Ontology-Independent and QOS-enabled Dynamic Composition of Web Services in Business Domains

by

Rui Ding

A Thesis Submitted to Saint Mary's University, Halifax, Nova Scotia,
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Applied Science

April 8, 2011, Halifax, Nova Scotia

Approved:	Dr. Dawn Jutla Supervisor Department of Finance, Computing Information Systems & Management Science
Approved:	Dr. Nur Zincir-Heywood External Examiner Faculty of Computer Science Dalhousie University
Approved:	Dr. Stavros Konstantinidis Supervisory Committee Member Department of Math and Computing Science
Approved:	Dr. Michael Zhang Supervisory Committee Member Department of Finance, Computing Information Systems & Management Science
Approved:	Dr. Muhong Wang Program Representative
Approved:	Dr. Jason Clyburne Graduate Studies Representative
Date:	April 23, 2011

Your file Votre référence
ISBN: 978-0-494-75793-2
Our file Notre référence
ISBN: 978-0-494-75793-2

Table of Contents

List of Tables

List of Figures

Acknowledgements

To my supervisor Dr. Dawn Jutla

This thesis would not have been possible without your encouragement, guidance, and invaluable help

To my Wife Xiao

Your patience and support sailed me through it all.

To my beloved parents Jiazheng & Ligu

Your unconditional love and encouragement were my guiding star.

Nothing would ever suffice to repay your endless sacrifices.

To my Daughter Lorna & my Son Bryan

Your lovely smiles are the origin of my strengths and courage.

Abstract

Ontology-Independent and QOS-enabled Dynamic Composition of Web Services in Business Domains

By Rui Ding

Abstract: This thesis proposes a novel and high-performance ontology-independent approach and methods for Quality of Services (QoS)-enabled dynamic web services discovery and composition. One proposed method uses Google distance for calculating semantic similarities instead of using the state-of-the-art ontological-based approaches in the semantic matching stage. A further new method is architected for the QoS operational matching stage of web services discovery. Moreover, the thesis proposes a hybrid approach to dynamic web services composition, called FOIQOS, consisting of using a prescriptive system for web services discovery and composition. Another problem the thesis addresses is the absence of comparisons of existing QoS-enabled composition approaches in the literature. To compare the new methods proposed in the thesis, FOIQOS and three other approaches for QOS-enabled dynamic web services composition were implemented. Experimental results show that the proposed FOIQOS approach significantly outperforms its ontology-based and heuristic-based method counterparts, in terms of both increased accuracy and reduced overhead.

April 23, 2011

Chapter 1

Introduction

The Quality of Service (QoS)-based services computing environment is rich and complex with theoretical proposals and implementation workarounds. Approved standards do not yet exist for QoS-enabled web service composition although many methods have been proposed in the past decade (e.g. Ponnekanti and Fox, 2002, Cardoso and Sheth, 2003, Wu et al, 2003, Liu et al, 2005, Karunamurthy et al, 2006, Thissen and Wesnarat, 2006, Fang et al, 2009). Some methods (e.g. Cardoso and Sheth, 2003, Wang et al, 2006, Ye and Zhang, 2006) can use or rely on ontological reasoning, for identifying service concepts and their properties, within service discovery which would be useful in domains where ontologies exist. However, ontologies are absent or poorly maintained in many domains, including business and public policy domains. Automated reasoning over a badly maintained ontology is not useful, creates new problems, and certainly incurs overhead in terms of delay.

Hence for the many situations that lack a well-defined ontology, this thesis proposes a *F*lexible *O*ntology-*I*ndependent and *QOS*-enabled (*FOIQOS*) approach, for automatically discovering and selecting web services for composition that incorporate QoS parameters to meet predefined application-level QoS objectives.

Moreover, trade-offs among methods for QoS-enabled dynamic web service compositions are not readily understood as direct scientific comparisons of these methods are absent from the literature. This thesis fills that gap.

The main purpose of this thesis is to explore an ontology-independent approach for

dynamically composing web services that incorporate QoS parameters to meet predefined QoS objectives. Furthermore, the thesis provides direct comparisons of the author's approach and other approaches for dynamic web services composition.

1. 1 Research Objectives

(1) Propose an ontology-independent approach to automatically discover web services which are then dynamically composed to meet application-defined QoS objectives;

(2) Compare the thesis's methods with other types of QoS -enabled web service composition methods to evaluate their relative performance and understand their tradeoffs.

1.2 Thesis Organization

The thesis organization is as follows. Chapter 2 introduces the background literature. Chapter 3 presents a design of a proposed ontology-independent approach for web service composition, reviews methods in the approaches selected for comparison, and proposes relevant methods. Chapter 4 describes the implementation environment for the proposed and comparative approaches. Chapter 5 presents the results of the proposed method and its peers. The final chapter offers a summary and conclusions.

Chapter 2

Literature Review

This chapter provides relevant background literature spanning web services research. First, the concepts of web services and semantic web services are introduced. Then principles of service composition, available tools, and methods for dynamic service composition are reviewed.

2.1 Semantic Web, Ontology, Web Service and Owl-S

2.1.1 Semantic Web

The Semantic Web is a vision for the future of the Web, in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web.

"What is the Semantic Web?" There is no clear definition. Tim Berners-Lee gave the description as following: "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [Technology Investigation Center, 2003]. From the above description, we can obtain the following meaning: the Semantic Web is the next generation of the World Wide Web, which can be understood and automatically processed by machines.

After learning the origin and development of the Semantic Web, we realize that AI (Artificial Intelligence) integrated with Web technologies resulted in the Semantic Web. The fundamentals of the Semantic Web are the formalization and conceptualization of the

knowledge and the relevant ratiocination. It has a consanguineous relationship with Artificial Intelligence. So, most of the analyses of the Semantic Web are considered using AI technologies. The knowledge in the Semantic Web is a series of descriptions and modeling of the resources. Resource here is a comprehensive conception. A Resource is anything that can have a URI (Uniform Resource Identifier). It could be a web site, a web page, or even a part of a web page. It uses symbols and expressions to describe the resource, other resources related with it, and the relationship between them. Traditional knowledge-representation systems such as an AI system typically have been centralized and each has its own narrow and particular set of rules for making inferences about its data. In contrast, in the semantic web, knowledge and its representation may be provided by vast amounts of people or organizations through various manners [W3C, 2005]. Further, knowledge can be understood by various applications and reasoning under the instructions of certain logic rules.

Currently, Web content is formatted for human readers rather than programs. HTML is the predominant language to create web pages. A portion of a typical Web page of a physical therapist might look like this:

<h1>Agilitas Physiotherapy Centre</h1>

Welcome to the home page of the Agilitas Physiotherapy Centre.

Do you feel pain? Have you had an injury? Let our staff

Lisa Davenport, Kelly Townsend (our lovely secretary)

and Steve Matthews take care of your body and soul.

<h2>Consultation hours</h2>

Mon 11am - 7pm

Tue 11am - 7pm

Wed 3pm - 7pm

Thu 11am - 7pm

Fri 11am - 3pm<p>

But note that we do not offer consultation during the weeks of the

State Of Origin games. [Grigoris, 2004]

For people, the information is presented in a satisfactory way, but machines will have problems. Keyword-based searches will identify the words physiotherapy and consultation hours. An intelligent agent might even be able to identify the personnel of the center. But it will have trouble distinguishing therapists from the secretary, and even more trouble with finding the exact consultation hours (for which it would have to follow the link to the State of Origin games to find when they take place). The semantic web approach to solving these problems is not the development of super intelligent agents. Instead it proposes to solve the problem from the Web page side. In addition to containing formatting information aimed at producing a document for human readers, they could contain information about their content. In our example, there might be information such as

<company>

<treatmentOffered>Physiotherapy</treatmentOffered>

<companyName>Agilitas Physiotherapy Centre</companyName>

<staff>

<therapist>Lisa Davenport</therapist>

<therapist>Steve Matthews</therapist>

<secretary>Kelly Townsend</secretary>

</staff>

</company>

This XML representation is far more easily process-able by machines. It is a kind of metadata. The term metadata refers to such information: data about data. Metadata capture part of the meaning of data, thus the term semantic in Semantic Web.

Web Applications on the web need to communicate with each other. Most of the machine readable information passed between those applications is descriptions about the resources on the web. According to the descriptive level, web information can be partitioned into several ranks [Jinghua, 2005]. The lowest rank, rank 1, is the raw data in the real life; Web page source information is located at the rank 2 (see HTML example mentioned above); metadata or patterns of the information resource is in rank 3; Logic reasoning and rules proof is in the highest rank. We can see the lower the rank, the more detailed and concrete the data is, therefore it is more suitable for human to process. In contrast, the higher rank data is more abstract, thus it is better for machines to process automatically. The data on the World Wide Web is disorderly and unsystematic. The information content and the information representations are lumped together. It is difficult to make use of the data because data with different "ranks" are not treated discriminatingly. To avoid the same problem, a logical architecture is necessary to Semantic Web. Figure 2.1 shows the architecture for the Semantic Web given by Berners-Lee [2000].

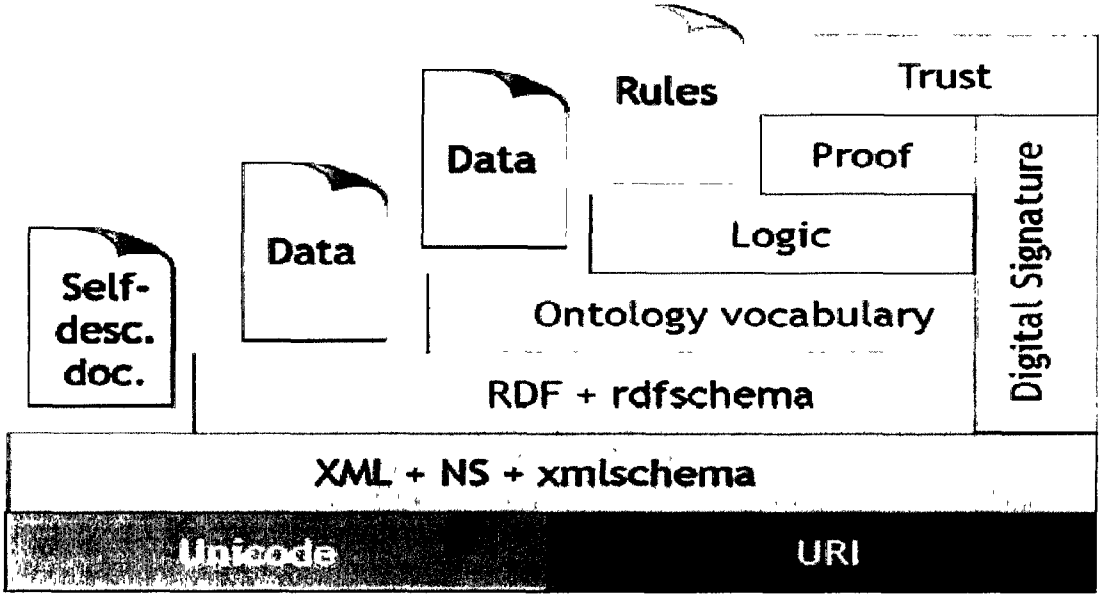


Figure 2.1 Architecture for Semantic Web [Berners-Lee, 2000]

2.1.2 URI and Unicode

According to Figure 2.1, the lowest layer of the architecture for the semantic web is the encoding layer. The semantic web adopts the URI (Uniform Resource Identifier) to identify resource and its properties. A URI can be further classified as a locator, a name, or both. The term URL (Uniform Resource Locator) refers to the subset of the URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of the URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable [Berners-Lee, 1998a]. In addition, since the final objective of Semantic Web is to build a global information network, all kinds of languages and character information need to be covered. So, it adopts Unicode as the solution for the character encoding question. URI and Unicode is the foundation of

the Semantic Web. It solves the problems of how to locate the resource on the web and how to encode all kinds of characters. In a word, the Unicode and URI layers ensure that we use international characters sets and provide means for identifying the objects in Semantic Web.

2.1.3 XML, NameSpace, and XML Schema

The second layer is the grammar layer. It is well-known that HTML had a tremendous contribution to the development of the Web. But with further development of the web, HTML is not sufficient any more. So, XML was used as the grammar of the Semantic Web. [Technology Investigation Center, 2003] HTML is used to display data, and it focuses on how data looks. While XML is designed to describe data and it focus on what data is. XML stands for Extensible Markup Language. It is a complement to HTML. With XML, data can be stored in separate XML files. Using this way people can concentrate on using HTML for data layout and display, and be sure that changes in the underlying data will not require any changes to HTML. Unlike the HTML, the tags and the structure of XML are not predefined; people can define their own tags. That's what "Extensible" stands for. The most useful advantage of the XML is that the data converted to XML can be exchanged and shared between incompatible systems [W3Schools, 2005]. Because XML tags can be freely defined by the author, there must be some unavoidable situations, in which the different tags have the same name. To solve this problem, W3C introduced the NameSpace mechanism. For example, a user can add an xmlns attribute to the <table> tag:

```
<f:table xmlns:f="http://www.w3schools.com/furniture">
```

It indicates the tag <table> is specified in the NameSpace represented by F:

http://www.w3schools.com/furniture

Hence, even if other persons define the <table> tag also, as long as their NameSpace is different, there will not be a conflict. In short, the XML layer with namespace and schema definitions makes sure we can integrate the Semantic Web definitions with the other XML-based standards. At this layer, XML gives the format for data exchanging, however, from a computational perspective, XML tags like <table> has no essential difference with HTML tag <H1>. A computer does not know what a table is. That means XML documents do not have any semantics.

2.1.4 RDF and RDF Schema

The third layer is metadata layer. XML provides the grammar for the web information encoding, while the Resource Description Framework (RDF), as its name implies, is a framework for describing and interchanging metadata. RDF is designed to represent information in the Web in a minimally constraining, and flexible way.

Resource Description Framework is built on the following rules [Grigoris, 2004] [W3C Recommendation, 2004a]:

1. Resource: which is anything that can have a URI; this includes web sites, web pages, or even a part of a web page, as well as individual elements of an XML document.

2. Property: which is a Resource that has a name and can be used as a property to describe attributes and characteristics of a Resource, for example Author or Title of this paper.
3. Property value: which is the value of a Property, for example "Report" is the value of Title property. A property value can be another resource.
4. Statement: RDF identifies things using Web identifiers (URIs), and describes resources with properties and property values. While the combination of a Resource, a Property, and a Property value forms a Statement. A Statement is the concrete descriptions of a Resource. Usually it can be described by using a <S,P,O> triple. Here S (Subject) denotes a particular thing (people, Web pages or whatever), P (Predicate) denotes the properties of that thing (such as "is a sister of," "is the homepage of"), and O (object) denotes the certain values of P regarding S (such as another person, another Web page).

In describing RDF statements, square brackets are used to denote RDF resources, containing a name for the resource. RDF properties are shown as labeled arrows from subject to object:

[SubjectName] --propertyName--> [ObjectName]

Any complex system can be simplified to an aggregation of <S, P, O> triples. For example:

<rdf:RDF>

<rdf:Description about="http://www.rfcs/rfc2396.html">

```
    <k:author> Tim Berners-Lee </k:author>

  </rdf:Description>

</rdf:RDF>
```

The above XML code makes the assertion: the author (Predicate) of http://www.rfcs/rfc2396.html (Subject) is Tim Berners-Lee (Object), which can be described as RDF statement:

[http://www.rfcs/rfc2396.html] –author--> [Tim Berners-Lee]

However, RDF just defines several basic modeling primitive. It doesn't provide the Property of its own. As shown in the above example, it didn't clarify the constraint that an author should be a person. Fortunately, RDF-Schema, an extension of RDF, further defines class hierarchies and property domains and data ranges. Simply speaking, RDF is domain-independent. RDF Schema provides a mechanism for describing specific domains. Classes in RDF Schema are much like classes in object oriented programming languages. This allows resources to be defined as instances of classes, and subclasses of classes. For example, if we wish to say that the class "lecturer" is a subclass of "academic staff member". How to describe it by using RDFS? Remember that RDF allows one to express any statement about any resource, and that anything that has a URI can be a resource. So, first, we define resource lecturer, academicStaffMember, and subClassOf, and then define subClassOf to be a property, and then write the triple (subClassOf, lecturer, academicStaffMember) [Grigoris, 2004]. All these steps are within the capabilities of RDF. So, an RDFS document (that is an RDF schema) is just an RDF document, and we

use the XML-based syntax of RDF.

RDF Schema is a primitive ontology language. It offers certain modeling primitives with fixed meaning [W3C Recommendation, 2005]. However, RDF plus RDFS is still not powerful enough for representing full semantics. There is a need for more powerful ontology languages that expand RDF Schema and allow the representations of more complex relationships between Web objects.

