4 Personalized Service Selection

Having shown QoS models and their corresponding computing algorithms for answering a user's QoS query on both individual services and composite services, we know that our strategy can solve the *empty result* problem, as shown in the introduction section, and can provide at lease one result to satisfy the user's QoS requirement. But meanwhile, another problem appears: the number of services in the query result may exceed the user's requirement and sometime she or he does not need all services in the query result. For example, a strict user may only needs fully satisfying services; whereas a user running out of money prefers a relatively cheaper service with other QoS metrics being satisfied approximately. So, a natural question is how to select good or appropriate services from the returned query result to satisfy the user's personalized requirement. We categorize this question into two cases as follows.

### 4.4.1 Selecting fully satisfying services

For a strict user requiring fully satisfying services, a question to ask is whether these services are included in the returned query result or not. Actually, it is important to point that the query result specified by Definition 3 or Definition 7 is consistent with the user's quality requirements. This fact can be derived from the following lemma.

**Lemma 1.** *The query result of a QoS query vector $p$ on a group of services $S$, $R(p,S)$, contains* ***at lease one*** *fully satisfying service if such services exist.*

*Proof.* Suppose $F$ is a set of all the fully satisfying services. Obviously, there exists a service $a \in F$ such that $a$ is not dominated by any other service in $F$. If $a \notin R(p,S)$, there must exist a non-fully satisfying service $b \in R(p,S)$, such that $b \succ a$. So, by Definition 2 we conclude the value of $D(p,b)$ on each dimension is not larger than that of $D(p,a)$) and strictly smaller on at least one dimension, which is wrong because all the dimension values of $D(p,a)$ are strictly less than zero and at least one dimension value of $D(p,b)$ is larger than or equal to zero. $\square$

Notice that not all fully satisfying services are contained in $R(p,S)$, as there may exist dominance relationship between them. By Lemma 1, we can simply select the fully satisfying services in $R(p,S)$ as the best answers for the strict user. Here we only discuss the case of individual services. For composite services, by slightly modifying Lemma 1, we can obtain fully satisfying composite services similarly.

### *4.4.2 Top-*k *Answers on out-of-range services*

Since we try to obtain the query result using a relaxation method by computing the *quality difference*, rather than seeking services satisfying all the four conditions at one time, our proposed models can still return services to answer a user's QoS query, even when no fully satisfying services are available. Reasonably, the user may only be interested in part of the returned out-of-range services. However, different users may have different preferences for one fixed metric, and even one user may have different preference for different metrics. So, an important problem is how to compute the $k$ best out-of-range services.

In our models, the metrics are treated equally. In order to satisfy the user's personalized QoS requirements, we allow the user to specify a weight for each metric to adjust the *quality difference* when she issues QoS queries. Specifically, the quality difference of a service $s$ with respect to a query vector $p$ is modified as $D(p,q(s)) = (w_1 \times \delta_1, w_2 \times \delta_2, w_3 \times \delta_3, w_4 \times \delta_4)$, where $w_i (i = 1,2,3,4)$ are weights specifying the importance of the quality difference on each metric. Furthermore, we define the user's preference score for $s$ as $w_1 \times \delta_1 + w_2 \times \delta_2 + w_3 \times \delta_3 + w_4 \times \delta_4$. Thus, after the out-of-range service set $R$ is available, we can select $k$ services in $R$ with the $k$ smallest preference scores to satisfy the user's top-$k$ requirement.

The step of finding the top-$k$ answers can also be combined into the algorithms for computing $R(p,S)$ or $R(p,C)$. To achieve this, we use a queue to store the $k$ best services obtained so far. When a new candidate service is available, we compute its preference score and update the queue. For the limit of space, the detailed procedure is omitted here.

## *4.5 Experiments*

In this section, experiments are conducted to evaluate the performance of the proposed algorithms. The experiments are implemented on a P4 Windows XP machine with a 2GHz Pentium IV and 512M main memory. Our experiments are divided into two groups. Group 1 is for evaluating the performance of naïve , RC and RRC algorithms for QoS query on individual services. Group 2 is for evaluating the performance of algorithms for QoS query on composite services, including straightforward method and LPM algorithm. We use the simulation approach in [141] to study the performance of these algorithms. The comparisons of these algorithms are done by running time.

We first evaluate the efficiency of algorithms running on individual services. For simulation, we generate a group of candidate services, the number of which ranging from 1000 to 5000. Four quality values for each service are randomly generated with a uniform distribution between [0, 1]. Figure 4.3(a) shows the result of the three proposed methods as the number of services increases. The test result shows RC is much better than the naïve method. The reason is that in the naïve strategy every pair of services is compared without any pruning. However, RRC is faster than RC, especially when the services are large. This is because RRC uses R-tree to index all service quality vectors, and the number of comparisons is reduced largely with its minimum bounding rectangles (MBRs).

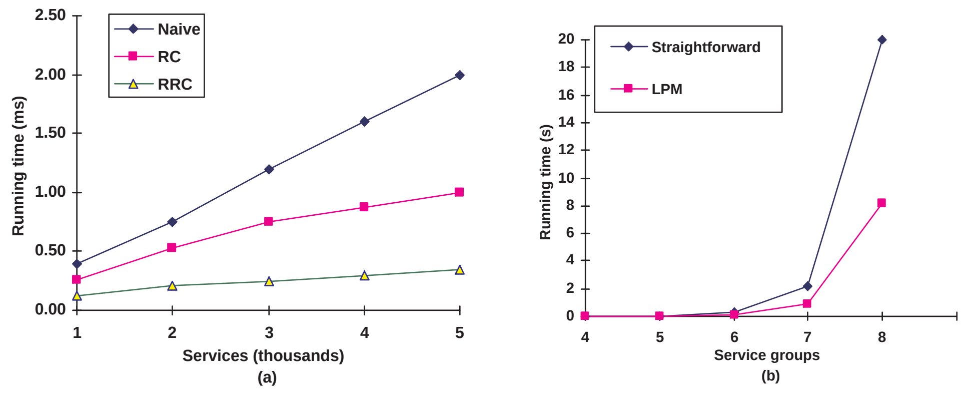


Figure 4.3: Running time comparisons on individual and composite services

Then, the algorithms for QoS query on composite services are evaluated. We randomly construct 4-8 groups of services, each group having 10 candidate services. The results of straightforward and LPM algorithms are shown in Figure 4.3(b). As can be seen, the straightforward method is the most expensive because it has to traverse all services in each service group, which leads to the exponential increase of running time. LPM performs much better, indicating that by pruning the number of services in each group, the efficiency of finding qualifying composite services improves significantly. Although LPM spends time to compute QoS query on each service group first, its performance is still good due to fact that the number of services in each service group is a key factor in the cost.