2.1.5 Ontology Vocabulary

Layer 4 is the glossary layer. The extension of RDF Schema, Ontology, is layer 4 of the Semantic Web Architecture. RDF Schema can define class, subclass relations, property, subproperty relations, and domain and range restrictions. So, in a sense, RDF Schema is a kind of simple Ontology language. However the expressiveness of RDF and RDF Schema is very limited: RDF is roughly limited to binary ground predicates, and RDF Schema is limited to a subclass hierarchy and a property hierarchy, with domain and range definitions of these properties. [W3C Recommendation, 2005] While the number of characteristic use-cases for the Semantic Web identified by the Web Ontology Working Group of W3C requires much more expressiveness than RDF and RDF Schema offer. Therefore we need an ontology layer on top of RDF/RDFS.

The most famous and frequently referenced definition about ontology is:" ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of

discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory." [Gruber, 1993]

N. Guarino and P. Giaretta (1995) gave the similar definition: "an ontology is an explicit, partial account of a conceptualization/ the intended models of a logical language."

Fensel analyzed the above definition and summarized it to four words: [Fensel, 2001]

- 'conceptualization': an abstract model of a phenomenon,
- 'formal': a precise mathematical description,
- 'explicit': the precision of concepts and their relationships clearly defined,
- 'shared': the existence of an agreement between ontology users

The Ontology layer is on top of the RDFS primitive class-property descriptions. It supports the evolution of vocabularies as it can define relations between the different concepts. Ontology provides a bridge to exchange semantic information and share the concept among different intelligent entities. It is the pivot in the layers of the semantic web. Furthermore, it can use Ontology languages, such as OIL (Ontology Inference Language), DAML (DARPA Agent Markup Language), KIF (Knowledge Interchange Format), SHOE (Simple HTML Ontology Extensions), XOL (Ontology Exchange

Language), and OWL (Web Ontology Language), to write explicit, formal conceptualizations of domain models.

2.1.6 Logic, Proof and Trust

By using RDF/RDFS and Ontology languages, we can provide the descriptions to Web resources. But these descriptions are not enough, as web applications based on semantics need to reason from these descriptive knowledge based on some rules. This kind of reasoning capability is provided by logic. Logic has a well-understood formal semantics, and it can provide a high-level language in which knowledge can be expressed in a transparent way. The aim of the logic layer is to provide a method to describe the rules, [Berners-Lee, 1998b] in such a way so that rules can be exchanged across different applications. DLML (Description Logic Markup Language) is a language to express rules. It encapsulates the connections of description logics through the DTD, and is able to embed formal knowledge in description logic in documents. [DLML, 2003]

The Proof layer involves the actual deductive process as well as the representation of proofs in Web languages (from lower levels) and proof validation. Finally, the Trust layer will emerge through the use of digital signatures and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and consumer bodies. Sometimes "Web of Trust" is used to indicate that trust will be organized in the same distributed and chaotic way as the WWW itself. Being located at the top of the pyramid, trust is a high-level and crucial concept: the Web will only achieve its full potential when users have trust in its operations (security) and in the quality of information provided. [Grigoris, 2004]

2.1.7 OWL

The OWL (Web Ontology Language) is designed for use by applications that need to process the content of information and perform useful reasoning tasks on the information instead of just presenting information to humans. It is developed from the DAML+OIL and has become the standard web ontology description language recommended by W3C. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S). It provides additional vocabulary along with a formal semantics, such as Local Scope of properties, Disjointness of classes, Boolean combinations of classes, Cardinality restrictions, and so on. According to the different requirement, OWL has three increasingly-expressive sublanguages [W3C Recommendation, 2004b], as shown in the following table:

Table 2.1. OWL sublanguages

Sublanguage	Description	Example
OWL Lite	supports those users primarily needing a classification hierarchy and simple constraint features.	supports cardinality constraints, and it only permits cardinality values of 0 or 1.
OWL DL	OWL DL (Description Logic) includes all OWL language constructs with restrictions that how the constructors from OWL and RDF may be used, such as type separation. It supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL was designed to support the existing Description Logic business segment and	a class may be a subclass of many classes, a class cannot be an instance of another class. a class cannot also be an individual or property, a property cannot also be an individual or class

	has desirable computational properties for reasoning systems.	
OWL Full	The entire language is called OWL Full and uses all the OWL languages primitives. It supports users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. It allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is too powerful for a reasoning software to support it completely.	a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

As we know, OWL builds on RDF and RDF Schema and uses RDF's XML-based syntax. Consider the relationships between the three sublanguages of OWL and RDF. OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF. Every OWL document is an RDF document, and every RDF document is an OWL Full document, but only some RDF documents will be a legal OWL Lite or OWL DL document.

Users should consider several rules when choosing which sublanguage best suits their needs. The main rules are listed as following:

- The choice between OWL Lite and OWL DL depends on the extent to which users require the more-expressive constructs provided by OWL DL.
- The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modeling facilities of RDF Schema (e.g., defining classes of classes, or attaching properties to classes).
- When using OWL Full as compared to OWL DL, reasoning support is less predictable because complete OWL Full implementations will be impossible. [W3C Recommendation, 2004b]

The detailed language features and specific syntax are defined at http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3. So they will not be discussed in this report.

2.2 Web Services

A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web services fulfill a specific task or a set of tasks. They can be used alone or with other Web services to carry out a complex aggregation or a business transaction. [Heather, 2001]

Web services combine the best aspects of component-based development and the Web. Like components, Web services represent functionality that can be easily reused without knowing how the service is implemented. Unlike current component technologies which are accessed via proprietary protocols, Web services are accessed via ubiquitous Web protocols such as HTTP, using universally-accepted data formats such as XML. [W3C Recommendation, 2005] Any type of application can be offered as a Web service. Web services are applicable to any type of Web environment: Internet, intranet, or

extranet. Web services can support business-to-consumer, business-to-business, department-to-department, or peer-to-peer interactions. A Web service consumer can be a human user accessing the service through a desktop or wireless browser; it can be an application program; or it can be another Web service.

2.2.1 Web Service Architectures

There are three roles in the Web service architectures: service provider, service registry and service requestor. The interactions among three roles involve publish, find and bind operations. Figure 2.2 illustrates these operations, the components providing them, and their interactions.

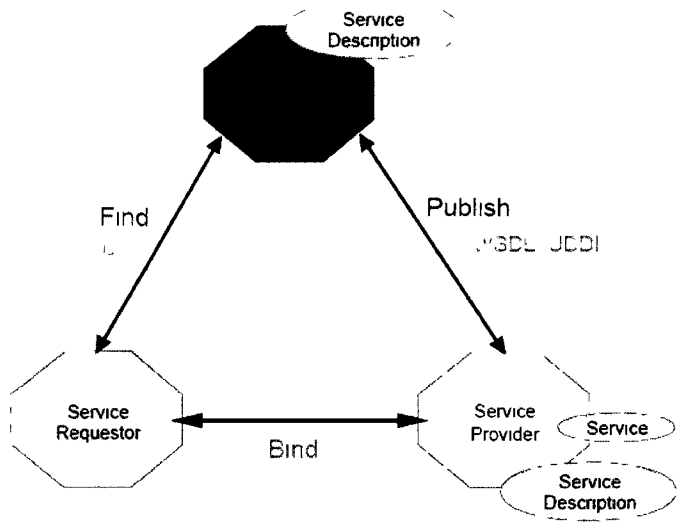


Figure 2.2 Web Services roles, operations and artifacts [Gruber, 1993]

The service provider is the owner of the service. From an architectural perspective, it is the platform that hosts access to the service. The service requestor is the business that requires certain functions. From an architectural perspective, it is the application that is

looking for and invoking an interaction with a service. The service requestor role can be browser-driven by a person or a program without a user interface, for example another Web service. The service registry is a searchable registry of service descriptions where service providers publish their service descriptions. Service requestors find services and obtain binding information (in the service descriptions) for services. [Gruber, 1993]

As shown in Figure 2.2, three operations are defined in the Web Service architectures: Publish, Find, Bind. [Gruber, 1993]

- Publish. To be accessible, a service description needs to be published so that the service requestor can find it.
- Find. In the find operation, the service requestor retrieves a service description directly or queries the service registry for the type of service required.
- Bind. In the bind operation the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

2.2.2 Web Services Stack and Related Technologies

To perform the three operations of publish, find and bind in an interoperable manner, there must be a Web Services stack that embraces standards at each level. Figure 2.3 shows a conceptual Web Services stack. The upper layers build upon the capabilities provided by the lower layers. The vertical towers represent requirements that must be addressed at every level of the stack. The text on the left represents standard technologies that apply at that layer of the stack.

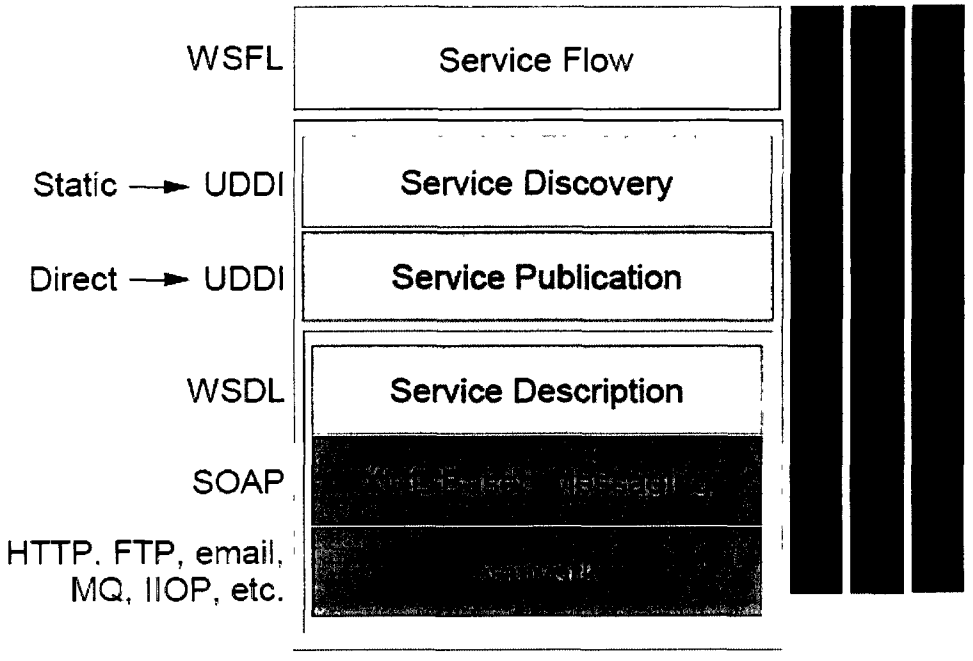


Figure 2.3 Web Services Conceptual Stack [Gruber, 1993]

The foundation of the Web Services stack is the network. Because of its ubiquity, HTTP is the standard network protocol for Internet-available Web Services. Other Internet protocols can be supported, including SMTP and FTP.

The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. SOAP (Simple Object Access Protocol) defines a standard communications protocol for Web Services.

The service description layer is actually a stack of description documents. WSDL (Web Services Description Language) is used for base-level service description. WSDL is an XML document for describing Web Services. WSDL can be created manually by XML editors or automatically by special tools like Java2WSDL from existing service interfaces.

UDDI (Universal Description, Discovery and Integration) is a standard mechanism to

register and discover Web Services. Although UDDI is often thought of as a directory mechanism like "yellow page", it also defines a data structure standard for representing service description information in XML and provides a Web based user interface to publish and query business information.

The publication of Web Services includes the production of the service descriptions and the subsequent publishing. A service description can be published using a variety of mechanisms. UDDI is the most often used mechanism for service publication and discovery.

The discovery of Web Services includes the acquiring of the service descriptions and the consuming of the descriptions. Acquiring can use a variety of mechanisms. Like publishing Web service descriptions, acquiring Web service descriptions will vary depending on how the service description is published and how dynamic the Web service application is meant to be. Service requestors will find Web Services during two different phases of an application lifecycle--design time and runtime. At design time, service requestors search for Web service descriptions by the type of interface they support. At runtime, service requestors search for a Web service based on how they communicate or qualities of service advertised.

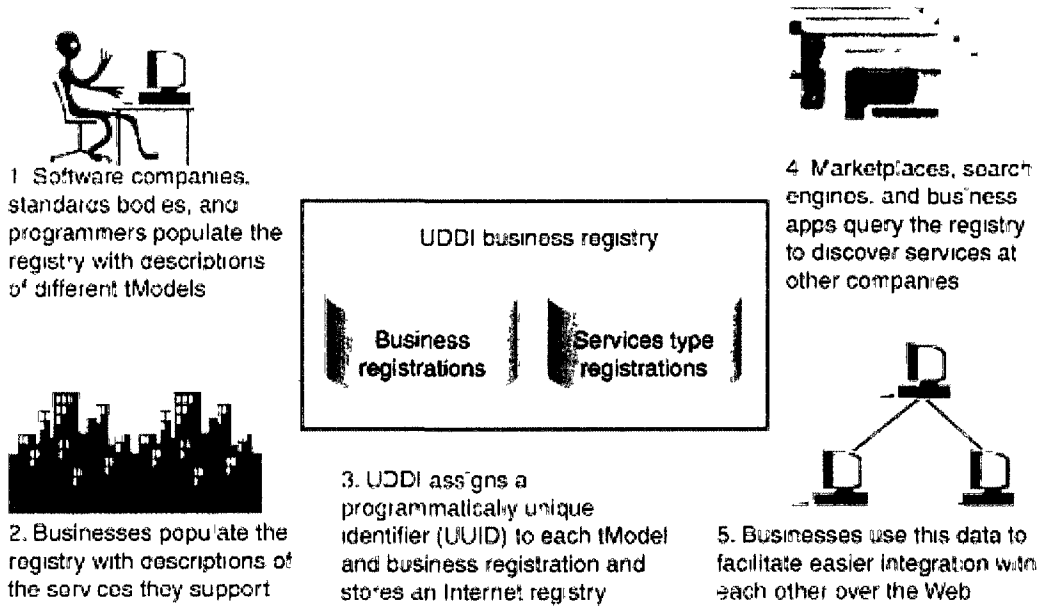


Figure 2.4 Working principles of UDDI [Fensel, 2001]

Figure 2.4 depicts how to send a message to the registry center, and how customers can discover and use the services. UDDI registry center is based on the data offered by the customers. As shown in Figure 2.4, there are several steps to make the best use of the data in the UDDI. When a service provider (software developers or business) wants to make the service or tModel available to service consumers, it describes the service using WSDL and registers the service in a UDDI registry. A technical specification is modeled as a tModel. A tModel can model many different concepts, such as, a type of service or a platform technology. The UDDI registry will then assign a UUID to each service or tModel and maintain pointers to the WSDL description and to the service. When a service consumer wants to use a service, it queries the UDDI registry to find a service that matches its needs and obtains the WSDL description of the service, as well as the access point of the service. The service consumer uses the WSDL description to construct a SOAP message with which to communicate with the service.

2.2.3 Semantic Web Services

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [Berners-Lee, 2001]

Web services are Web-based enterprise applications which are accessible over the Web and have interfaces that can be called from another program. A Web service is registered and can be located through a Web Service Registry, such as an UDDI Registry. Web services communicate by passing messages to each other, and support loosely coupled connections between systems. A Web service is described using a standard, formal XML notation, called its service description. Service description languages, such as the mainstream Web Service Description Language (WSDL), covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. [Systinet Corp., 2003]

The Semantic Web service is an integrated technology for the next generation of the Web. It combines semantic web technologies and web services and aims at turning the Internet from an information repository for human consumption into a world-wide system for automatic and distributed Web computing. The major difference between semantic web services and "regular" web services is that the descriptions of the semantic web services are well-defined in computer-interpretable forms. This will enable the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation.

2.3 Service Composition

Web services communicate by passing messages through interfaces. This enables developers or users to compose autonomous services to achieve new functionality. There are two types of Web services: simple and composite. Simple services are Internet-based applications that do not rely on other Web services to fulfill consumers' requests. A composite service is defined as a composition of outsourced Web services (called participant services) working in order to offer a value-added service. Actually, it is difficult to solve a real problem by using only a simple service. Service composition accelerates rapid application development, service reuse, and complex service consummation. It also reduces business risks since reusing existing services avoids the introduction of new errors.

There are two types of services compositions: static composition and dynamic composition. With static composition, the role of each participating service and the logical flow of messages between them are pre-defined by the developer manually at the design time. BPEL4WS (Business Process Execution Language for Web Services) [Andrews, 2003] or WSCI (Web Service Choreography Interface) [McIlraith, 2001], for example, are primarily designed for supporting this approach. While with dynamic composition, services to be composed are decided at the run time.