## *4.6 Related Works*

QoS-based web service selection is an active research area and has attracted many researchers. A lot of methods are proposed in previous work, such as workflow model, global planning strategy using linear programming, graph/tree approach, QoS based selection of semantic web services, etc. In this section, we briefly discuss the relationships between our work and existing methods.

Industrial standard specifications have been proposed to provide infrastructure for web services composition. Among them BPEL4WS (Business Process Execution Language for Web Services) is the most widely used language for process-based service composition. It describes the execution procedure and abstract process of workflow. Other specifications include E-Flow, BPML, etc. WSLA [16] provides a framework for specifying and monitoring Service Level Agreements (SLA) for Web Services. A WSLA document defines assertions of a service provider to perform a service according to agreed guarantees for IT-level and business process-level service quality parameters such as response time and throughput, and measures to be taken in case of deviation and failure to meet the asserted service guarantees.

However, these specifications only provide mechanisms to carry out local services selection in dynamic environment, and QoS optimization from a global view is not supported. Our work is based on these proposals and aims to provide a QoS-based and personalized services selection model for the underlying workflow.

Besides industrial standards, many prototypes have been developed to assist in services composition. In particular, SWORD [100] uses a rule-based engine to realize a composition by existing web services; SELF-SERVE [31] proposes a declarative language to carry out service composition based on state-chart. But these projects only focus on planning or analyzing workflow process, and neither QoS criteria nor QoS optimization issues are addressed.

QoS-based service selection could be considered as a special case of the more general problem of global optimization. [42] presents a QoS model, addressing time, cost, and reliability dimensions. The model computes the quality of service for workflows automatically based on QoS attributes of an atomic task. In [144], the QoS of web services is computed using a muti-dimensional model, and the global QoS optimization is solved by linear programming techniques. In [140], authors present a QoS broker to maximize the user-defined utility value. In the broker, the optimized services selection is modeled as the Multiple Choice Knapsack Problem and the shorted path problem in graph theory. In [128], authors use an AND/OR tree structure to model the service composition problem. The procedure of service composition is implemented through tree traversal, and a heuristic-based search method is proposed to retrieve composite services with top-k QoS values. [125] uses a qualitative graphical representation of preference, CP-nets, to deal with services selection in terms of user preferences. However, these approaches can not handle the cases where there is no fully satisfying service available, and the personalization issue is not discussed either.

Several works are related to QoS-based selection of semantic web services. Specifically, DAML-S and ebXML provide well defined, computer-interpretable semantics for web services. Reference [127] describes a QoS model using the Web Service Modeling Ontology. Also, the idea of QoS model extension has been presented by some researchers, such as [41], [74], etc. In [41], the extended QoS model includes generic and domain or business specific criteria and allows users to express their preferences. [74] models web service configurations, associated prices and preferences using utility function policy, and the optimal service selection combines declarative logic-based matching rules with linear programming optimization methods. However, these works do not solve the empty result problem either. Furthermore, although users' interest is concerned, it is only based on one preference and thus infeasible for practical purpose. To the contrary, our model takes into account all QoS criteria factors and a relaxation strategy is applied to describe users' comprehensive preference.

Another area related to this chapter is skyline query. For example, [37] proposed the skyline operator; [97] developed an optimal and progressive algorithm for skyline queries; [78] introduced a skyline framework for defining the semantics of selection and join queries on relational database. Inspired by these works, we use a relaxation-based approach to perform QoS-based web service selection.

## *4.7 Conclusions*

In this chapter we studied the problem of service selection with QoS constraints. A novel QoS model was proposed to perform flexible service selection. Based on the presented model, we developed various algorithms for making service selection on individual and composite services, respectively. We also introduced a top-*k* ranking strategy to reflect a user's personalized requirement. The performance of the algorithms has been evaluated, showing the proposed QoS model is an efficient and practical strategy to satisfy users' QoS requirements. As part of on-going work, we are interested in improving performance of the QoS query algorithms, as well as investigating more complicated workflow patterns. We also plan to integrate exception handling mechanism into our model during personalized service selection.

# Chapter 5
# VERIFICATION OF TRANSACTIONAL REQUIREMENTS IN WEB SERVICE COMPOSITION

## *5.1 Introduction*

Web services are loosely coupled reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols. Web services provide integration and interaction mechanism for automating B2B interactions. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can dynamically combine individual services to provide new value-added services [34]. The composition of web services provided by different organizations provides an efficient way to build complex application logics. In order to create a new web service satisfying user requirements, it is necessary to provide a web service composition mechanism that supports the combination of existing simple web services and design the interaction relation among them. Therefore, web services composition has been a hot research area. Many works have been done on this issue, including BPEL [1], OWL-S [11], WS-CDL [17], Petri-net [60], planning [95][103], etc.

However, these methods can not address the transactional problem with service composition context. As web services operate in a highly dynamic distributed environment and interact with each other, the possibility of unexpected behavior is high. For example, the invocation of a service may fail because of temporary unavailability of the service. The unexpected behavior from an individual service may bring negative impact on all the component services in the composition, even lead to failure of the running of the composite service. So, we need a strategy that not only allows selecting a set of component web services satisfying a user's functional and non-functional (for example, QoS criteria such as cost, response time, reliability, etc) requirements, but also provides transactional support to business integration via composing individual web services to ensure the overall consistent and reliable execution of a business process. A main problem that remains is, given a user's transactional requirements, how to select individual services and verify their transactional properties to ensure the correct composition and reliable execution of a composite service.

Although there exist several transaction web service standards, they only provide limited transactional support for composite web services without giving much thought to the transactional features [80]. In this chapter, we propose a novel approach to verify at design time whether a service composition can be implemented by a set of web services satisfying transactional requirements specified by a user. Service selection passing the verification can ensure the correct composition and reliable execution of a composite service according to user transactional requirements. In particular, the contributions of this chapter can be summarized as below:

- We define transactional models for component web services. The transactional properties of component web services together with the dependencies between component services, which are defined by the workflow patterns that specify how services are combined together and how the behavior of a service interacts with other services, contribute to the transactional properties of composite services.

- Based on automata theory, we model both component web services and composite services as transition systems, and use the Accepted Termination States to describe the transactional properties of a composite service. Then we propose a novel algorithm to carry out automated verification of Accepted Termination States on the composite service.

- In addition, for the cause of efficiency we use Temporal Logic to describe the transactional properties of a composite service, and then we employ the *SPIN* model checker to carry out automated verification of temporal logic properties on the composite service.

The rest of the work is organized as follows. Section 5.2 provides a brief overview of transactional properties of web services. In Section 5.3, we formally present the model of web services and describe it with automata. In Section 5.4, we discuss our transition-based algorithm on how to ensure the consistent termination of composite web services. In Section 5.5, we discuss the use of SPIN for verification of transactional composite services. Related work is shown in Section 5.6. We give concluding remarks and some future research work in Section 5.7.