Static composition can support complex interaction patterns such as branch and loop, but it lacks adaptability therefore is not suitable for customers' changing requirements. Because the participating services may be collected from the inter organization domain, public or external, the accessibilities of each participating service may not be certain, and hence the composed service may not be guaranteed to be executable.

While with dynamic composition, given an ultimate goal and specified parameters for evaluating successful composition, the solution automatically generates the logical flow, identifies the candidate services, and composes them together. It is flexible and can be adaptive to different customer requirements and different environments.

Most methods to realize automated or semi-automated composition fall in the realm of workflow composition or AI planning. The definition of a composite service includes a set of simple services together with the control and data flow among the services. That is similar to a workflow. On the other hand, dynamic composition generates the plan automatically.

Different methods provide different levels of automation in service composition, but that does not mean the higher automation the better. Workflow methods are usually used in the situation where the requester has already defined the process model, but an automatic program is required to find and compose the simple services to fulfill the requirement. AI planning methods are used when the requester has no process model but has a set of constraints and preferences, and hence the process model can be generated automatically by the program.

A composed Web process can be executed either via a centralized approach or a distributed approach. The centralized approach is based on the client/server architecture, with a scheduler, which controls the execution of the components of the Web process. The controller/scheduler invokes a Web service, gets the results, and based on the results and the Web process design specification, the controller then invokes the next appropriate Web service. The distributed approach is more complex, in which each Web service hosts a coordinator component to share the execution context and collaborate with other

coordinators to realize the execution. The distributed approach usually is achieved through peer-to-peer communication or using agent based solutions.

2.3.1 Methods for Dynamic Service Composition

Dynamic service composition uses the notion of a semantic web service and methods for dynamically composing them. The major difference between semantic web services and regular web services is the descriptions of the semantic web services are well-defined in computer-interpretable forms. This will enable the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation.

Typically, the process of dynamic composition includes the following five phases: publication of simple services, translation of the definition language, generation of the composition process model, evaluation of the composite service, and execution of composite service. Several methods to realize automated or semi-automated composition have been proposed in the last decade.

In [McIlraith, 2002], a Golog-based[1] method to compose web services is presented. The authors adapt and extend the Golog language to perform automatic composition by applying logical inference techniques on pre-defined plan templates. The user's requirements and constraints can be presented by the first-order language of the situation calculus – Golog. The authors extend it to support sensing actions that can find values of variables at runtime. Essentially, Golog-based systems are user-provided plan

[1] Golog is a high-level logic programming language developed at the University of Toronto, for the specification and execution of complex actions in dynamical domains It is based on a formal theory of action specified in the situation calculus, a first-order logic language for representing dynamically changing world by reasoning about actions and changes [Lesp´erance, 1997]

templates which are modified based on user preferences at runtime. The final plan is generated automatically but the templates are not automatically built.

SWORD [Ponnekanti, 2002] is a developer toolkit for web service composition. In SWORD, a service is modeled by its preconditions and postconditions and represented by a rule expressing that given certain inputs, the service is capable of producing particular outputs. A rule-based expert system is then used to automatically determine whether a desired composite service can be realized using existing services. SWORD uses an Entity-Relation (ER) model to describe web services and does not support any existing service-description standards such as WSDL and OWL-S.

Semantic E-Workflow Composition [Cardoso, 2003] talks about service composition in workflow systems. A workflow is an abstraction of a process. It is built using components called tasks or activities. The design of traditional workflow application selects the appropriate tasks from a workflow repository which contains modest number of tasks therefore the process is humanly manageable. The authors of [Cardoso, 2003] devised an algorithm to discover and select appropriate web services by using a feature-based model to find similarities. Issues about how web services can be integrated into workflows by syntactic, operational metrics, and semantic integration of inputs and outputs are also discussed in [Cardoso, 2003]. More details about the feature-based model can be found in Chapter 3.

SHOP2 [Wu, 2003] is HTN-based planner for composing web services. HTN (Hierarchical Task Network) planning is an AI planning method that creates plans by task decomposition. "This is a process in which the planning system decomposes tasks into smaller and smaller subtasks, until primitive tasks are found that can be performed

directly." [Wu, 2003] The authors find that the concept of task decomposition is similar to the concept of composite process decomposition in OWL-S process ontology. The authors also give a very detailed description on the process of translating OWL-S to SHOP2. HTN planner is more efficient than other planning language such as Golog. However, the SHOP2-based composition requires an assumption that each simple service either produces outputs or effects but not both. This can distinguish between information-gathering services and effect-producing services. To meet this assumption, each service used for composition has to produce only outputs or only effects.

There are over a dozen proposals in the literature developing QoS ontologies to play a part in web services composition. See Tran [2008] for a useful tabulation of eight of them. Seven of these proposals do not provide support comparison of QoS values while WS-QoSOnto [Tran 2008] provides weak support. Some QoS ontologies are blurred with domain ontologies and these suffer more from maintenance issues. The pure QoS ontologies can provide complementary infrastructure to FOIQOS for similarity matching when more complex QoS parameters are considered such as security. Although there are WS protocols, such as WS-Security, that can handle this issue.

In [Ye, 2006], the author introduced a novel and very interesting approach to perform the matching between a web service and a request by using mappings between web services and the domain ontology. The basic idea is to simply replace the terms in a description (could be web service description or request description) by the concepts defined in the domain ontology. By this means, both service descriptions and request descriptions can be formalized by the concepts within the same domain ontology and therefore the matches could be more easily and precisely. A thing should be noticed is

that the range (e.g. 100<price<200) defined in a constraint should be formalized also. The paper realized this by converting all numerical constraints into "interval sets". By using this kind of annotation mechanism, the matching between a web service and a request is converted to the matching between their semantic annotations. However, to support the annotation mechanism, a set of preprocessing steps should be taken before web service publishing and discovering. Specifically, before the publishing phase, the following preprocessing should be finished: extract the functional descriptions of a web service from its definition documents; formalize the functional descriptions through the annotation mechanism; build the mapping between web services and the ontology. In a similar manner, before the discovering phase, the functional descriptions should be extracted from a request and formalized to the semantic annotations.

A current UDDI registry only provides keyword-based discovering methods, which are not strong enough to meet the application needs. The simplest way to publish semantic information of a service and service properties like QoS is to register T-models which refer to the external description files. However, this approach has very poor efficiency. [Liu, 2005] proposed a domain oriented UDDI registry architecture. An external centralized database to store service-related information and service properties is used in this architecture. The interesting and useful idea in this architecture is: they assume all the services belong to the same category of the taxonomy would have the similar characteristics. So a service property schema, regarding the service properties, constraints and semantic information, is defined for each category. All the services published in a category share the same service property schema. For service discovering, a service requester can get the service property schema based on the category of the

requested service and then perform searching based on the properties defined in the schema. The major advantage of this idea is the properties used in discovering are not fixed. It is definable according to different types of services.

[Thissen, 2006] considered QoS aspects when selecting candidate services for a composition and developed a QoS broker to complement the UDDI registry with non-functional aspects. Aggregation formulas of simple composition patterns are applied to the whole workflow of a composed service to select the most suitable collection of simple services. The paper uses a bottom-up composition to compute the QoS of the composed service. With the workflow pattern for a composed service, aggregation is done by collapsing the composition graph step-wise into a single node based on the basic composition patterns. A set of formulas was defined to model the aggregations of the QoS parameters typically used in web service composition. QoS aggregation is very useful in AI planning dynamic composition, in which the workflow are generated automatically when given a ultimate goal and specified parameters. Because the composition process is not predefined, there could be many different possible combinations of web service. QoS aggregation can be used to select the best combination by providing methods to compare the overall QoS values of the combinations instead of considering only a single web service.

[Nie, 2006] proposed a definition language to describe user requirements by integrating semantic descriptions and SLA (Service Level Agreement) with the process description. This description language supports the dynamic composition and can adapt the change of QoS constraints automatically when SLA violations occur. The proposed description language describes user requirements of service composition in five ways: 1.

The semantic description declares the user profile and domain constraints of a web service; 2. The SLA description provides the negotiations between service providers and service requestors regarding the quality of service, and discovers appropriate web services based on QoS constraints; 3. The policy description defines the compensation policy based on user requirements when SLA violations and service faults occur; 4. The service partner declaration defines abstract partner names which are used to replace the web service entities from the business process to support the dynamic composition. 5. The business process description defines the business process of the composition which provides the definition of message type, variables declaration and executing process. The description language proposed in the paper supports the dynamic service composition on demand very well. Firstly, it allows service requestors to describe the composite service clearly and dynamically. This is realized by using a concept called "abstract partner". An abstract partner is basically a template, in which the profile, semantic constraints, and SLA of a requested service are defined. A composite service is defined in four components: semantic descriptions, SLA descriptions, a set of abstract partner descriptions, and the process descriptions.

In [Fang, 2009], the author presents a novel global QoS optimizing and multi-objective Web Services selection algorithm based on a Multi-objective Ant Colony Optimization (MOACO) for the Dynamic Web Service composition. Ant Colony Optimization (ACO) is a meta-heuristic proposed by Dorigo et al. [Dorigo, 1999]. The basic idea is to model the problem to solve the search for a minimum cost path in a graph, and use artificial ants to search for good paths. The MOACO approach first generates an Abstract Service Plan, which is a combination of composition work-flow and service

templates. This Abstract Plan is composed of individual Web Service Types, which specifies web service functional properties (IOPE). Web Service Type definition is very similar to ST definition in the original E-Workflow Composition approach. Each web service instance must be categorized to a web service type. The MOACO approach then models a Web Service Instance Selection graph based on all the web service instances and the Abstract Service Plan, and applies the Global QoS Optimizing Web Services Selection Algorithm on that graph to find out the optimized paths which meet the global QoS constraints set by the user. The key advantage of the approach is that a user can set the "global" QoS constraints (most approaches just find the path to meet each local QoS constraints. But that doesn't guarantee the selected path will meet the global constraints.); another interesting point of the approach is that the algorithm can find a set of possible paths which meet the global QoS constraints. However, as a learning-based algorithm, MOACO takes much longer execution times than other algorithms. And the accuracy of the results is highly dependent on the number of ants and the number of the iterations. This approach is good enough to handle the sequence process, but for the parallel process, it need some tweaks to the algorithm, e.g. treat the web service types involved in a parallel process as an individual web service type.

Chapter 3

System Design and Methods

This thesis covers several technologies including semantic web services and dynamic service composition. This chapter will introduce the proposed FOIQOS (Flexible Ontology Independent QoS-enabled) approach for dynamic web service composition. The system design and the detailed methodologies used in the approach will be discussed in this chapter. Specifically, we will introduce main QoS metrics for web services, and similarity measures for syntactic, semantic, and operational matching.

3.1 System Design

As shown in Figure 3.1, the central part of the dynamic web services composition system is a software agent, named the Compose Agent, which will read in Process (workflow) and Templates (demands) as inputs, and realize service discovering and composition automatically. There are two types of inputs, one is Predefined Process, which defines the workflow of the whole task; the other is Templates for each service, which defines each individual service involved in the task. Templates are formalized demands which provide expected service profiles like service names, service descriptions, and QoS parameters. The Compose Agent reads in Templates and search for the best participating services (service discovering) by performing matchmaking between the templates and related web services, and composes the selected web services automatically. This thesis proposes the FOIQOS approach for service discovery. See Figure 3.1 for how FOIQOS fits in the services composition system. Automatic service discovering is the most important function within this work and is detailed in Figure 3.2.

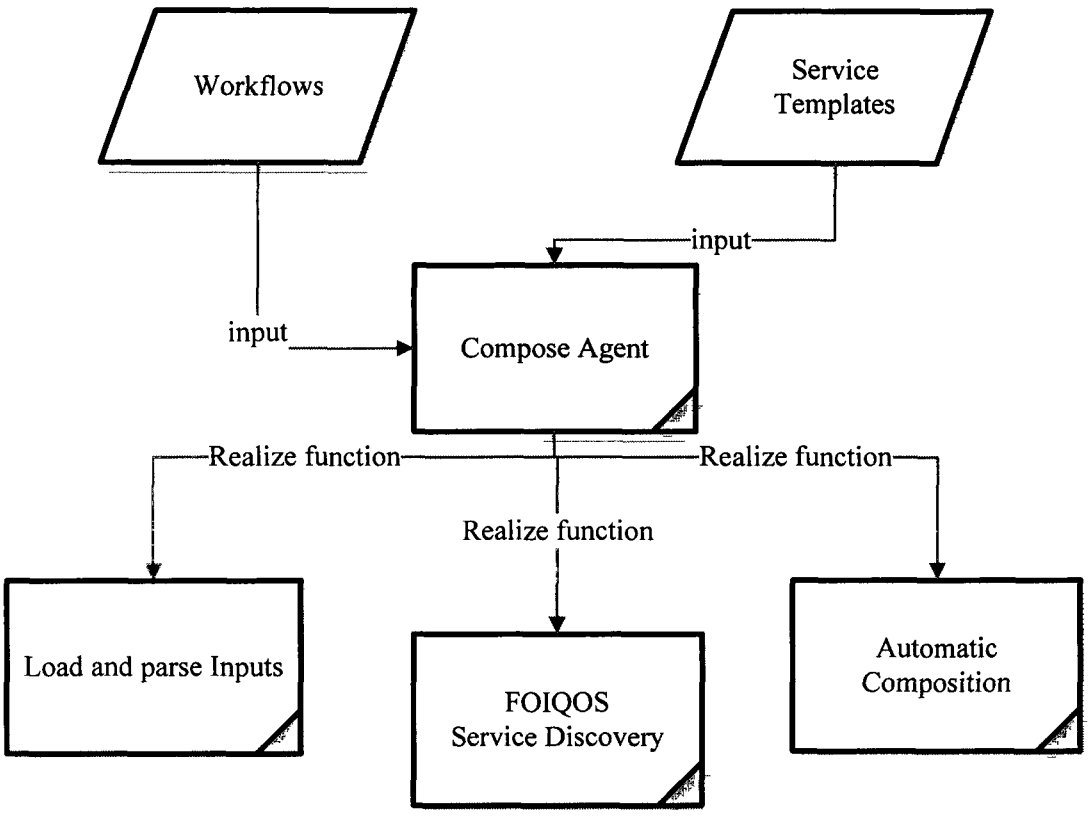


Figure 3.1 System architecture

The proposed FOIQOS approach for service discovery consists of three main components of which the proposed use of Google Distance for the semantic matching stage is the main innovation. The three components of FOIQOS are:

(1) Syntactic matching using the Q-gram method.

(2) Semantic matching using Google Distance to remove dependence on ontology usage.

(3) QoS parameter matching using relative distance calculations vs. absolute distance in operational matching formulae.