## *5.2 Preliminaries*

In this section, we define the transactional properties offered by component services, and the accepted termination states for composite services, which are normally used as the notion of correction of transactional web service composition.

### *5.2.1 Transactional Properties of Services*

Transactions are one of the most fundamental mechanisms for reliable applications. However, traditional transactions which support ACID properties do not apply to the context of web services well because of the characteristics of web services such as long-running, loose coupling, internet basis, etc. Also, a set of component web services can be combined to carry out very complex activities, which makes it very hard for composite services to follow the ACID properties strictly. In order to ensure the reliable execution of web services in distributed environments, we need to add transactional support into web services and their composition. As the first step, we identify transactional properties of elementary services. The main transactional features of an elementary web service that we are considering are *retriable* ($r$), *compensatable* ($c$) and *pivot* ($p$) [34][59][80][87][81]. More specifically, we have the definitions as follows:

**Definition 1.** ***Retriable***. *A service is said to be retriable if it is sure to complete successfully after a finite number of invocations. A retriable service is able to offer forward recovery mechanism.*

**Definition 2.** ***Compensatable***. *A service is said to be compensatable if it is able to offer compensatable supports. A compensatable service has operations to semantically undo the original activity effect.*

**Definition 3.** ***Pivot***. *A service is said to be pivot if it is neither compensatable nor retriable. A pivot service supports atomic transactions which means once it successfully completes, its effects remains forever and cannot be semantically undone. It is worth to note that a pivot service may fail or have no effect at all. In other words, there is no guarantee that this type of service can execute successfully.*

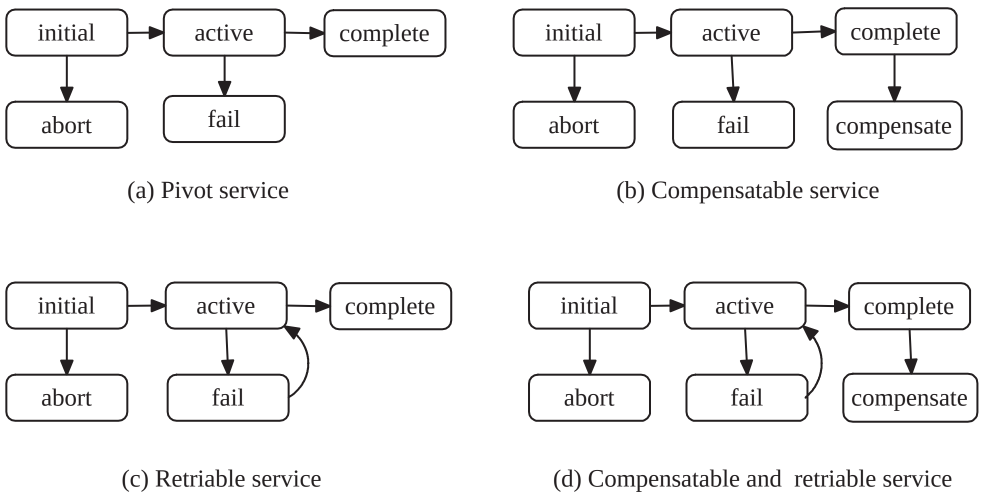


Figure 5.1: Service states describing their transactional properties

A service can combine behavioral properties. For example, a service could be compensatable and retriable at the same time, the combination of which leads to a new behavioral property of this service. In order to understand the transactional behavior of a service, we adopt the states model to describe its state transitions. A service has a minimal set of states (*initial*, *active*, *abort*, *fail* and *complete*). Figure 5.1 shows the state transitions of services according to their different transactional properties. The internal state transitions diagram of a pivot service is shown in Figure 5.1(a). Basically, the service enters the initial state when it starts, then its execution can be either active or abort. If it is active, the instance can normally continue its execution. The execution can complete successfully or it can fail. We add a new state *compensate* for a compensatable service, as shown in Figure 5.1(b). Figure 5.1(c) illustrates the states diagram of a retriable service; Figure 5.1(d) gives the state transitions of a compensatable and retriable service that combines both compensatable property and retriable property.

### 5.2.2 *Transactional Composite Services*

The transactional property of a composite service depends on its component services and the dependencies between them. Therefore, the transactional property of a composite service can be derived from those of component services conforming to $\{c,r,p\}$ with workflow patterns specifying the dependencies. Given a composite service *CS*, there exist many executions according to the same pattern, since different component services can be instantiated for each workflow task. At a given time, the state of *CS* is an *n*-tuple $(t_1,t_2,...,t_n)$, where $t_i$ is the state of service $s_i$. The states considered in this work are: initial, abort, active, fail, complete and compensate. We say that a composite service execution succeeds if it leads the service into a termination state, where the composite service terminates and its termination state falls in the set of all *Accepted Termination States* (*ATS*) [107] for which the designer of the service accepts the service termination. We use the concept of ATS as the notion of correction.

Different users may specify different ATS to reflect their transactional requirements. Our goal is to verify a web service composition according to the given ATS.

## 5.3 *Modeling*

In this section, we give a conversation-based transition model for component services and composite services respectively. In our composite service model, service choreography is based on component service actions and message exchanges between them. Each message exchange is associated with an action offered by a service and implies an exchange of information between the invoking service and the service providing the desired function. A linearization of messages is called a *conversation*. We formally model services and their choreography as follows [96][40][56].

**Definition 4.** ***Web service transition system***. *The model of a web service ws is a transition system, denoted as $T_{ws} = (\sum, S, \delta, s_0, F)$. In $T_{ws}$, $\sum = \sum_{in} \cup \sum_{out}$ is the alphabet of the transition system. Each element in $\sum$ is a tuple $(op, msg, dir)$, where op represents an operation, msg represents an input or output message of op and dir equals to $\longrightarrow$ if $msg \in \sum_{in}$, i.e., msg is an input message or $\longleftarrow$ if $msg \in \sum_{out}$, i.e., it is an output message. S is a finite set of states. $s_0 \in S$ is the initial state, and F is the set of final states. $\delta$ is the transition function $S \times \sum \longrightarrow S$.*

A state in the transition system represents the state of the interaction between the service and another service. The role of a web service in interaction with another web service can be *client*, *server* or *both*. If the initial state of the web service is only in the state of waiting for receiving messages, then it is a server; if its initial state is only in the state of sending messages, then it is a client; otherwise the web service have both roles. A transition is the result of the receipt of an input message or the sending of an output message of an operation.

Let us assume Figure 5.1(a) denotes a *Seller* service with the pivot transactional property. We give its transition diagram in Figure 5.2.