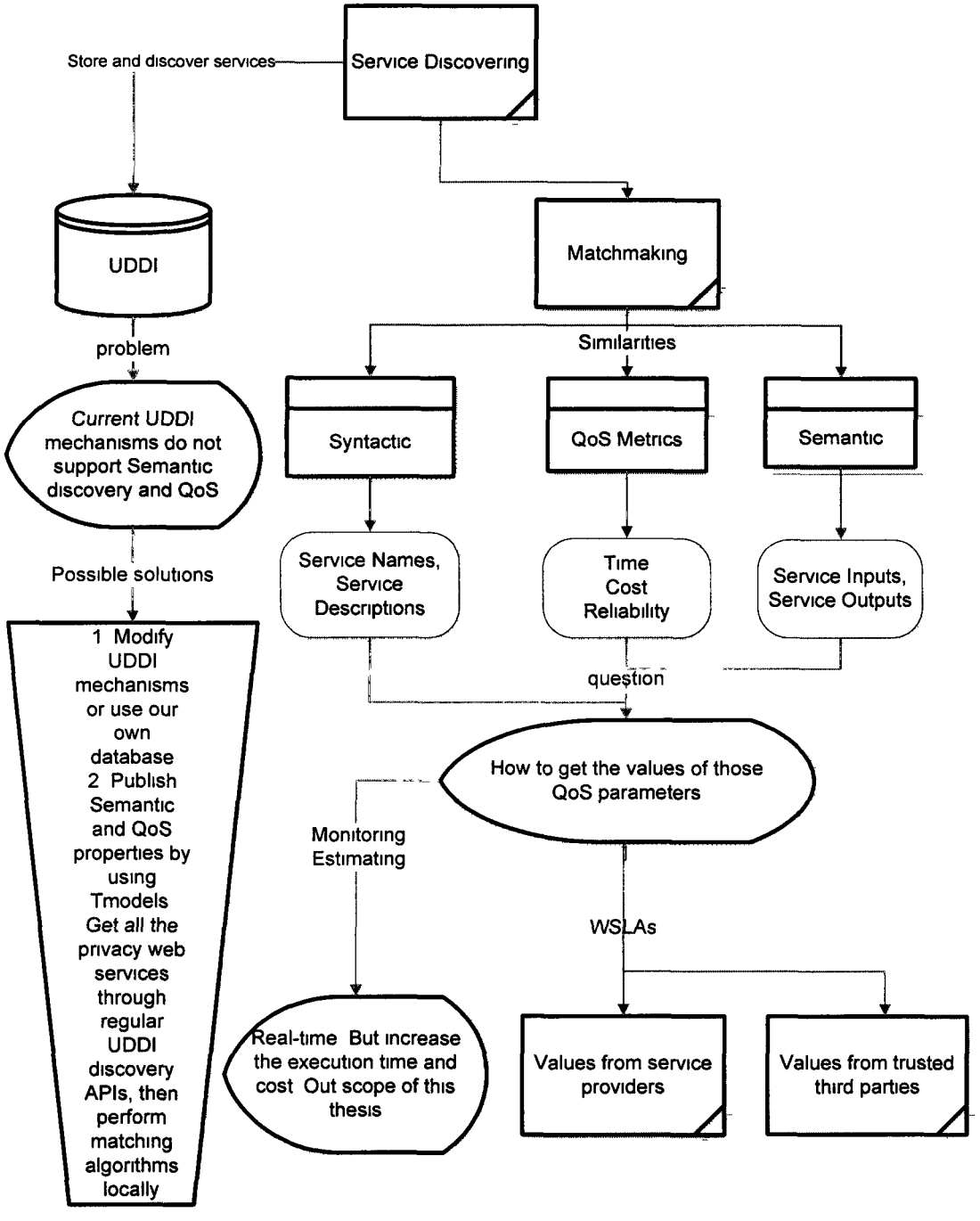


Figure 3.2 Service discovering process, questions and possible solutions

Currently, the industry standard for web service publishing/discovery is UDDI version3, which does not support the semantic descriptions and operational properties

(QoS) of web services. Although using our own web services database will make it easier for us to store and search the semantic descriptions of web services, following the industry standard can make our work more general and more acceptable to the public. So, in this work, we still prefer to use the UDDI registry (public/private) as the directory storage mechanism. However, current UDDI APIs only support key-words based discovery. Although we can store QoS information and semantic description in a UDDI directory by using tModels, how to read them out and perform matchmaking based on them is a problem. A simplified approach would be to create customized categories (e.g. Privacy) in the UDDI and we assume all the web services in, for example, the privacy domain must belong to Privacy category. During the discovering process, we first get all the Privacy web services by using regular UDDI discovery APIs, and then perform the syntactic matching, semantic matching, and QoS matching locally. Investigating approaches to extend the UDDI APIs for fast QOS processing is for future work.

3.2 Methods for Similarity Measurement

The design of traditional workflow application selects the appropriate tasks from a workflow repository. If the repository contains a modest number of tasks, the process is manageable. However composing web services within a workflow application is not that easy because the potential number of web services for the composition process can be very large. The designer faces two problems: (1) discovering a Web service with the desired functionality and operational metrics to accomplish a specific task; (2) resolving the structural and semantic differences between the services found and the tasks within a workflow.

Discovering a Web service manually is impossible, since thousands of services are

available on the Internet. One approach to discover and select appropriate web services is using feature-based models to find similarities across tasks (activities) and Web service interfaces. Web service interfaces are constructed using service templates. We borrow the following terminology from [Cardoso, 2003].

A service template represents a structure or blueprints that the designer uses to indicate the characteristics of the desired Web service. A service template is specified as: ST = <sn, sd, QoS, Os, Is>. The five fields here: sn, sd, QoS, Os, Is, correspondence to service name, service description, quality of service (operational metrics), output parameters and input parameters. Term service object are used to indicate the potential services can be applied to service template. It is specified as SO = <sn, sd, QoS, Os, Is>. The five fields here have the same meaning as the ones defined in ST.

After creating service templates, the solution discover and select appropriate web services by computing the similarities between SO and ST. The similarities here include three parts: syntactic similarity, operational similarity, and semantic similarity.

3.2.1 Syntactic similarity

In [Cardoso, 2003] the syntactic similarity of a ST and a SO is computed by using "string matching" method based on their service names and service descriptions. Formula (1) shows the function to calculate the syntactic similarity. The functions SynNS and SynDS are used to compute the similarities between two service names, and two service descriptions, respectively. The two weights ω_1 and ω_2 indicate the degree of confidence that a designer has in the service name and service description he supplied when constructing a ST.

$$SynSimilarty(ST, SO) = \frac{\omega_1 SynNS(ST.sn, SO.sn) + \omega_2 SynDS(ST.sd, SO.sd)}{\omega_1 + \omega_2};$$

Formula (1)

Many algorithms (e.g. Hamming distance, edit distance, block distance, q-gram, TF-IDF) have been proposed to perform "string matching" tasks: hamming distance is defined as the number of bits which differ between two binary strings. This approach is only suitable for exact length comparison [Chapman, 2006]; edit distance is defined as the minimum edit steps to transform one string to another string. The typical edit operations are defined as following: 1) Copy character from string1 to string2 (cost 0), 2) Delete a character in string1 (cost 1), 3) Insert a character in string2 (cost 1), 4) Substitute one character for another (cost 1) [Gilleland, 2006]; block distance is a vector based approach where two strings are defined as two points in n-dimensional vector space and the distance is calculated by summing the edges between points that must be traversed to get from one string1 to string2 [Teknomo, 2006]; q-gram is an approach typically used in approximate string matching. "q-gram" is realized by first "sliding" a window with length q over the characters of a string to create a collection of 'q' length grams, then rating the number of q-gram matches within the second string over q-grams collected from first step. Q-gram is intuited by the fact that when two strings are similar to each other (the edit distance between two strings is small), the number of the same q-grams they share is large [Chapman, 2006]; TF-IDF (Term Frequency-Inverse Document Frequency) is a technique borrowed from the information retrieval area. It is also a vector-based approach and typically used to calculate the relevance of text documents. Considering that strings are short text documents, especially in our case, a service description usually contains one or

more sentences, TF-IDF is also a reasonable approach to compute the syntactic similarities. TF-IDF realized by consider each text document as a vector and all the words within all the text documents (in our case, all the strings) as attributes of the vectors. Then the weight of each word in a document is obtained by calculating term frequency and inverse document frequency and this weight is used as the value of the attribute according to the word. Finally, we calculate the distances between vectors to get the most similar document.

This thesis will implement and evaluate the performance of the q-gram approaches in computing similarities of service names and service descriptions. To achieve a better performance, data preprocessing algorithms often used in the information retrieval area will be employed also. Specifically, this thesis will consider word-stemming[2] and stop words[3] removal in the data preprocessing part.

3.2.2 Semantic similarity

Semantic similarity could contain two meanings: (1) the semantic similarity between two inputs/outputs of a ST and a SO; (2) the semantic similarity between the outputs of a ST and the inputs of a SO, which is a candidate service of the next ST in the workflow; and the semantic similarity between the outputs of a SO and the inputs of the next ST in the workflow.

---

[2] In most cases, morphological variants of words have similar semantic interpretations and can be considered as equivalent for the purpose of IR applications For this reason, a number of so-called *stemming Algorithms*, or *stemmers*, have been developed, which attempt to reduce a word to its *stem* or root form Thus, the key terms of a query or document are represented by stems rather than by the original words This not only means that different variants of a term can be *conflated* to a single representative form – it also reduces the *dictionary size*, that is, the number of distinct terms needed for representing a set of documents A smaller dictionary size results in a saving of storage space and processing time [Lancaster University, 2004]

[3] Words which are very frequent and do not carry meaning (such as "a", "the") are called stop-words These words are assumed not to carry any important information and so are usually ignored in order to save storage space of the inverted file First, you should define the list of stop-words Then, when you read in a new document you should remove all the stop-words before proceeding to the next stage [Hong Kong University, 2004]

The meaning (1) is straightforward. It is to compare a service template to all the candidate services in terms of the inputs and outputs. It can be described as Similarity(ST.INs, SO.INs) and Similarity(ST.OUTs, SO.OUTs).

The meaning (2) falls into the realm of web service integration. Consider the "sequence" operation within a process model. It requires that the outputs (or part of the outputs) of the former service should be the inputs (or part of the inputs) of the later service. It can be described as in Figure 3.3.

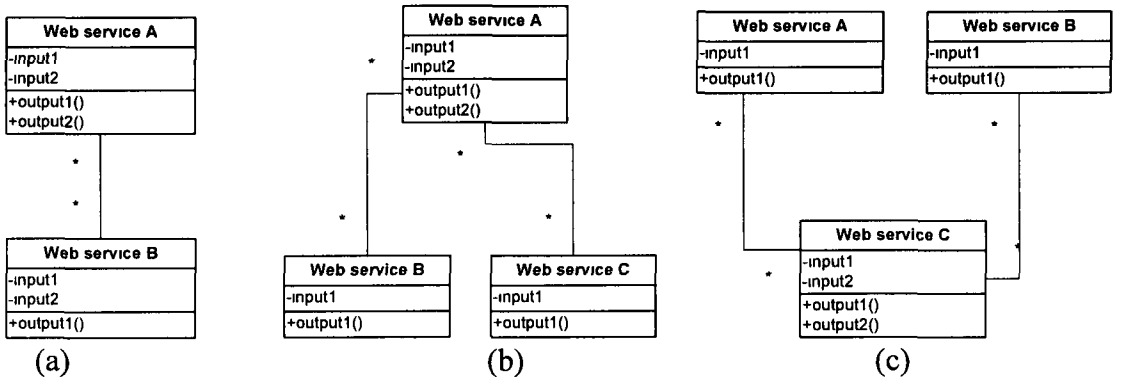


Figure 3.3 (a) shows the outputs of web service A are the inputs of web service B; (b) shows the outputs of web service A are the inputs of web service B and web service C; (c) shows the outputs of web service A and outputs of web service B are the inputs of web service C

Both meanings of semantic similarity can be realized by evaluating the similarity of two concepts associated with an output and an input. In the real case, there should be existing domain ontology for all the inputs and outputs. The similarities are calculated based on concepts and their properties within that domain ontology.

3.2.2.1 Semantic similarity through feature-counting using an ontology

In [Cardoso, 2003], the authors proposed a similarity function as showed in Formula (2) based on a general feature-counting model introduced by Tversky [Tversky,

2005]. The basic idea is that "common features tend to increase the perceived similarity of two concepts, while feature differences tend to diminish perceived similarity." [Cardoso, 2003]

$$similarity'(O,I) = \sqrt{\frac{|\ p(o) \cap p(I)\ |}{|\ p(o) \cup p(I)\ |} * \frac{|\ p(o) \cap p(I)\ |}{|\ p(I)\ |}} \qquad \text{Formula (2)}$$

Here, p(x) indicates all the properties associated with a concept. So that the similarity between two concepts can be approximated by the number of properties shared among two concepts. Figure 3.4 is a fragment of a sample ontology designed by me to demonstrate the concepts and related properties and relationships within a Race ontology. It is used as an example to illustrate how to calculate the semantic similarities by using the methods introduced in [Cardoso, 2003].

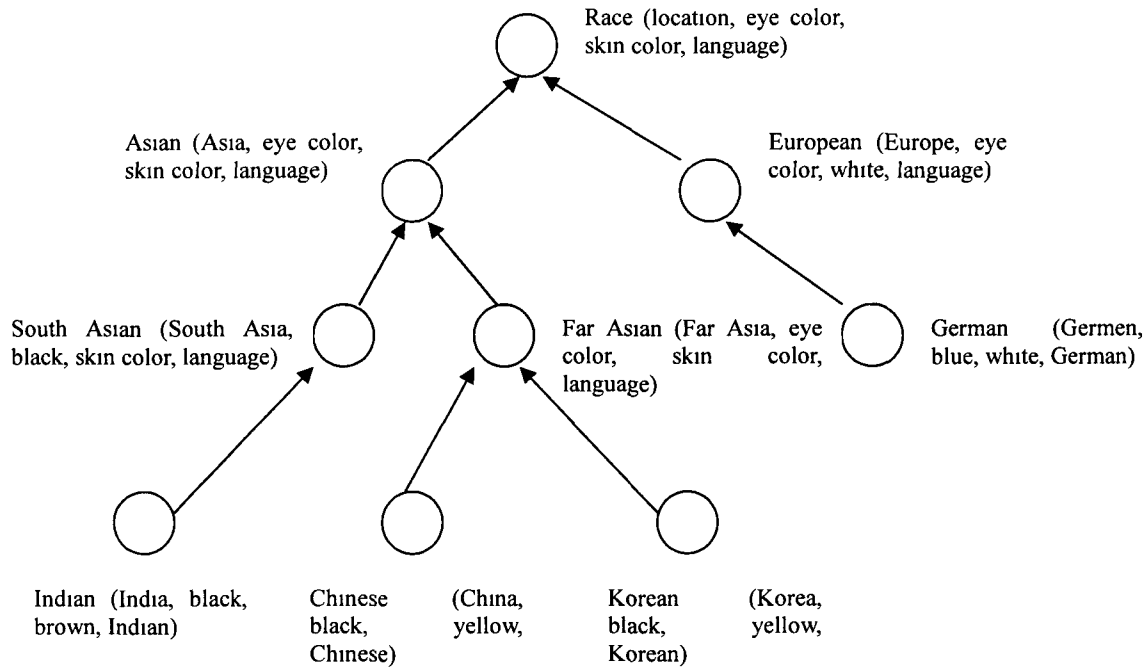


Figure 3.4 fragment of race ontology. Each concept within the ontology has four

properties, location, eye color, skin color, and language.

As discussed above, the semantic similarity is calculated based on the number of properties shared among two concepts. So, the similarity between Chinese and Korean is calculated as following:

S_1 = p(Chinese) = (China, black, yellow, Chinese);

S_2 = p(Korean) = (Korea, black, yellow, Korean);

S_3 = p(Chinese) $\cap$ p(Korean) = (black, yellow);

S_4 = p(Chinese) $\cup$ p(Korean) = (China, black, yellow, Chinese, Korea, Korean)

$$similarity(Chinese, Korean) = \sqrt{\frac{|S_3|}{|S_4|} * \frac{|S_3|}{|S_2|}} = \sqrt{\frac{2}{6} * \frac{2}{4}} \approx 0.408$$

3.2.2.2 Semantic similarity through Google distance

Google distance is a novel approach proposed by Paul Vitanyi and Rudi Cilibrasi in [Vitanyi, 2004] to realize automatic meaning extraction. This approach is based on the fact that when the Google search engine is used to search for a particular term it will return the number of hits (web pages containing that term). Suppose the hits returned by the Google search engine for the term "hat" is H1, and the total number of the web pages Google could have returned is H2, then the result of H1/H2 can be treated as the probability that term "hat" occurred in the world wide web. This actually complies with the definition of "marginal probability"[4] in the theory of probability. It together with

[4] Marginal probability is the probability of one event, ignoring any information about the other event Marginal probability is obtained by summing (or integrating, more generally) the joint probability over the ignored event The marginal probability of h is $P(h)$, and the marginal probability of e is $P(e)$

conditional probability, joint probability and Bayesian' theorem, can be used to represent the relationships between two terms.

Conditional probability is defined as the probability that the event h will occur, given the knowledge that the event e has already occurred. Conditional probability is written P(h|e), and is read "the probability of h, given e" [Wikipedia, 2004]. In the case of Google distance, conditional probability can be defined as the probability that term B appears on a webpage, given the condition that the term A also appears on the webpage. This is the basis of Google distance, because higher conditional probabilities imply a closer relationship between the two terms.

Joint probability is defined as the probability of two events in conjunction. That is, it is the probability of both events together. The joint probability of h and e is P(h, e). In the case of Google distance, joint probability can be defined as the number of the hits when input both term A and term B as a query string into Google search engine.

Bayes' theorem is defined in Formula (3):

$$P(h \mid e)P(e) = P(h,e) = P(e \mid h)P(h)$$

$$P(h \mid e) = \frac{P(e \mid h)P(h)}{P(e)} \qquad \text{Formula (3)}$$

Based on Bayes' theorem, we can easily get the conditional probabilities which reveal the relationships between two terms. The original distance function proposed by Paul Vitanyi and Rudi Cilibrasi to calculate the distance between term A and term B is simply the minimum number of P(A|B) and P(B|A).

The normalized Google distance (NGD) function is as following:

$$NGD(x,y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{\log M - \min\{\log f(x), \log f(y)\}} \qquad \text{Formula (4)}$$

Where f(x) is defined as the number of hits a Google search for term x returns. M is the overall number of web pages that Google indexes.

The research in [Evangelista, 2006] tested the assumption that the NGD of two random and independent terms should be close to 1. The testing results showed the expectation value of the distance between two random and independent words is not 1 but 0.7. In order to achieve the desired value of unity between independent words, it is necessary to recalibrate the NGD function (Formula (4)) by dividing by 0.7:

$$NGD^{*}(x,y) = \frac{NGD(x,y)}{0.7}$$

The authors of [Evangelista, 2006] also concluded that NGD values depend on the number of hits that each term has. Factors such as which Google server was connected to and the number of websites connected to the world-wide-web can yield different NGD values. So the NGD values are not stable and accurate enough.

This thesis will investigate the performance of both the feature-counting model coupled with ontology use, and our proposed Google distance-based approach within business domains.

3.2.3 Operational similarity

The operational similarity of ST and SO is calculated based on QoS metrics. The purpose is to determine the best candidate web service based on operational capabilities of each SO and the QoS values defined in a ST.

3.2.3.1 Definition of QoS metrics

Quality of service can be characterized according to various dimensions. The QoS metrics will be considered in this work involve following dimensions (parameters): Time (T), Cost (C), and Reliability (R).

Time is a commonly used performance measure. To simplify, this thesis considers Time (T) as the total processing time of a web service, from invocation to result output on a device.

Cost (C) corresponds to the cost associated with the execution of a web service. Cost is an important QoS parameter in the real world. In the real case, some web services are not free, so customers have to pay for using web services. That is why some research works also called it Price [Li, 2005].

Task reliability (R) is a function (see Formula (5)) of the failure rate which corresponds to the likelihood that a component will perform when the user demands it.

$$R(t) = 1 - \text{failure rate} \qquad \text{Formula (5)}$$

Here failure rate is given as the ratio of successful executions, which is computed as the number of times the task has been scheduled for execution and how many times the task has not successfully executed. In the real case, failure rate can be computed in terms of the total amount of time in which a service is not available during a given time interval. This is similar to the definition of Unavailability in [Li, 2005].

3.2.3.2 Computing operational similarities

In [Cardoso, 2003], the operational similarity of a ST and a SO is computed by using the "geometric distance" method based on their QoS parameters as shown in Formula (6). The idea is to determine how close the operational capabilities of two Web services are;

$$OpSimilarity(ST, SO) = \sqrt[3]{QoS\dim D(ST, SO, time) * QoS\dim D(ST, SO, \cos t) * QoS\dim D(ST, SO, reliability)}$$

Formula (6)

The distance of two QoS parameter values is calculated using function QoSdimD(ST, SO, dim), where dim is a QoS parameter. The function calculates the geometric distance of the ST and of the SO by using Formula (7).

$$QoS\dim D(ST, SO, \dim) = \sqrt[3]{dcd_{\min}(ST, SO, \dim) * dcd_{avg}(ST, SO, \dim) * dcd_{\max}(ST, SO, \dim)}$$

Formula (7)

Here $dcd_{\min}$ indicates the distance between the minimum values of a QoS parameter in ST and SO and calculated by Formula (8). Similarly, dcd_{avg} and $dcd_{\max}$ respectively indicate distances between the average values and maximum values of a QoS parameter in ST and SO.

$$dcd_{\min}(ST, SO, \dim) = 1 - \frac{|\min(SO.qos(\dim)) - \min(ST.qos(\dim))|}{\min(ST\ qos(\dim))}$$ Formula (8)

The main problem with this approach is that regarding the quality of the service, the web service with the highest similarity value may not be the one with best QoS values. For example, considering the QoS parameter time, when a service requester defines time in a ST, usually, it only cares about the longest process time (t1) acceptable to it and possibly it will define the preferred process time (t2) as well. With the approach in [Cardoso, 2003], the algorithm will tend to choose a service which process time is between t1 and t2 rather than choose a service which process time is shorter than t2. That can be improved because regarding the QoS parameter time, the shorter the better.

This thesis will define only an acceptable value for each QoS parameters in ST and

suppose each QoS parameters in SO is defined by (min, avg, max) triple. Different to the reliability function (Formula (5)) used in [Cardoso, 2003], this thesis uses failure rate to represent the reliability. This is to make sure that for values of all the three QoS parameters, the smaller values are always better than larger ones. The similarity is calculated base on two rules:

(a), the values in the triple must be better than the acceptable value in ST (e.g. for time, the minimum time, maximum time and average time of SO must be less than the acceptable time defined in ST);

(b), the similarity should be calculated using Formula (9):

$$\begin{cases} \{\min(SO.qos(\dim)) \prec ST.qos(\dim)\} \& \{avg(SO.qos(\dim)) \prec ST.qos(\dim)\} \& \{\max(SO.qos(\dim)) \prec ST.qos(\dim) \\ dcd_{\min}(ST,SO,\dim) = 1 - \dfrac{ST.qos(\dim) - \min(SO.qos(\dim))}{ST.qos(\dim)} \\ dcd_{avg}(ST,SO,\dim) = 1 - \dfrac{ST.qos(\dim) - avg(SO.qos(\dim))}{ST.qos(\dim)} \\ dcd_{\max}(ST,SO,\dim) = 1 - \dfrac{ST.qos(\dim) - \max(SO.qos(\dim))}{ST.qos(\dim)} \\ QoS\dim D(ST,SO,\dim) = 1 - \sqrt[3]{dcd_{\min}(ST,SO,\dim) * dcd_{avg}(ST,SO,\dim) * dcd_{\max}(ST,SO,\dim)} \end{cases}$$

Formula (9)

Notice that the last equation in Formula (9) is different from Formula 4. This is because in this model, the smaller the value of a QoS parameter of SO is, the larger the distance (dcd) between that value and the acceptable value defined in ST is. As defined in rule (a), the values in the triple of SO must be better than the acceptable value in ST, which means dcd is inversely proportional to the distance between ST and SO.

3.2.3.3 Approaches to obtain QoS values

In order to facilitate the operational similarity of QoS parameters, it is necessary to define methods to obtain the values of QoS parameters discussed above. The possible

approaches include: (1) monitoring the changes of the QoS values at the run time by updating those values after each execution, or (2) estimating QoS values through (a) simulation mechanisms (e.g. test the task based on specific inputs); (b) using the QoS values provided by service providers; and/or (c) obtain QoS values from TTP (Trusted Third Party) which provides QoS testing services.

Both monitoring approach and estimating approaches can get correct and real-time values of QoS parameters, but they will increase the execution time and/or cost of the system and is out of the scope of this research. To simplify this process my thesis will only investigate and implement the latter two approaches (b) and (c) and incorporate SLA (Service Level Agreement) constraints. Based on the two copies of QoS parameter values obtained from the service provider and TTP, the system selects the best participation services through certain rules. For example, if the two copies have similar values, we consider it more reliable.

SLA is a part of a service contract used by both service provider and requestor regarding the performance and cost. It regulates the common understanding about services with the main purpose to form an agreement on the level of service. Usually, a SLA may specify the levels of availability, performance, or other attributes of the service. Generally, the technical interpretation of SLA is described through a set of SLO (Service Level Objective), which contains one or more service parameters in terms of quality measurements.

WSLA stands for Web Service Level Agreement, which is a standard published by IBM for specifying and monitoring SLAs for Web Services. It enables web service providers and requestors to define a variety of SLAs, specify the QoS parameters and the

related measurements. [Keller, 2003] In addition to the service provider and service requestor, WSLA may also involve third parties in the SLA monitoring and enforcement process. This happens when SLAs published by a service provider are not fully trusted by service requestors. Those trusted third parties perform a part of all of the measurement and computation activities defined within an SLA. They also implement violation detection by comparing actual values against the values provided by service providers.

This dissertation examines the use of WSLA to describe the QoS parameter values provided by service providers and trusted third parties, and use those values to calculate the operational similarities between STs and SOs.

Chapter 4

Implementation

The following subsections discuss the implementation of the composition system proposed in Chapter 3. To implement the proposed FOIQOS system, we need to realize the following tasks:

- Set up the development and test environment;
- Get a UDDI registry to register / discover web services. Since all the public UDDI registry were retired already, we have to create our own private UDDI registry;
- Find a way to publish and inquiry the semantic descriptions and QoS parameters in UDDI registry;
- Create sample workflow and service templates as input; Read in the input and based on the inputted service templates to find the proper web service instances and compose them together based on the inputted workflow;
- Implement the FOIQOS service discovering method consisting of implementing Google distance and the modified similarity matching methods discussed in Chapter 3.

To compare FOIQOS proposal with a workflow-based method and an AI planning-based web services discovery method, I implement also (1) the E-Workflow composition method, and (2) the MOACO algorithm, and complete the following implementation tasks.

- Implement dynamic web service composition within a Compose agent;
- For testing, create and publish a large number, say n= 100, of web services onto a

UDDI registry;

4.1 Tools involved in the implementation

The following tools and public resources were involved in the implementation:

- Java EE SDK 5 Update 2 from SUN Microsystems, Inc as the development language and runtime environment
- JUDDI version 2.0.1 from the Apache Software Foundation. It is a Java implementation of the UDDI version 2.0 specification. And it is used as our private UDDI registry
- MySQL 5.086 community Database server from MySQL AB as the database server in support of the UDDI registry
- Tomcat 5.5.27 application server from Apache Software Foundation. Our private UDDI registry, web services, and compose agent are all run on that server;
- UDDI4J version 2.0.5 from IBM and HP. It provides a Java API to interact with a UDDI registry
- Eclipse SDK version 3.2.2 from the Eclipse Foundation. It is a open source Java development platform
- Apache Axis2 version 1.4.1 from Apache Software Foundation. It is an implementation of the SOAP (Simple Object Access Protocol)

For experimental purposes, the system presented in this thesis is tested on an Intel Core 2 Due CPU @ 3GHz computer with 3GB memory. As shown in Figure 4.1, the JUDDI registry with mySQL database server, the Apache Tomcat application server to

deploy the UDDI registry, web services, and the Compose Agent, will be located on the same machine.

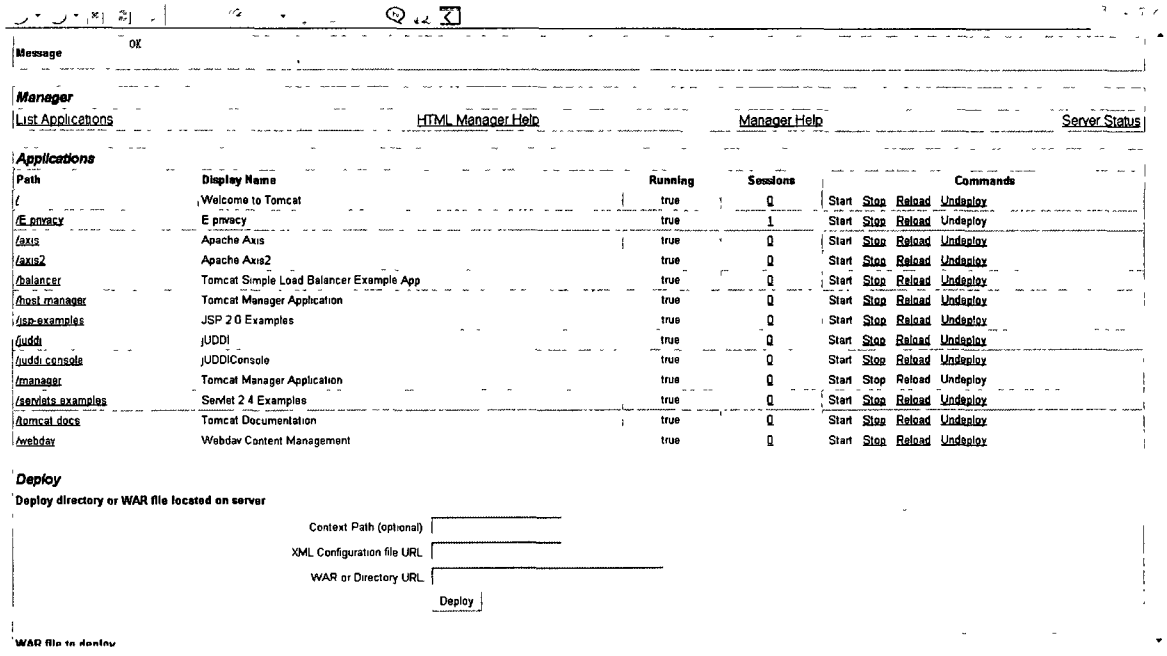


Figure 4.1, Apache Tomcat application server manager page.

Figure 4.1 shows a list of applications deployed on the server. Among them, juddi is the application for JUDDI registry, while E-privacy is the application for the proposed system.

4.2 The Factors for Performance Evaluation

The thesis will evaluate the performance of the implementation and compare with the Semantic E-Workflow Composition approach. As well, it will be compared to the MOACO algorithm for web service discovery. In order to examine the robustness of the algorithms, we ran the experiments against different amount of web service instances starting from 10 to 100, incrementing by 10 each time. The following factors will be considered to evaluate the performance of the system:

(1) Accuracy of results: it can be interpreted in terms of whether the system always

chooses the "best participating web services" in the composition. The "best participating web service" can be defined as: a web service which performs the exact functions as expected with most optimized QoS values among all the candidate web services."

(2) Time to obtain results: it can be defined as the total time the compose agent used to realize service discovering and automatic composition. It can be approximated as following:

Total time ≈ Total time before Matching + Web service Matching Time + Web service Composing Time where

(a) The Total time before matching ≈ time for loading and parsing input + time for searching UDDI + time for initiating Google distance or loading an ontology

And (b) Web service matching Time ≈ Text preprocessing time + time to calculate Q-gram distance + time for semantic selection + time for QoS selection;

This thesis uses text preprocessing to remove the stop words and stem the words in the web service names and web service descriptions so that one can get more accurate syntactic matching results.

In the proposed approach and MOACO approach, we use Google distance algorithm to calculate the semantic similarities. (The authors of MOACO did not specify an approach to use for calculating semantic similarity).

MOACO is a learning based algorithm. We know all the learning based algorithms will have a learning process. They usually have to iterate many times to train the program. For MOACO, we set the iteration limit to 100 and 200, which are used in [Fang, 2009] as well.

In order to obtain more accurate values of the performance metric, we will repeat experiments and present the averages within an acceptable confidence interval.

4.3 JUDDI and web services

Since all the public UDDI registries have been retired, we have to create our own private UDDI registry for us to publish / inquire web services. There are several private UDDI registries available, such as the UDDI services included in Windows Server 2003, and the IBM WebSphere UDDI registry. We finally chose JUDDI because it has the following advantages:

- It is an open source freeware. Easy to set up and be customized.
- Unlike the UDDI services in Windows Server 2003 and IBM WebSphere UDDI registry (they are bundled together with a bunch of Enterprise solutions), JUDDI is light weighted and portable.
- Its implementations of the UDDI standards are very clear and straightforward;
- Support both UDDI v2.0 and v3.0;

4.3.1 Publish to JUDDI registry

JUDDI registry supports the actions listed in the Figure 4.2.

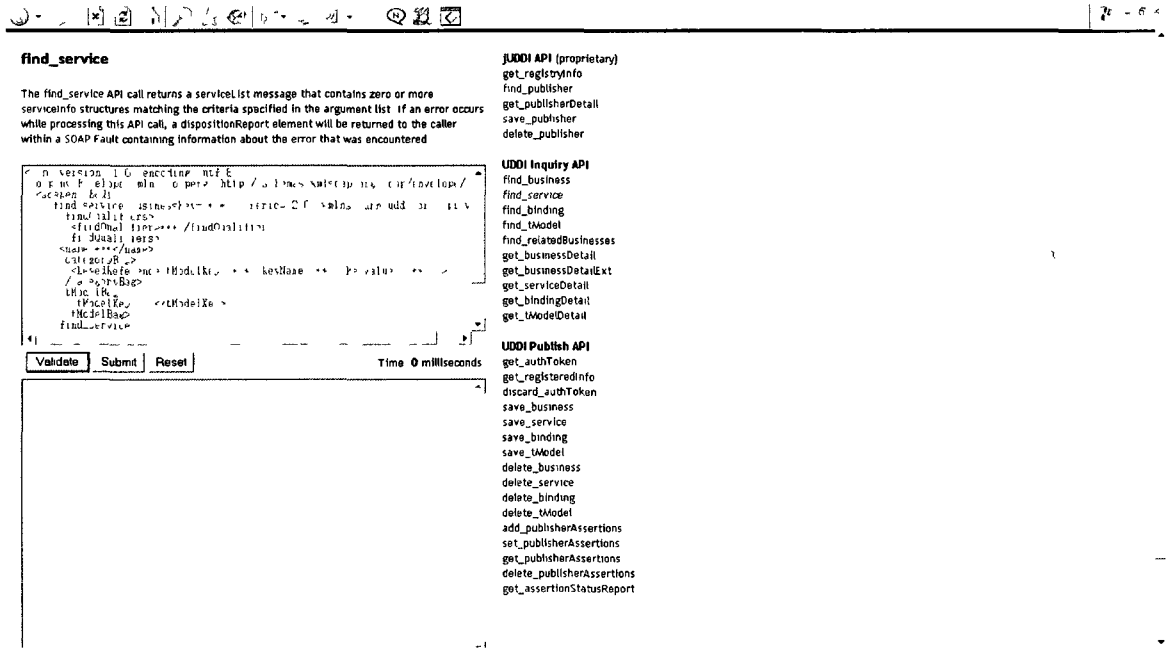


Figure 4.2 screenshot of JUDDI console page.