**Definition 5.** ***Composite service transition system***. *Given a composite service CWS consisting of services $s_1, s_2, ..., s_n$, the interaction of services $s_1, s_2, ..., s_n$ can be described by a non deterministic transition system $T_{CWS} = (S, A, T, s_0, s_f)$, where S is a finite set of choreography states, of which each state is a tuple of the form $((s_1, state_1), (s_2, state_2), ...,$ $(s_n, state_n))$, where in each tuple $state_i$ represents the state of service $s_i$. $s_0 \in S$ is the initial state and $s_f \in S$ is the set of final states. A is a sequence of messages. Each message*

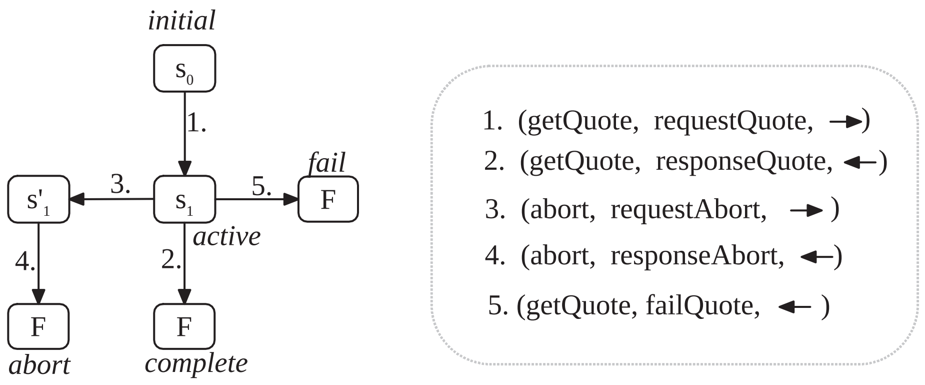

initial
s0
1.
fail
3.
5.
s'1
s1
F
active
4.
2.
F
F
abort
complete
1. (getQuote, requestQuote, →)
2. (getQuote, responseQuote, ←)
3. (abort, requestAbort, →)
4. (abort, responseAbort, ←)
5. (getQuote, failQuote, ←)

Figure 5.2: Transition of a *seller* service

*exchange is represented by a tuple* $(s_{out}, s_{in}, s_{op}, m)$*, where* $s_{out}$ *is the service sending the message,* $s_{in}$ *is the service receiving the message,* $s_{op}$ *is one operation of* $s_{in}$ *invoked by* $s_{out}$ *initiating the message exchange, and m is the message being exchanged. T is the transition function* $T : S \times A \rightarrow S$*. A transition* $(s, a, s') \in T$ *if there exists* $a = (s_{out}, s_{in}, s_{op}, m)$ *and the tuples* $(s_{out}, state_{out})$ *and* $(s_{in}, state_{in})$ *in state s are replaced by the tuples* $(s_{out}, state'_{out})$ *and* $(s_{in}, state'_{in})$ *in state* $s'$ *respectively because of the invocation of operation* $s_{op}$*.*

Note that the message exchanges in a composite service transition system is choreographed by workflow patterns. A workflow pattern can be seen as an abstract description of activation dependency between an invoking service and an invoked service. For example, the AND-join pattern shown in Figure 5.3 specifies that service *D* can not enter an *active* state until it receives the complete messages from both service *B* and service *C*.

The transition system for a composite service may have multiple termination states, which shows the different execution results of the composite service. Based on the definition of ATS, we have the following theorem.

**Theorem 1.** *A composite service is not valid if* $s_f \notin ATS$*, i.e., there exists some termination* *state $s_f$ that does not belong to ATS specified by a designer.*

*Proof.* Given a composite service *CS*, its termination state is a subset of $\{initial, abort, active, fail, complete, compensate\}$. There are only a limited number of subsets that are acceptable to users, which is $ATS$. So, if the termination of *CS* falls out of $ATS$, we say *CS* is invalid. □

## 5.4 Verification with Transition Systems

### 5.4.1 Objective and Overview

The problem is how to ensure a composite service can terminate in an accepted state required by a designer. By Theorem 1, we only need to verify if there is a final state in the transition system for the composite service that is not acceptable for the designer. Therefore, in order to carry out the verification process, the transition system model for each component service participating in the composition is needed.

The WSDL descriptions of web services specify their behavior. Firstly, we assume that an application analyst has already provided the transition system representing the behavior of each component service and the designer has selected a set of component services whose function match the sub-task requirements in the composition. Secondly, we ask the designer to compose a new service using a set of predefined operators (AND, OR, split or join) and a set of workflow patterns (sequence, synchronize, ... ), and then to specify the ATS for the composite service. Figure 5.3 shows a sample composite service defined using AND-split and AND-join patterns. Next, we check for correctness of the composition based on available transition systems. If a wrong $s_f$ can be reached by the set of services chosen by the designer, we ask the designer to select again by choosing new services with same

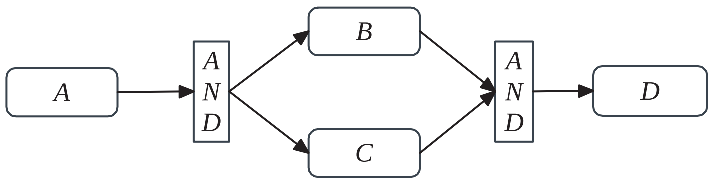


Figure 5.3: A composite service defined with AND-split and AND-join patterns.

function yet new behavior (transition system). Finally, If the selected services are consistent with the ATS specified, the composite service can be deployed and executed.

### *5.4.2 Verification Algorithm*

In this section, we present the algorithm for verification of transactional requirements (ATS) given a composite web service $CS = \{s_1, s_2, , s_i, , s_n\}$. We select a web service $s_i$ for each sub-task participating in $CS$. The Verification algorithm has as inputs $T_{s_1}$, $T_{s_2}$, ..., $T_{s_n}$, where $T_{s_i}$ is the transition system for service $s_i$. It constructs the transition system $T_{CS}$ and returns its final set of states $s_f$. If some of the states are not consistent with ATS, then the composition needs to be corrected.

The main idea of the algorithm is to dynamically generate all the possible states and transitions of $T_{CS}$ and store the current sates in a temporary set $state$. For reducing memory cost, we create the states and transitions only if they are reachable. Algorithm 5.1 outlines the computation procedure.