On the right side of Figure 4.2, we can see a list of all the supported actions by JUDDI. To publish to the JUDDI, we can use the JUDDI console (shown in Figure 4.2); or program a JAVA program and call the UDDI4J APIs to publish to or query the UDDI registry; or the simplest way, to publish / inquiry through UDDI browser. Eclipse IDE offers a WSE (Web Services Explorer) component, by which we can easily publish / inquiry UDDI registry through a graphic user interface. In this thesis, we created and published 100 web services as shown in the Figure 4.3.

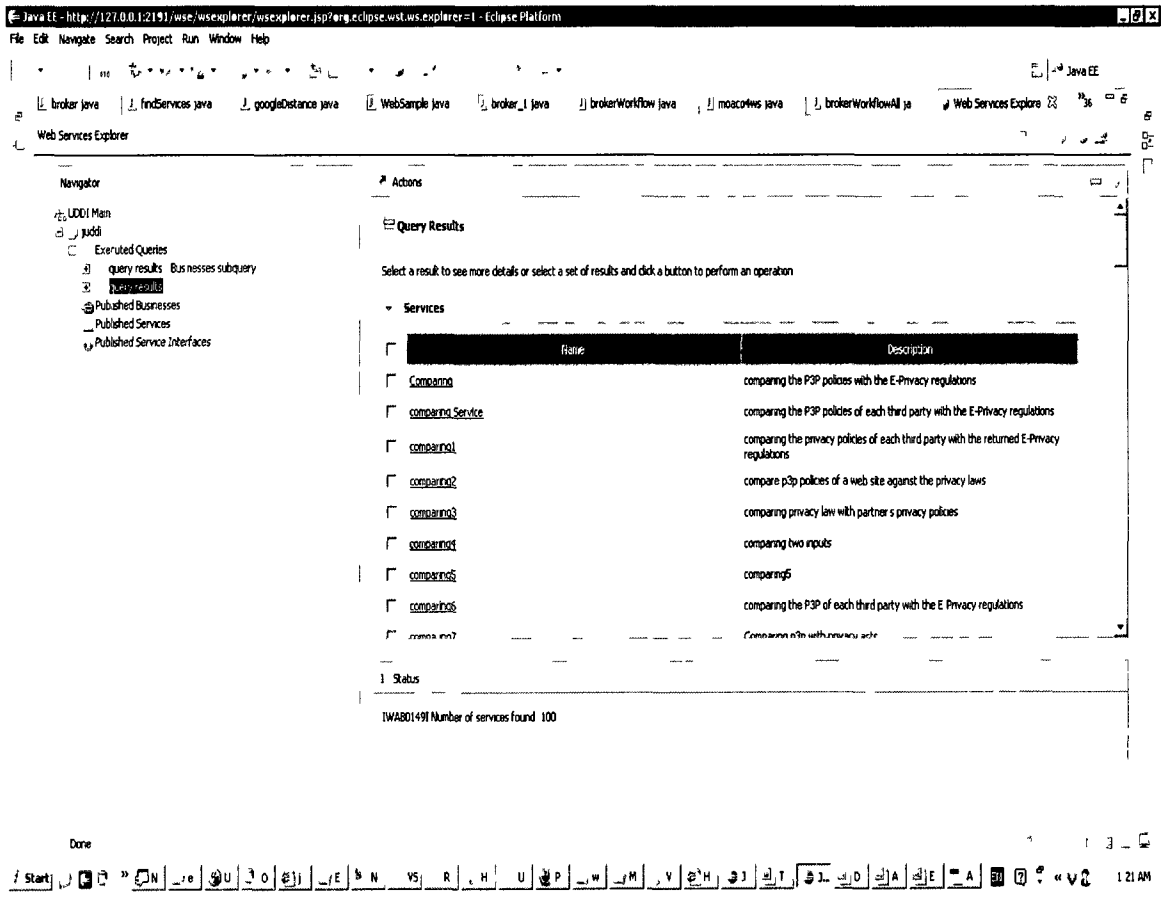


Figure 4.3 100 web services published to the private UDDI registry

4.3.2 Present semantic information and QoS parameters in UDDI registry

As a central directory for publishing and inquiring web services, UDDI must be capable of representing data and Meta data about web services. As well, it should offer a standard mechanism to classify, catalog and manage Web services, so that they can be discovered and consumed. For those purposes, four core data structure types were defined to represent information with UDDI: the businessEntity, the businessService, the bindingTemplate and the tModel. Figure 4.4 shows the relationships between the four core types. Each businessEntity contains 0 or more businessServices. The businessService structure represents the logical services that belong to a single businessEntity. A typical

businessService entity is structured like Figure 4.5. The bindingTemplate entities present a list of technical descriptions (such as tModel instance details, access point) about the businessService. The categoryBag presents a list of categories that each describes a business aspect of the businessService. Users may define their own category classifications by using tModels.

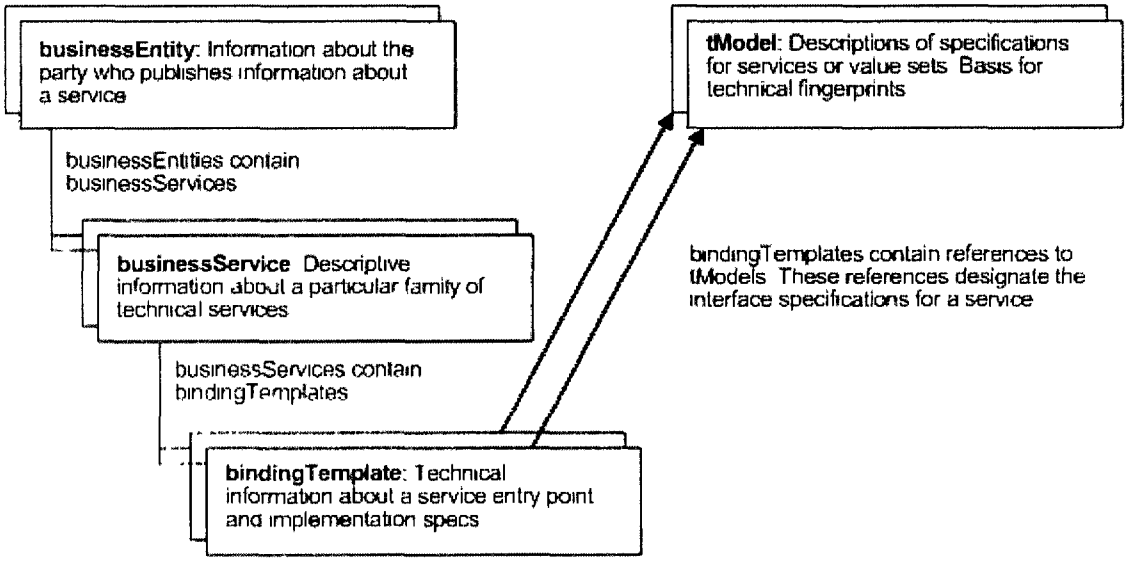


Figure 4.4 UDDI core data structures and their relationships

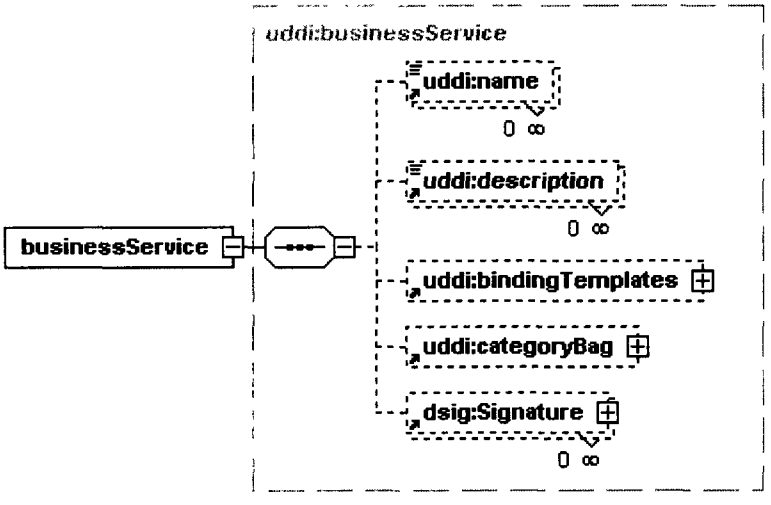


Figure 4.5 Structure Diagram for businessService entity

In order to store the semantic information and QoS parameters in the UDDI registry yet without making any modifications to the existing UDDI standards, in this thesis, we use customized categories to save the semantic information and QoS parameters in UDDI registry. Figure 4.6 shows a web service published in our private UDDI registry and how the semantic information and QoS parameters are presented by using categories.

Language	Description	Actions
	comparing the P3P policies with the E Privacy regulations	Edit

Categories Add Remove Edit Cancel

Type	Key name	Key value	Actions
undefined	input	partner_name, regulation	Edit
undefined	output	recommendations	Edit
undefined	QoS time min	20	Edit
undefined	QoS time avg	30	Edit
undefined	QoS time max	40	Edit
undefined	QoS cost min	10	Edit
undefined	QoS cost avg	10	Edit
undefined	QoS cost max	10	Edit
undefined	QoS reliablity min	0 5	Edit
undefined	QoS reliablity avg	0 7	Edit
undefined	QoS reliablity max	0 9	Edit

Figure 4.6 CategoryBags to store semantic info and QoS parameters in a UDDI registry

The tModel concept is very important in the UDDI world. Typically, a tModel can be used to represent interfaces. The interface could be a standard which will be followed by a group of other entities. Or it could be a contract between service provider and service consumer. Furthermore, a tModel can be used to represent customized category classifications which can then be added into interface tModels to make search easier. Finally, tModels can be used as namespaces to add more meanings into the UDDI data structure.

Specifically, we first created a tModel "QueryIf" as a customized category. Then for each web service published by E-privacy web services providers, they must reference to the tModel "QueryIf" and specify the "Key values" for the following 11 "Key names": input, output, QoS time min, QoS time avg, QoS time max, QoS cost min, QoS cost avg, QoS cost max, QoS reliability min, QoS reliability avg, QoS reliability max. By this means, we can easily store the semantic information (input, output) and QoS parameters in the UDDI registry without making any changes to the UDDI structures.

4.3.3 Find web services

In this thesis, we created a tModel "Eprivacy web services providers" which is used as an interface or category. We assume all the E-privacy web services providers have to use that tModel. And our web service discovering and matching algorithms will only apply to the web services published by those providers. So, all we need to do is to first search the UDDI by using a tModel key to find all the businesses that use that tModel, then get all the web services published by those businesses, and then perform the syntactic matching, semantic matching, and QoS matching on top of those web services. Following is the pseudo code of the findServices function:

Construct a UDDI Proxy object.

Set UDDI registry query URL

Construct a tModel Bag

Construct a tModel Key with the known tModel Key

Add the tModel Key into the tModel Bag

Construct a Find Quantifier

Set Find Quantifier to 'caseSensitiveMatch'

```
Construct a Vector
Add the Find Quantifier into the Vector
Search the UDDI registry by tModel Bag, Find Qualifier Vector and maximum
number of return entries
Get vector of business information
        For (each business information) {
            Search the UDDI registry by Business Key, Find Qualifiers
            Get vector of service information
                For (each service information) {
                    Get service key
                    Get service detail by the service key
                    Get the first business service
                    Get service wsdl URL
                    Get service description
                    Get service category bag
                    For (each category) {
                        If (key name = "input")
                            Input = key value
                        Else if (key name = "output")
                            Output = key value
                        Else if (key name = "QoS time min")
                            TimeMin = key value
                        Else if (key name = "QoS time avg")
```

TimeAvg = key value

Else if (key name = "QoS time max")

TimeMax = key value

Else if (key name = "QoS cost min")

CostMin = key value

Else if (key name = "QoS cost avg")

CostAvg = key value

Else if (key name = "QoS cost max")

CostMax = key value

Else if (key name = "QoS reliability min")

ReliMin = key value

Else if (key name = "QoS reliability avg")

RelitAvg = key value

Else if (key name = "QoS reliability max")

ReliAvg = key value

}

}

}

Create a new SO object by using service key, service name, service description, input, output, timeMin, timeAvg, timeMax, costMin, costAvg, costMax, reliMin, reliAvg, reliMax

Add newly created SO object to a list of SO objects

```
<?xml version="1.0" encoding="UTF-8"?>
<process id="1" type="sequence" >
 <process id="2" type="parallel" >
  <ST id = "1">
    <Name>Ontology Query 1 </Name>
    <Description>get the related e-privacy regulations through ontology query</Description>
    <Input>Ontology Query</Input>
    <Output>E-Privacy regulations</Output>
    <QoS>
        <Time>50</Time>
        <Cost>30</Cost>
        <Reliability>0.5</Reliability>
    </QoS>
  </ST>
  <ST id = "2">
    <Name>Partners</Name>
    <Description>Return the third party list of the input web site</Description>
    <Input>web site</Input>
    <Output>third party list</Output>
    <QoS>
        <Time>50</Time>
        <Cost>20</Cost>
        <Reliability>0.5</Reliability>
    </QoS>
  </ST>
</process>
  <ST id = "3">
    <Name>Comparing service</Name>
    <Description>comparing the P3P policies of each third party with the E-Privacy
regulations</Description>
    <Input>partner name, e-privacy regulation</Input>
    <Output>comparing results</Output>
    <QoS>
        <Time>50</Time>
        <Cost>50</Cost>
        <Reliability>0.5</Reliability>
    </QoS>
  </ST>
</process>
```

Figure 4.7 Sample Input File

4.4 Load and parse the input

The input to the compose agent includes the workflow and the web service templates.

They are specified in a XML file. Figure 4.7 is a sample input file used in the experiments:

The XML tags used in the above XML file are self-explanatory. Each process node specifies a process; each ST node specifies a service template; while each QoS node specifies a QoS matrix. The compose agent will load the above XML file, parse it and finally get a list of ST objects. Following is the pseudo code for loading and parsing input file:

```
Construct a DocumentBuilderFactory object
Construct a DocumentBuilder object
Parse the input XML file into a document object
Get document element (root element)
Get a node list of ST elements
   For (each node in the list) {
         Get the ST element
         Parse the following values from ST element: id, Name, Description, Input,
Output, Time, Cost, and Reliability
         Create a new ST object by using those values
         Add the newly created ST object to a list of ST object
   }
```

4.5 Implementation of the proposed service matching method

In **4.3.3** (find web services), we searched the UDDI, got all the available web service instances, and transformed them into a list of SO objects. In **4.4** (load and parse input), we read in the input xml file, parse the file, and finally get a list of ST objects. The service matching process basically compares each SO object against each ST object in terms of the syntactic similarity, semantic similarity, and QoS similarity to find the best matching

SO object for each ST object. We compare the syntactic similarity first, then semantic similarity, and finally the QoS similarity. To improve the performance, we set the threshold for syntactic similarity scores and semantic similarity scores. Only when a SO's similarity score is less than the threshold, the SO can move to the next matching step. Following is the pseudo code for this matching process:

```
Construct a Stemmer object
Construct a delete_words object
Construct a editDistance object
Construct a qGram object
Construct a googleDistance object
Construct a qosDistance object
Set threshold
For (each ST) {
    Stem the ST name
    Remove the stop words from ST description and stemming
    For (each SO) {
            Stemming the SO name
            Remove the stop words from SO description and stemming
            Calculate Q-gram distance for ST name and SO name
            Calculate Q-gram distance for ST description and SO description
            Get the syntactic similarity score
            If (the syntactic similarity score < threshold) {
                    Calculate Google distance for ST input and SO input
```

```
                    Calculate Google distance for ST output and SO output
                    Get the semantic similarity score
                    If (the semantic similarity score < threshold) {
                        Calculate QoS distance by using proposed formula
                        If (the QoS score > 0 AND the QoS score < saved best
score for the current ST ) {
                        Save the QoS score as the best score for the current ST
                    Save the current SO as the best matching SO for the current ST
                        }
                    }
                }
            }
        }
```

4.6 Implementation of the E-Workflow composition method;

The E-Workflow composition method basically follows the same matching process. Although it is not mentioned in the original paper, to improve the performance and make the results more comparable, we set the threshold for syntactic similarity scores and semantic similarity scores as well.

Different to our proposed approach, which only set the acceptable values, the E-Workflow composition method set minimum, average, maximum values for each QoS parameter in a ST. As shown in the following XML code fragment, the ST node in the input XML file for the E-Workflow method is different as well.

```
<ST id = "1">
  <Name>Ontology Query</Name>
  <Description>get the related e-privacy regulations through ontology query</Description>
  <Input>Ontology Query</Input>
  <Output>E-Privacy regulations</Output>
  <QoS>
      <TimeMin>10</TimeMin>
      <CostMin>30</CostMin>
      <ReliabilityMin>0.5</ReliabilityMin>
      <TimeAvg>30</TimeAvg>
      <CostAvg>30</CostAvg>
      <ReliabilityAvg>0.8</ReliabilityAvg>
      <TimeMax>50</TimeMax>
      <CostMax>30</CostMax>
      <ReliabilityMax>1</ReliabilityMax>
  </QoS>
</ST>
```

The E-Workflow method also uses the Q-Gram algorithm to calculate similarities for both service names and service descriptions. Instead of using the Google distance algorithm, the E-Workflow method calculates semantic distances for service inputs and outputs through use of a feature-based algorithm. In order to make the feature-based algorithm workable, a pre-defined domain ontology is required. For experimental purposes, we used two test domain ontologies. The first is represented in a fast data structure, a chained hash table, with few terms. It contains 41 terms and each term has 5 properties. The usage of this first test domain ontology is to help isolate the overhead due to ontology reasoning engines. The second ontology is the classic wine ontology (http://krono.act.uji.es/Links/ontologies/wine.owl) which is represented in an OWL file and processed by the ontology reasoning engine, Jena. Figure 4.8 shows a fragment of wine.owl. Experiments will examine the performance of the E-Workflow approach and the overhead of using a semantic approach to represent the ontology versus a prescriptive approach using an in-memory data structure. Jena 2.6.4 OWL APIs are used to load, parse,

and query the OWL file The Pellet OWL Reasoner is used together with Jena to perform ontology reasoning over the wine ontology.

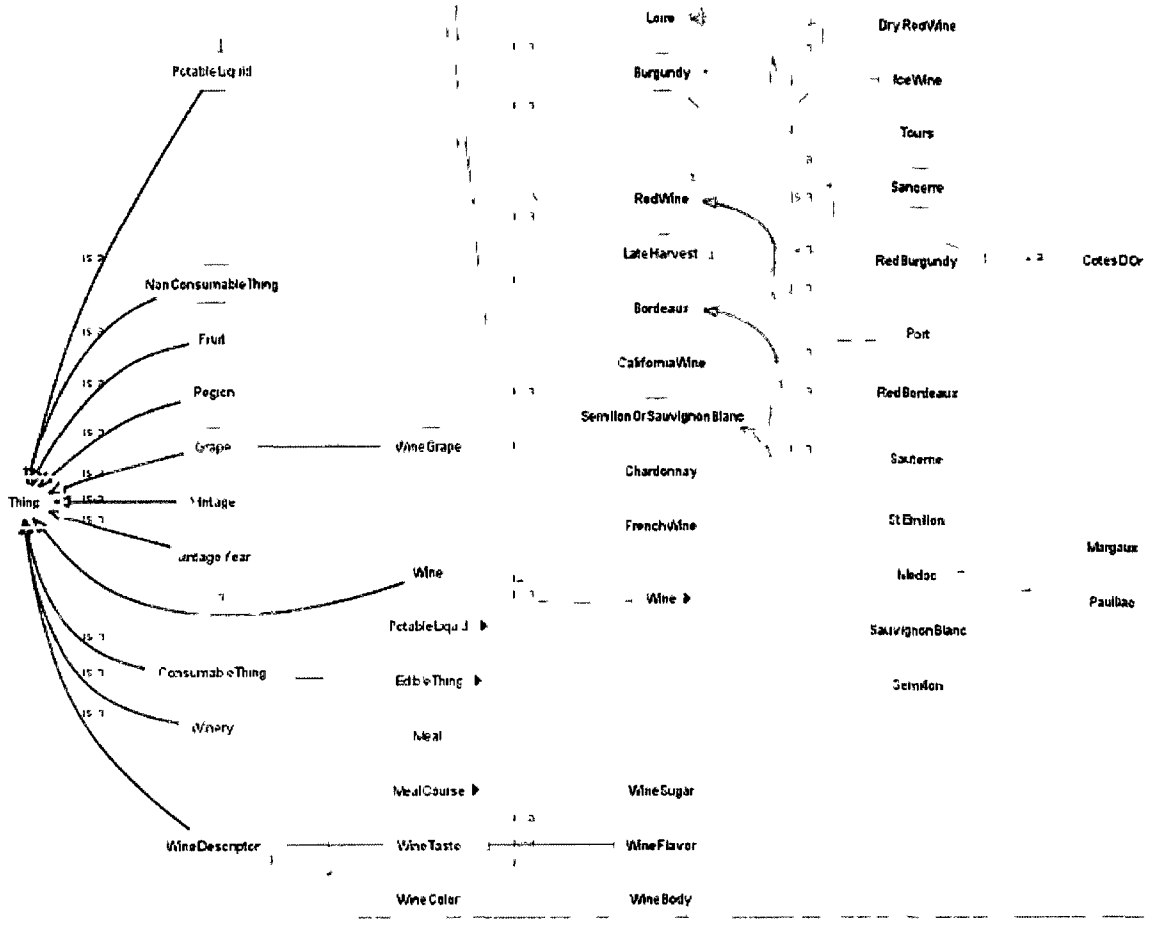


Figure 4.8 Fragment of wine ontology

To make use of the feature-based algorithm, we have to realize the functionality to get the properties and the value of the properties for a term in the ontology. To realize this, we first use the Pellet reasoner to filter out the unwanted global properties which will pollute the ontology query results. When we compute the distance of two terms in the ontology, we try to get all the declared properties for each term, if there is no property defined for the term, then we get the shortest path between the two terms instead of using the feature-based algorithm. If there are properties defined for both terms, then we use a

recursive function to get the value for each property. Due to the complexity of the OWL language, we need to consider several different situations during this simple process. For example, the queried term might be defined as an intersection class of two other terms. In that case, we might need to get the property values from those two terms as well. The details of this ontology reasoning implementation can be found in the source code attached to this Thesis. After this process, we can get results such as the following:

property for the term RedBurgundy: PinotNoirGrape
property for the term RedBurgundy: Red
property for the term RedBurgundy: Winery
property for the term RedBurgundy: hasWineDescriptor
property for the term RedBurgundy: Dry
property for the term RedBurgundy: hasFlavor
property for the term RedBurgundy: hasBody
property for the term RedBurgundy: producesWine
property for the term RedBurgundy: BourgogneRegion
property for the term RedBurgundy: madeFromFruit

property for the term Burgundy: madeFromGrape
property for the term Burgundy: hasColor
property for the term Burgundy: Winery
property for the term Burgundy: hasWineDescriptor
property for the term Burgundy: Dry
property for the term Burgundy: hasFlavor
property for the term Burgundy: hasBody
property for the term Burgundy: producesWine
property for the term Burgundy: BourgogneRegion
property for the term Burgundy: madeFromFruit

The Feature-based algorithm is performed based on those properties. The similarity score for the two terms RedBurgundy and Burgundy in the above example is 7.302967433402214, which indicates a relatively high similarity.

Another key difference between the E-Workflow composition and our proposed FOIQOS approach is the way to calculate the QoS distance. Recall the E-Workflow method calculates QoS distance by comparing a ST's minimum QoS values with a SO's minimum QoS values, a ST's average QoS values with a SO's average QoS values, and a

ST's maximum QoS values with a SO's maximum QoS values. It intends to get a service object, whose minimum, average, maximum QoS values are closest to the minimum, average, maximum QoS values set for a ST, as the best matching SO for that ST. As well, it only considers absolute distance whereas FOIQOS considers relative distance. E-Workflow then cannot guarantee the best choice selection which we instrument as accuracy.

4.7 Implementing the MOACO algorithm

Before applying MOACO approach to perform web service discovering / composition, there are some preprocessing tasks.

- Predefine composition workflow and web service types involved in the workflow;
- For each web service instance, categorize it to a web service type according to the syntactic and semantic similarities between the web service instance and web service type;

Since in [Fang et al, 2009], the authors did not mention what methods they used to categorize each web service instance into web service types, in this thesis, we simply used the same inputs and same syntactic and semantic matching methods used by the proposed approach to realize those preprocessing tasks for the MOACO approach.

After the preprocessing, based on the composition workflow and categorized web service instances, the MOACO approach constructs a weighted directed graph, where each web service instance is a node in the graph. The direction of the graph follows the direction defined in the workflow. Two virtual nodes, S and T, are added to the graph to represent the beginning vertex and target vertex. The weight of each node is determined

by the QoS values of the web service instance. The MOACO approach then uses a colony of ants to construct possible paths from S to T. At every generation (iteration), the known Pareto Front P_{know} [Van Veldhuizen, 1999] is updated and finally the pheromone matrix IPSi is updated as well. Figure 4.9 shows the pseudo code for the MOACO algorithm:

```
Set global QoS constraints for time, cost, and reliability
Set generation limit (number of iterations)
Set colony size (number of ants)
Initiate Pknow, IPSi, and construct the directed graph based on the inputted
workflow and web service instances
While (iteration number less than iteration limit) {
   For each (ant in the colony) {
         Build paths (see figure 4 10 for pseudo code)
         For each (path found in Build paths step) {
               If (global QoS values meet the constraints) {
                     If (current path not in Pknow) {
                           Add the current path to Pknow
                     }
               }
         }
   }
   Iteration number + 1;
   Update IPSi (see figure 4.8 for pseudo code)
}
```

Figure 4.9 pseudo code for MOACO algorithm

Depending on the state of P_{know}, if new paths were added to P_{know} then the pheromone matrix is reset to the initial values to improve exploration. Otherwise, the method updates the pheromone matrix with the following formula to get a better exploitation. See Figure

4.10. for pseudo code for the global update of ISPi.

$$ISPi = (1-\rho) \times IPS_0 + \rho \times \Delta IPS,\ \rho \in [0,1] \qquad \text{Formula 4.1,}$$

Where ΔIPS is calculated by using the formula 4.2:

$$\Delta IPS = 1/\sum_{\forall P \in P_{know}} \sqrt{Cost^2(P) + Time^2(P) + (1/\text{Re}liability(P))^2} \text{ , Formula 4.2}$$

```
Initiate Delta IPSi;
For each (P in Pknow) {
    Get the QoS values for P
    Get the delta value for P by using formula 4.2
    Update IPSi value for P by using formula 4.1
}
```

Figure 4.10 pseudo code for global update of IPSi

```
While (path length < workflow length) {
    Get the heuristic values for all the nodes in the current level
    Set the probability of being chosen for each node by using the formula 4.3
and 4.4
    Get a random number
    Select the node by comparing the probabilities of each node with that
random number
}
```

Figure 4.11 Pseudo code for building a path (referenced in Fig. 4.9.)

Each path is a possible composition solution that starts from the virtual beginning vertex S, follows the workflow direction, and finally reaches the target virtual vertex T. When building a path, the following formula 4.3 is used to set the probability of being chosen for each node.

$$P_{ij} = \frac{[IPS_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{s \subset N_i}[IPS_{is}]^{\alpha}[\eta_{is}]^{\beta}}$$ Formula 4.3, where Ni is the node in the neighbourhood

of node I that the ant has not visited yet, η_{ij} is the heuristic value of moving from node i to node j. η_{ij} is calculated by using the formula 4.4:

$$\eta_{ij} = 1/\sqrt{Cost^2(j) + Time^2(j) + (1/\mathrm{Re}liability(j))^2}$$ Formula 4.4

4.8 Implementation of dynamic web service composition

To realize the dynamic web service composition, we need to know the composition workflow, the original inputs, and the best matching web service objects for each ST (Service Template) in the workflow. Through the matching process, we obtain the best matching web service objects for each ST. The composition workflow is defined in the input XML file and is loaded into a document object. The original inputs are stored in a string list object. (The whole workflow can be treated as a composite web service. The original inputs are actually the inputs of that composite web service.) The compose agent will take all the above objects as input parameters and use a recursive function to go through the composition workflow, pass by the inputs / outputs, invoke the matching service objects, and return the final output.

The following is the pseudo code for the recursive function used in dynamic web

service composition to control the composition process. This recursive function takes a Node object, the matching service objects, and a list of the input parameters as inputs. Depends on the workflow, the outputs of the function may be used as the inputs of the next step:

```
If (the type of the input Node == ELEMENT_NODE) {
    Get all the attributes of the node;
    For (each attribute) {
        If (Node name == "ST" AND Attribute name == "id") {
            Get the matching service object for this ST;
            Invoke the matching service object with the input parameters
            Return the outputs;
        } else if (Node name == "process" AND Attribute name == "type") {
            Get the process type (sequence / parallel)
        }
    Get all the child nodes of the input Node;
    For (each child node) {
        If (process type == "sequence" AND child node name == "ST") {
            Clear the original inputs
            Treat the outputs from the previous step as the new inputs
            Call the recursive function itself by taking the child node, matching
service objects, and the new inputs parameters as inputs
        } else if (process type == "parallel" AND child node name =="ST") {
            Call the recursive function itself by taking the child node, matching
```

service objects, and the inputs parameters as inputs

Add the outputs of the above call to the overall outputs of this step

} else {

Call the recursive function itself by taking the child node, matching service objects, and the inputs parameters as inputs

}

}

}

Another difficult part in the dynamic web service composition is how to invoke a web service object dynamically. Because all the service objects are discovered and selected during the run time, we do not know the actual web service name, method name, and we have no idea about the parameters at the design time. Fortunately, during the web service selection process, when we get the matching service objects, we've already obtained the WSDL urls for each matching service objects. With a WSDL URL, we can get the WSDL file for a web service. A WSDL file is written in XML format. It usually describes a web service by using following elements:

- Service: contains a set of related port / endpoints. An port / endpoint defines the address / URL for invoking the service;
- Binding: specifies how the service will be implemented, defines the communication protocols and data format specification for a port type;
- PortType: an interface which defines a set of operations performed by endpoints;
- Message: contains the information needed to perform the operation. It can serve

as the input or output of an operation;

- Types: defines the data types used by the web service. The type definitions are usually referenced from higher-level message definitions.
- Operation: defines the SOAP actions. It is similar to a method or function call in a traditional programming language.