## 5.5 Verification with SPIN

As we have seen in the previous section, our algorithm enumerates all possible states of the transition systems of component services. With the number of states increases, the cost of

```
input  : $T_{s_0}, T_{s_1}, ..., T_{s_n}$, ATS
output: $Flag$: satisfying transactional requirements or not
1  $S_0 = \{T_{s_0}.s_0, T_{s_1}.s_0, ..., T_{s_i}.s_0, T_{s_n}.s_0\}$;
2  $State = \{S_0\}$;
3  $Flag = True$;
4  repeat
5      if $\exists (op, msg, dir)$ such that $T_{s_i}.s_{cur} \times (op_i, msg_i, dir_i) = T_{s_i}.s'_{cur}$ AND
       $T_{s_j}.s_{cur} \times (op_j, msg_j, dir_j) = T_{s_j}.s'_{cur}$ AND $op_i = op_j$ and $msg_i = msg_i$
       AND $dir_i \neq dir_j$ then
6          obtain $State'_{cur}$ by replace $T_{s_i}.s_{cur}$ and $T_{s_j}.s_{cur}$ with $T_{s_i}.s'_{cur}$ and
           $T_{s_j}.s'_{cur}$, respectively;
7          set $State = State \bigcup State'_{cur}$;
8      end
9  until there is no change ;
10 if $\exists E \in State$ such that all $T_{s_i} \in E$ having $F$ states AND $E \notin ATS$ then
11     $Flag = False$;
12 end
13 Return $Flag$;
```

**Algorithm 5.1**: Verification of transactional requirements

| BPEL struct | FSP notation |
|---|---|
| <sequence> <receive name=”act1”> </receive> <receive name=”act2”> </receive> </sequence> | ACT1 = (action1 -> END). ACT2 = (action2 -> END). SEQUENCE = ACT1;ACT2;END. |
| <switch name= "MPS"> <case condition= "cond1" = “true”>…… [act1] <otherwise>…[act2] </switch> | SWITCH = if cond1-true then ACT1;END else if cond2-true then ACT2;END else END. |
| <while condition = "cond1" = "true"> <sequence>... </sequence> </while> | WHILE = If condition-true then ACT1;WHILE else END. |
| <pick name ="pick1"> <onMessage> <invoke ACT1>... <onAlarm> <invoke ACT2>... </pick> | PICK1 = ( event1 -> ACT1; END \| event2 -> ACT2; END). |
| <flow name="flow1"> <receive name="act1">... <receive name="act2">... </flow> | \|\|FLOW1 = (ACT1 \|\| ACT2). |

Figure 5.4: Translate BPEL to FSP

the algorithm will increase as well. In order to reduce cost, in this section we discuss the use of the SPIN model checker to verify the transactional properties of composite services. SPIN model checker verifies LTL properties of Promela specifications using an exhaustive state space search [64], so we need to convert the composite service represented with BPEL into Promela, then the ATS needs to be expressed in temporal logic properties.

We use the idea in [55] to do the translation of the control flow of BPEL to finite state machines expressed by Finite State Process (FSP), which can then easily be converted into the Promela specification consisting of a set of concurrent processes, one for each automaton. Figure 5.4 lists the translation from BPEL to FSP.

After we have obtained the Promela specification for the composite service to be verified, we can express the correct ATS states as temproal logic properties against which the verification process can be implemented by SPIN automatically.

## *5.6 Related works*

Transactional support for web services has been a hot area. Many works have been done recently to address this issue. Industrial standards, such as Business Transaction Protocol (BTP) [3], WS-Coordination [8], WS-Transaction [20], WS-Atomic Transaction, WS-BusinessActivity, have defined standardized ways for transaction processing of distributed web services. Specifically, BTP defines how to track and manage such complex, multi-step B2B transactions over the Internet using XML messages; WS-Coordination describes an extensible framework for providing protocols that coordinate the actions of distributed applications; WS-Transaction builds upon WS-Coordination framework and defines the protocols for centralized and peer-to-peer transactions; WS-Atomic Transaction defines completion, volatile two-phase commit, and durable two-phase commit protocols; WS-

BusinessActivity specification provides the definition of a business activity coordination type used to coordinate activities that apply business logic to handle business exceptions; WS-TXM [9] is designed to ensure a composite application always either reaches successful completion or transits to predictable, known state if one or more of the individual web services fail in the composite application. All these standards provide support for compensation mechanism for long running business activities.

Some efforts aim at analyzing transactional properties of composite web services. In [80] authors investigate the transactional properties of workflow constructs. The concept of connection point is also introduced to derive the transactional properties of composite web services. [59] propose a model, in which web services are selected to satisfy both users' QoS requirements and transactional requirements at some risk level. Other efforts focus on how to assist users to design composite services with required transactional properties. For example, In [34][33], authors propose a transactional approach to ensure the failure atomicity of a composite service. The Accepted Termination States (ATS) property is used to express the required failure atomicity. Then, a set of transactional rules are derived to assist designers to compose a valid composite service regarding to the specified ATS. Authors of [87] propose a new process to automate the design of transactional composite web services. The resulting composite web service is compliant with the consistency requirements expressed by business application designers and its execution can easily be coordinated using the coordination rules. A framework to ensure consistent termination of composite web services with temporal constraints is designed in [81]. Some works focus on exception handling in web services composition, such as [145][46][32][53].

There are also some works discussing the modeling of behavior of web services. In [56][57], authors model the interactions of composite web services as conversations, the global sequence of messages exchanged by the web services. The guarded automata model is used as an intermediate representation for composite web services. In [96], authors verify whether a choreography can be implemented by a set of web services based on their access control policies and the disclosure policies. Web services are modeled as transition systems and credential disclosure policies are represented as directed graphs. Our work tries to combine the ideas of conversation model and ATS to ensure the consistent termination states of composite web services.

## *5.7 Conclusions*

In this work, we have proposed a novel approach to verify at design time whether a service composition can be implemented by a set of web services satisfying transactional requirements specified by a user. We defined transactional properties for component web services and analyzed the dependencies between them. Based on automata theory, we modeled both component web services and composite services as transition systems, and used the Accepted Termination States to describe the transactional properties of a composite service. Then, we presentd a novel algorithm to verify all possible state transitions of a composite service to ensure consistent termination states according to the specified ATS by the user. In addition, for the cause of efficiency we used Temporal Logic to describe the transactional properties of a composite service, and then we employed the *SPIN* model checker to carry out automated verification of temporal logic properties on the composite service.

As part of on-going work, we plan to improve the efficiency of our verification algorithm. We are also interested in investigating how to use other model checking tools, such as CPN-tools to verify transactional properties of composite web services.

# Chapter 6
# CONCLUSIONS

In this thesis, a web service composition framework has been presented for efficient web services discovery and composition in service oriented environments. We have studied three major problems in the framework: (1) efficient web services discovery, (2) dynamic web service selection with QoS (*Quality of Service*) constraints, and (3) verification of web service selection for transactional composition.

In this chapter, we summarize the contributions of this thesis in Section 6.1, and then in Section 6.2 we discuss some future research issues and directions.

## *6.1 Contributions*

A Web Service is a set of related functionalities that can be loosely coupled with other services programmatically through the Web. With the rapid development and popularity of Internet and e-commerce, business organizations are investigating ways to expose their current software components into web services so as to make use of distributed service computing resources. As a growing number of web services are available on the Web, how to discover appropriate web services and compose them has become an ever more important problem to many business organizations. Apart from functional specification required by users during service discovery process, some non-functional requirements such as QoS (*Quality of Service*) and transactional properties are also major concerns, because users need to select and compose web services not only according to their functional requirements but also to their transactional properties and QoS characteristics defined using a quality model. They are particularly important in web service composition.

In this thesis, we have proposed a service composition framework assisting users to discover web services and compose them according to QoS and transactional constraints. The main contributions of this thesis are stated below.

1. We have proposed a novel IR-Style mechanism for discovering and ranking web services automatically, given a textual description of desired services. We designed novel algorithms for supporting web service operations matching. The key part of our algorithms is a schema tree matching algorithm, which employs a new cost model to compute tree edit distances. Our new schema tree matching algorithm can not only catch structures, but also the semantic information of schemas. Based on service operations matching, we used the agglomeration algorithm to cluster similar web service operations. Also, an approach to identify associations between web-service operations is presented. This approach uses the concept of *attribute closure* to obtain sets of operations. Each set is composed of associated web service operations. Moreover, we introduced the notion of *preference degree* for web services and then we defined *service relevance* and *service importance* respectively as two desired properties for measuring the preference degree. Novel algorithms for computing the relevance and importance degree of services were developed. In addition, we defined *service connectivity*, a novel metric to evaluate the importance of services. Various experiments have been done to evaluate our approach. Initial results show the proposed IR-style search strategy is efficient and practical.

2. We have proposed a novel personalized QoS model for performing flexible and adapt-

able service selection. Our model can solve the problem of how to aggregate and leverage individual service's QoS information to derive the optimal QoS of the composite service, given personalized user QoS constraints. The key idea of the model is to use skyline technique to relax users' QoS constraints and try to find the most possible services which meet users' requirements. The problem of QoS computation and policing is converted into the skyline deduction problem. Furthermore, based on the proposed QoS framework, we developed various algorithms for making service selection on individual and composite services, respectively. We also introduced a top-*k* ranking strategy to reflect a user's personalized requirements. In addition, we presented experimental results of a thorough evaluation. Experimental evaluation shows the proposed QoS model is efficient and practical.

3. We have proposed a strategy for verification of web service selection for transactional composition. Our approach is able to verify at design time whether a service composition can be implemented by a set of web services satisfying transactional requirements specified by a user. Service selection passing the verification can ensure the correct composition and reliable execution of a composite service according to user transactional requirements. Firstly, we defined a transactional model for component web services. The transactional properties of component web services together with the dependencies between component services, which are defined by the workflow patterns that specify how services are combined together and how the behavior of a service interacts with other services, contribute to the transactional properties of composite services. Secondly, based on automata theory, we modeled both component web services and composite services as transition systems, and used the *Accepted*

*Termination States* to describe the transactional properties of a composite service. A verification algorithm was developed to carry out automated verification of Accepted Termination States on composite services. In addition, for the cause of efficiency we use *Temporal Logic* to describe the transactional properties of a composite service, and then we employ the *SPIN* model checker to carry out automated verification of temporal logic properties on the composite service.

## *6.2 Future Research Issues*

In the following, several research issues of our interests for future work are identified.

- Semantic web services discovery. In this thesis, our focus has been on discovering desired web services according to users' textual description. Currently, there is a trend to annotate web services with semantic information, such as OWL-S standard, etc., in order to enable service discovery to be more accurate and intelligent. Usually, web services discovery processes use the same domain ontologies. Therefore, it is possible to build a knowledge base based on web service ontologies, construct formal description of web services and user interest model, and then match services using their ontological classifications. However, building a reasonable and efficient service oriented knowledge base poses a big challenge.

- Monitoring the QoS for web services. As web services operate in dynamic environments, service providers may provide lower QoS than expected because of hardware failure, network delay, etc. So, sometimes the service selection result according to user QoS requirements may not be reliable or accurate. it is necessary to build a QoS monitoring architecture which is based on not only detecting and measuring the quality of services actively and automatically so as to make the available QoS values up to date, but also collecting user feedbacks actively.

- Exception handling and recovery management in web service composition. The loosely coupled, distributed and dynamic nature of web services makes it possible for exceptions. For example, new services may become available at any time; existing services may become obsolete or temporarily unavailable, etc. Therefore, it is crucial to provide support for exception handling and recovery management policy in service composition to ensure reliable and consistent execution of services.

# BIBLIOGRAPHY

[1] Business process execution language for web services, version 1.0, 2000. http://www-106.ibm.com/developerworks/library/ws-bpel/.

[2] Business process management. http://en.wikipedia.org/wiki/business_process_management.

[3] Business transaction protocol (btp) specification. http://www.oasis-open.org/committees/business-transaction/.

[4] Defining web services. http://www.perfectxml.com/xanalysis/tsg/tsg_definingwebservices.pdf.

[5] http://searchsoa.techtarget.com.

[6] Ieee services computing community. https://www.ieeecommunities.org/services.

[7] Matrix representation of petri nets. http://www.techfak.uni-bielefeld.de/∼mchen/biopnml/intro/mrpn.html.

[8] Oasis web services coordination specification. http://docs.oasis-open.org/ws-tx/wscoor/2006/06.

[9] Oasis web services transaction (ws-tx). http://www.oasis-open.org/committees/ws-tx.

[10] Overview of w3c technologies. http://www.w3.org/consortium/offices/presentations/overview.

[11] Owl-s: Semantic markup for web services. http://www.w3.org/submission/owl-s/.

[12] Petri nets. http://www.cse.fau.edu/∼maria/courses/cen4010-se/c10/10-7.html.

[13] Service computing and business process management. http://www.eii.edu.au/taskforce0506/bpm.

[14] Skyline website developer manual 1.0.
http://www.ccs.neu.edu/home/jarodwen/skylinedoc/html/.

[15] Web service choreography interface (wsci) 1.0.
http://www.w3.org/tr/wsci/.

[16] Web service level agreement (wsla) language specification version 1.0.
http://www.research.ibm.com/wsla/.

[17] Web services choreography description language version 1.0.
http://www.w3.org/tr/2004/wd-ws-cdl-10-20041217/.

[18] Web services conversation language (wscl) 1.0, 2002.
http://www.w3.org/tr/wscl10/.

[19] Web services flow language (wsfl 1.0).
http://www-4.ibm.com/software/solutions/webservices/pdf/wsfl.pdf.

[20] Web services transactions specifications.
http://www.ibm.com/developerworks/library/specification/ws-tx/.

[21] Workflow/business process management (bpm) service pattern.
http://enterprisearchitecture.nih.gov/archlib/at/ta/workflowservicepattern.htm.

[22] Xlang: Web services for business process design, 2002.
http://www.gotdotnet.com/team/xml wsspecs/xlang-c/default.htm.

[23] SOAP Version 1.2. http://www.w3.org/tr/soap/.

[24] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, Santiago de Chile, Chile, 1994.

[25] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[26] Eyhab Al-Masri and Qusay H. Mahmoud. Qos-based discovery and ranking of web services. In *ICCCN*, pages 529–534, 2007.

[27] Eyhab Al-Masri and Qusay H. Mahmoud. Relevancy ranking of web services. In *SMC*, pages 783–788, 2007.

[28] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *WWW*, pages 795–804, 2008.

[29] Paul C. Attie, Munindar P. Singh, Amit P. Sheth, and Marek Rusinkiewicz. Specifying and enforcing intertask dependencies. In *VLDB*, pages 134–145, 1993.

[30] Boualem Benatallah and Fabio Casati. Guest editorial. *Distributed and Parallel Databases*, 12(2/3):115–116, 2002.

[31] Boualem Benatallah, Marlon Dumas, and Zakaria Maamar. Definition and execution of composite web services: The self-serv project. *IEEE Data Eng. Bull.*, 25(4):47–52, 2002.

[32] Sami Bhiri, Walid Gaaloul, and Claude Godart. Discovering and improving recovery mechanisms of compositeweb services. In *ICWS*, pages 99–110, 2006.

[33] Sami Bhiri, Claude Godart, and Olivier Perrin. Transactional patterns for reliable web services compositions. In *Proceedings of International Conference on Web Engineering (ICWE)*, pages 137–144, 2006.

[34] Sami Bhiri, Olivier Perrin, and Claude Godart. Ensuring required failure atomicity of composite web services. In *WWW*, pages 138–147, 2005.

[35] BindingPoint. http://www.bindingpoint.com.

[36] D. Booth, H. Haas, F. McCab, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. http://www.w3.org/tr/ws-arch/. 2004.

[37] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[38] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[39] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[40] Tevfik Bultan, Jianwen Su, and Xiang Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1):18–25, 2006.

[41] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Francesco Perfetto, and Maria Luisa Villani. Service composition (re)binding driven by application-specific qos. In *ICSOC*, pages 141–152, 2006.

[42] Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1(3):281–308, 2004.

[43] James Caverlee, Ling Liu, and Daniel Rocco. Discovering and ranking web services with basil: a personalized approach with biased focus. In *ICSOC*, pages 153–162, 2004.

[44] D. Chappel and T. Jewell. *Java Web Services*. OReilly, 2002.

[45] The Workflow Management Coalition. ”terminology and glossary,wfmc-tc-1011, http://www.wfmc.org/standards/docs/tc-1011_term_glossary_v3.pdf, 1999.

[46] Francisco Curbera, Rania Khalaf, Frank Leymann, and Sanjiva Weerawarana. Exception handling in the bpel4ws language. In *Business Process Management*, pages 276–290, 2003.

[47] Xinguo Deng, Ziyu Lin, Weiqing Cheng, Ruliang Xiao, Lina Fang, and Ling Li. Modeling web service choreography and orchestration with colored petri nets. In *SNPD ’07: Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 838–843, Washington, DC, USA, 2007. IEEE Computer Society.

[48] IBM Developerworks. www.ibm.com/developerworks/webservices.

[49] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for web services. In *VLDB*, pages 372–383, 2004.

[50] Prashant Doshi, Richard Goodwin, Rama Akkiraju, and Kunal Verma. Dynamic workflow composition: Using markov decision processes. *Int. J. Web Service Res.*, 2(1):1–17, 2005.

[51] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, pages 1–30, August 2008.

[52] E. Allen Emerson. Temporal and modal logic. pages 995–1072, 1990.

[53] Xiaocong Fan, Karthikeyan Umapathy, John Yen, and Sandeep Purao. Team-based agents for proactive failure handling in dynamic composition of web services. In *ICWS*, pages 782–, 2004.

[54] Charles Foster. Xml databases and xml information exchange. http://www.cfoster.net/articles/xmldb-business-case. 2008.

[55] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *ASE '03*, pages 152–161. IEEE, 2003.

[56] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *WWW*, pages 621–630, 2004.

[57] Xiang Fu, Tevfik Bultan, and Jianwen Su. Wsat: A tool for formal analysis of web services. In *CAV*, pages 510–514, 2004.

[58] Aiqiang Gao, Dongqing Yang, Shiwei Tang, and Ming Zhang. Web service composition using markov decision processes. In *WAIM*, pages 308–319, 2005.

[59] Joyce El Haddad, Maude Manouvrier, Guillermo Ramirez, and Marta Rukoz. Qos-driven selection of web services for transactional composition. *Web Services, IEEE International Conference on*, 0:653–660, 2008.

[60] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[61] Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. Hypergraph based clustering in high-dimensional data sets: A summary of results. *IEEE Data Eng. Bull.*, 21(1):15–22, 1998.

[62] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[63] Qiang He, Jun Yan, Yun Yang, Ryszard Kowalczyk, and Hai Jin. Chord4s: A p2p-based decentralised service discovery approach. In *IEEE SCC (1)*, pages 221–228, 2008.

[64] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, Boston, Massachusetts, 2003.

[65] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[66] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[67] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[68] http://www.cs.umass.edu/∼mccallum. rainbow.

[69] Zhiyong Huang, Christian S. Jensen, Hua Lu, and Beng Chin Ooi. Skyline queries against mobile lightweight devices in manets. In *ICDE*, page 66, 2006.

[70] IBM. Web services and uddi.
http://www.ibm.com/services/uddi. 2001.

[71] Danish Irfan, Xiaofei Xu, Shengchun Deng, and Zengyou He. K-means clustering - a survey. In *IKE*, pages 190–196, 2007.

[72] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. Qos aggregation for web service composition using workflow patterns. In *EDOC*, pages 149–159, 2004.

[73] Kurt Jensen, Lars Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3):213–254, June 2007.

[74] Radu Jurca, Boi Faltings, and Walter Binder. Reliable qos monitoring based on client feedback. In *WWW*, pages 1003–1012, 2007.

[75] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, New York, 1990.

[76] Rania Khalaf. Business process with bpel4ws: Learning bpel4ws.
http://www.ibm.com/developerworks/webservices/. 2002.

[77] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[78] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210, 2006.

[79] H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[80] Li Li, Chengfei Liu, and Junhu Wang. Deriving transactional properties of compositeweb services. *IEEE International Conference on Web Services ( ICWS'07)*, 0:631–638, 2007.

[81] An Liu and Qing Li. Ensuring consistent termination of composite web services. In *Proceedings of SIGMOD2007 Ph.D. Workshop on Innovative Database Research 2007 (IDAR 2007)*, pages 15–20, 2007.

[82] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, pages 66–73, 2004.

[83] Jiří Matoušek. Computing dominances inen (short communication). *Inf. Process. Lett.*, 38(5):277–278, 1991.

[84] Microsoft. Microsoft advanced uddi search. http://uddi.microsoft.com/search.aspx. 2001.

[85] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[86] Document Object Model. http://en.wikipedia.org/wiki/dom.

[87] Frederic Montagut, Refik Molva, and Silvan Tecumseh Golega. Automating the composition of transactional web services. *Int. J. Web Service Res.*, 5(1):24–41, 2008.

[88] MSDN. http://msdn.microsoft.com.

[89] Saikat Mukherjee, Hasan Davulcu, Michael Kifer, Pinar Senkul, and Guizhen Yang. Logic based approaches to workflow modeling and verification. In *Logics for Emerging Applications of Databases*, pages 167–202, 2003.

[90] Petri net. Coloured petri nets. http://www.daimi.au.dk/cpnets/intro/.

[91] Petri net. http://en.wikipedia.org/wiki/petri_net.

[92] Hamid Reza Motahari Nezhad. Discovery and adaptation of process views. *PhD thesis*, The University of New South Wales.

[93] Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in xml documents. In *WebDB*, pages 61–66, 2002.

[94] OASIS. Introduction to uddi: Important features and functional concepts. http://www.uddi.org/pubs/uddi_v3.htm. October 2004.

[95] Seog-Chan Oh, Dongwon Lee, and Soundar R. T. Kumara. A comparative illustration of ai planning-based web services composition. *SIGecom Exch.*, 5(5):1–10, 2006.

[96] Federica Paci, Mourad Ouzzani, and Massimo Mecella. Verification of access control requirements in web services choreography. In *IEEE SCC (1)*, pages 5–12, 2008.

[97] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD Conference*, pages 467–478, 2003.

[98] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[99] Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008.

[100] Shankar R. Ponnekanti and Armando Fox. Sword: A developer toolkit for web service composition. In *Proceedings of the 11th International WWW Conference (WWW2002)*, Honolulu, HI, USA, 2002.

[101] K. Pu, V. Hristidis, and N. Koudas. Syntactic rule based approach toweb service composition. In *ICDE*, page 31, 2006.

[102] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994.

[103] Lirong Qiu, Fen Lin, Changlin Wan, and Zhongzhi Shi. Semantic web services composition using ai planning of description logics. *Asia-Pacific Conference on Services Computing. 2006 IEEE*, 0:340–347, 2006.

[104] SOAP Reference. http://msdn.microsoft.com/en-us/library/aa909268.aspx.

[105] D. D. C. Reis, P. B. Golgher, A. S. d. Silva, and A. H. F. Laender. Automatic web news extraction using tree edit distance. In *WWW*, pages 502–511, 2004.

[106] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology (WSMO). WSMO Final Draft 10, 2005.

[107] Marek Rusinkiewicz and Amit Sheth. Specification and execution of transactional workflows. In *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 592–620. ACM Press, 1995.

[108] A. Sajjanhar, J. Hou, and Y. Zhang. Algorithm for web services matching. In *APWeb*, volume 3007, pages 665–670, 2004.

[109] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun.ACM*, 18(11):613–620, 1975.

[110] Schema. http://www.w3.org/xml/schema.

[111] XML Schema. http://en.wikipedia.org/wiki/xml_schema.

[112] Web service usage scenario. http://www.w3.org/2002/04/17-ws-usecase.html.

[113] Z. Shen and J. Su. Web service discovery based on behavior signatures. In *SCC*, volume 1, pages 279–286 vol.1, 2005.

[114] UDDI Specification. http://www.uddi.org/pubs/uddi_v3.htm.

[115] Biplav Srivastava. Web service composition - current solutions and open problems. In *In: ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

[116] Simon St.Laurent. Why xml?. http://www.simonstl.com/articles/whyxml.htm. 1998.

[117] Katia P. Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. Larks: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.

[118] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[119] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 301–310, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[120] UDDI Registry tModels. http://www.uddi.org/taxonomies/uddi_registry_tmodels.htm.

[121] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[122] w3school tutorials. http://www.w3schools.com/schema.

[123] w3school tutorials. http://www.w3schools.com/xml.

[124] w3school XML DOM Tutorial. http://www.w3schools.com/dom.

[125] Hongbing Wang, Junjie Xu, and Peicheng Li. Incomplete preference-driven web service selection. In *IEEE SCC (1)*, pages 75–82, 2008.

[126] Jason Tsong-Li Wang, Bruce A. Shapiro, Dennis Shasha, Kaizhong Zhang, and Kathleen M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):889–895, 1998.

[127] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A qos-aware selection model for semantic web services. In *ICSOC*, pages 390–401, 2006.

[128] Xiaoling Wang, Sheng Huang, and Aoying Zhou. Qos-aware composite services retrieval. *J. Comput. Sci. Technol.*, 21(4):547–558, 2006.

[129] Y. Wang and E. Stroulia. Flexible interface matching for web-service discovery. In *WISE*, 2003.

[130] WebServiceList. http://www.webservicelist.com.

[131] XML Wikipedia. http://en.wikipedia.org/wiki/xml.

[132] Web Service Definition Language (WSDL). www.w3.org/tr/wsdl.

[133] T. Xie, C. Sha, X. Wang, and A. Zhou. Approximate top-k structural similarity search over xml documents. In *APWeb*, volume 3841, pages 319–330, 2006.

[134] XMethod. http://www.xmethods.org.

[135] Guandong Xu. Web mining techniques for recommendation and personalization. *PhD thesis*, Victoria University.

[136] Jian Yang and Prof Mike P. Papazoglou. The web-services phenomenon: Concepts technologies trends and research directions. In *CAiSE'02 Tutorial*.

[137] Wuu Yang. Identifying syntactic differences between two programs. *Softw., Pract. Exper.*, 21(7):739–755, 1991.

[138] YanPing Yang, QingPing Tan, and Yong Xiao. Verifying web services composition based on hierarchical colored petri nets. In *IHIS '05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 47–54, New York, NY, USA, 2005. ACM.

[139] Cong Yu and H. V. Jagadish. Schema summarization. In *VLDB*, pages 319–330, 2006.

[140] Tao Yu and Kwei-Jay Lin. Service selection algorithms for web services with end-to-end qos constraints. *Inf. Syst. E-Business Management*, 3(2):103–126, 2005.

[141] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1), 2007.

[142] Amy Moormann Zaremski and Jeannette M. Wing. Specification Matching of Software Components. *ACM Trans. Softw. Eng. Methodol.*, 6(4):333–369, 1997.

[143] Liangzhao Zeng. Dynamic web services composition. *PhD thesis*, The University of New South Wales.

[144] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW*, pages 411–421, 2003.

[145] Liangzhao Zeng, Hui Lei, Jun-Jang Jeng, Jen-Yao Chung, and Boualem Benatallah. Policy-driven exception-management for composite web services. In *CEC*, pages 355–363, 2005.

[146] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *WWW*, pages 76–85, 2005.

[147] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J.Comput.*, 18(6):1245–1262, 1989.