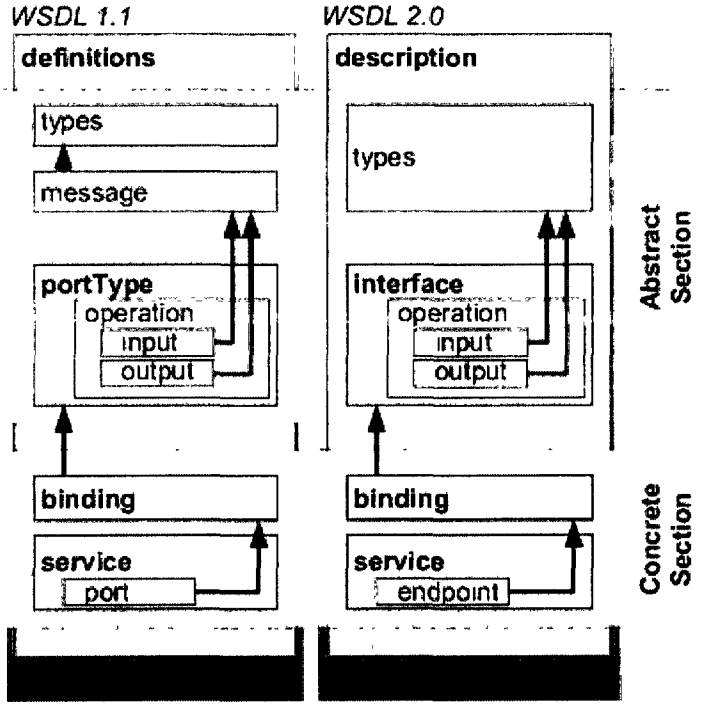


Figure 4.12 Structure diagram for WSDL

http://en.wikipedia.org/wiki/Web_Services_Description_Language

The WSDL file can be created manually or by using special tools like Java2WSDL, A WSDL file can be generated automatically through existing web services. After we get a WSDL file, we can parse the file and get all the elements mentioned above. With those elements, we can easily get the service name, method name, invoke URL, parameters, and all the information we need for dynamic invocation of web services. To parse WSDL files,

we employed AXIS APIs from Apache. Apache Axis is an implementation of SOAP. It provides extensive support for the WSDL. Following is the pseudo code for the dynamic invocation process:

Construct a WSDL Parser object

Parse the WSDL file for the input wsdlURL

Get all the element entries

Get service entry for the input service name

Get the service object

Get all the ports specified in the service object

Get the end point reference (invocation URL)

Construct a clientService object

Create a new service call

Set the end point reference value and operation name for the created call

Get the binding entry for this service and this port

Get the parameters contained in the binding entry

Assign input values to the input parameters

Invoke the service and assign the outputs to output parameter

4.9 Addressing Global QoS Constraints

To compare with the MOACO approach, which can find possible composition solutions and was evaluated by using the number of the solutions found, we also created a modification of the proposed FOIQOS algorithm to search for all the possible solutions based on the global QoS constraints. The basic idea of our algorithm is simply a graph walk through. We use the same inputs as MOACO approach, which can be modeled as a

kind of a directed graph. Each web service type in our composition workflow is modeled as a level in the directed graph, and each web service instance is modeled as a node. As shown in Figure 4.13, each service type is a level in the graph. If there are n service types involved in the composition workflow, then there will be n levels in the graph. We mark each level as WS_{Tn}. Each service type could have m service instances. Each service instance is a node in the graph. We mark each service instance as $WS^{Tn}{}_{Im}$. The direction of the graph just follows the direction in the composition workflow. Each node in the previous level can connect to all the nodes in the next level. So ideally if there are n levels and each level has m nodes, then there could be m^n paths connecting from the first level to the last level. However, because of the global QoS constraints, the composition solutions should be much less then m^n. For each path, one needs to add up the QoS values of each node contained in a path to see if it meets the specified global constraints. Only the paths whose total QoS values are less than the specified global QoS constraints can be treated as a composition solution.

I designed a recursive function to walk through the nodes in a level in the directed graph. For each node, we add up the QoS values, if the current total QoS values of the path did not exceed the constraints, then move to the next level. If the total QoS values exceed the pre-defined constraints, the modified method will discard the current path and move to the next node in the same level. By this means, time is not wasted to walk through the whole graph and we can still guarantee to find all the possible paths to accommodate the pre-defined QoS constraints.

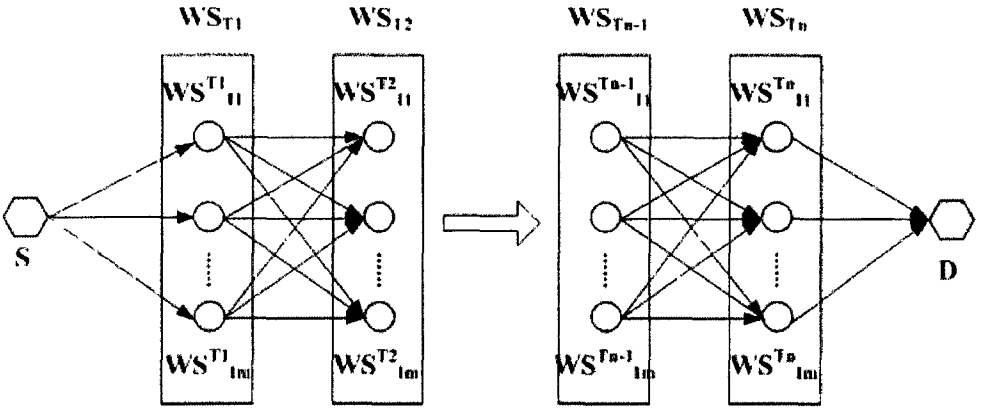


Figure 4.13 Web service instance selection graph [Fang, 2009]

The following is the pseudo code for the recursive function mentioned above:

Read in the temporary total QoS values for the current temporary path

Save a copy of the temporary total QoS values for the current temporary path

For (each node in the current level) {

Add up the QoS values to the temporary total QoS values

If (the temporary total QoS values <= global QoS constrains) {

Connect the node to the current temporary path

If (this is the last level) {

Add the temporary path to the solution set

} else {

Call the recursive function itself to go the next level

}

}

```
    Assign back the temporary total QoS values from the saved copy

    Move to the next node in this level

}
```

Chapter 5

Results Evaluation

In this chapter, we describe the experiments and evaluate the experimental results. The experiments are overviewed in section 5.1.

5.1 Experiments

In order to evaluate and compare the performance of the FOIQOS approach, we implemented and deployed the FOIQOS, E-Workflow, and MOACO approaches in the same test environment. While the FOIQOS and E-Workflow approaches aimed to find a best selection and composition of web services, the original MOACO approach aims to find the possible composition solutions. I decided to extend and implement the FOIQOS approach to find all the possible composition solutions, resulting in five approaches and comparisons: FOIQOS (one solution), E-Workflow (one solution), MOACO (one solution), FOIQOS (many possible solutions), and MOACO (many possible solutions).

For each approach, I ran the experiments against differing numbers of web service instances starting from 10 to 100, incrementing by 10 each time. As well, I repeated each experiment for 10 times and present the average to obtain more accurate values of the performance metric.

The experiment results for all the 5 implemented approaches, including the FOIQOS approach, E-Workflow approach, and MOACO-based approach modified to obtain one selection, FOIQOS approach for finding all the possible solutions, and the MOACO approach for finding possible solutions, will be compared together in terms of the execution time and the accuracy of the results.

A typical raw result of an experiment run is shown in Figure 5.1. It provides the summarized results such as total execution times for each step, the best matching web service objects for each web service template, and the inputs / outputs of the composition process. The designed program is capable of displaying the execution time for each detailed sub-step, such as stemming, stop words removal, etc., during the whole discovering and composition process.

5.2 Results

Figure 5.2 shows the spreadsheet we used to store and compare the experiment results for the experiments running against certain number of web service instances. The top part of the spreadsheet shows the total execution times for running each approach. The rest of the spreadsheet is divided into 5 parts and each part shows more detailed execution times for each major step in an approach and the quality of the approach, e.g. final matching web service instances, or the total solutions found. To make the experiment results more reliable, we repeat each experiment for 10 times and present the average values for comparison.

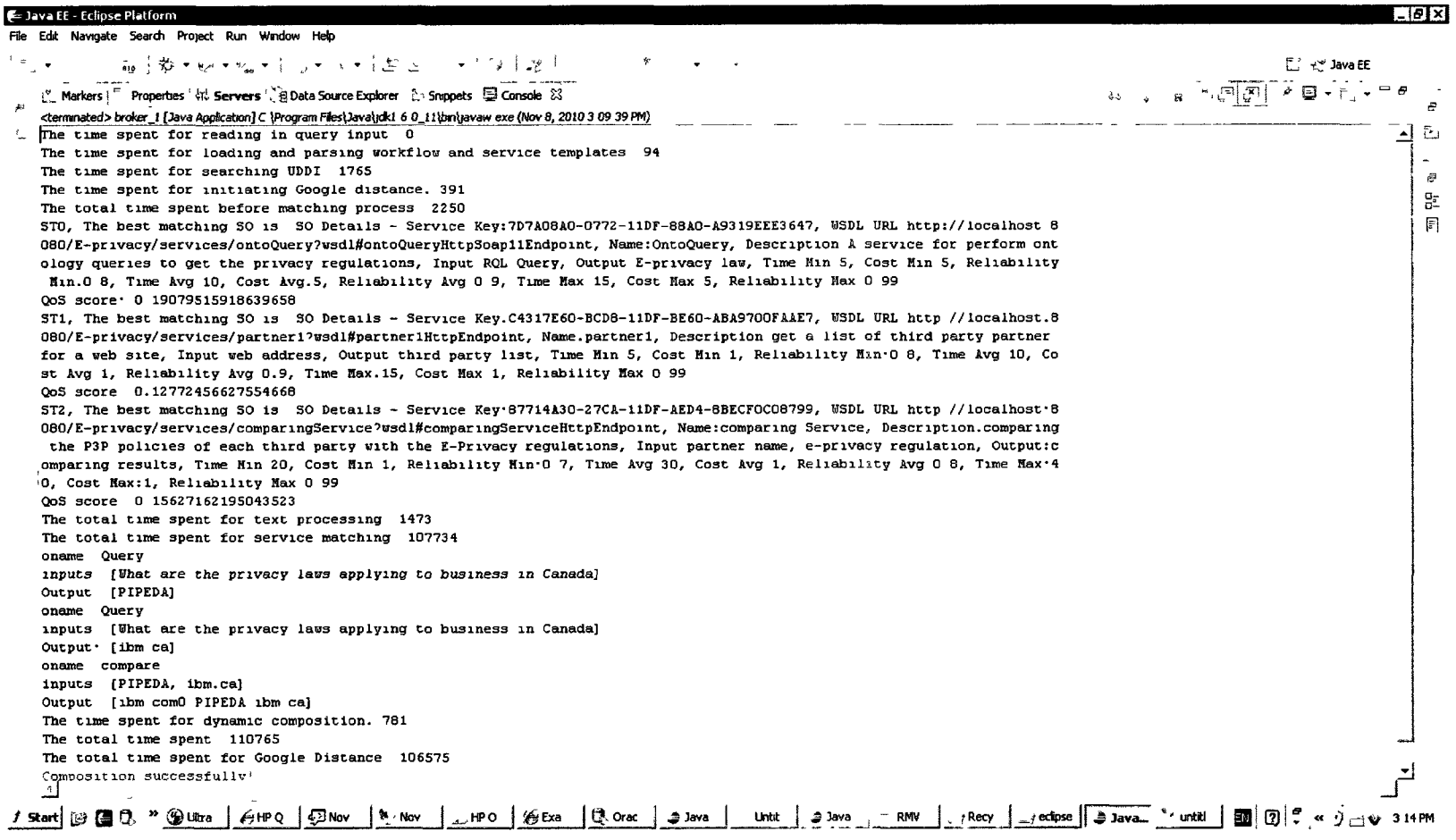
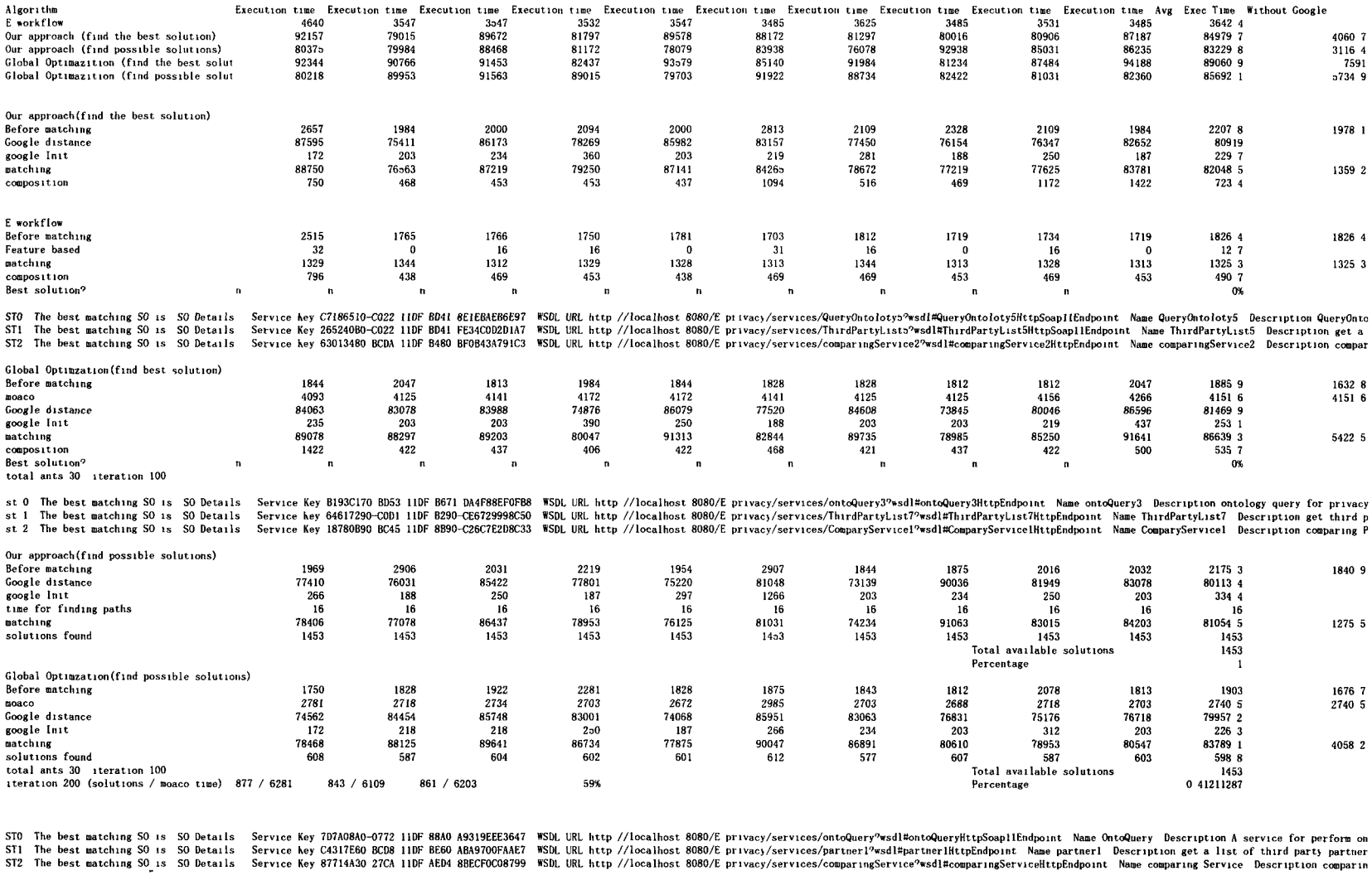


Figure 5.1 a typical raw result of an experiment run



Algorithm	Execution time	Execution time	Execution time	Execution time	Execution time	Execution time	Execution time	Execution time	Execution time	Execution time	Avg	Exec Time	Without Google
E workflow	4640	3547	3547	3532	3547	3485	3625	3485	3531	3485		3642 4	
Our approach (find the best solution)	92157	79015	89672	81797	89578	88172	81297	80016	80906	87187		84979 7	4060 7
Our approach (find possible solutions)	80375	79984	88468	81172	78079	83938	76078	92938	85031	86235		83229 8	3116 4
Global Optimazition (find the best solut	92344	90766	91453	82437	93579	85140	91984	81234	87484	94188		89060 9	7591
Global Optimazition (find possible solut	80218	89953	91563	89015	79703	91922	88734	82422	81031	82360		85692 1	5734 9
Our approach(find the best solution)													
Before matching	2657	1984	2000	2094	2000	2813	2109	2328	2109	1984		2207 8	1978 1
Google distance	87595	75411	86173	78269	85982	83157	77450	76154	76347	82652		80919	
google Init	172	203	234	360	203	219	281	188	250	187		229 7	
matching	88750	76563	87219	79250	87141	84265	78672	77219	77625	83781		82048 5	1359 2
composition	750	468	453	453	437	1094	516	469	1172	1422		723 4	
E workflow													
Before matching	2515	1765	1766	1750	1781	1703	1812	1719	1734	1719		1826 4	1826 4
Feature based	32	0	16	16	0	31	16	0	16	0		12 7	
matching	1329	1344	1312	1329	1328	1313	1344	1313	1328	1313		1325 3	1325 3
composition	796	438	469	453	438	469	469	453	469	453		490 7	
Best solution?	n	n	n	n	n	n	n	n	n	n		0%	
ST0 The best matching SO is SO Details	Service key C7186510-C022 11DF BD41 8E1EBAEB6E97	WSDL URL http //localhost 8080/E privacy/services/QueryOntology5?wsdl#QueryOntology5HttpSoap11Endpoint	Name QueryOntology5	Description QueryOnto									
ST1 The best matching SO is SO Details	Service Key 26524080-C022 11DF BD41 FE34C0D2D1A7	WSDL URL http 8080/E privacy/services/ThirdPartyList5?wsdl#ThirdPartyList5HttpSoap11Endpoint	Name ThirdPartyList5	Description get a									
ST2 The best matching SO is SO Details	Service key 63013480 BCDA 11DF B480 BF0B43A791C3	WSDL URL http 8080/E privacy/services/comparingService2?wsdl#comparingService2HttpEndpoint	Name comparingService2	Description compar									
Global Optimzation(find best solution)													
Before matching	1844	2047	1813	1984	1844	1828	1828	1812	1812	2047		1885 9	1632 8
moaco	4093	4125	4141	4172	4172	4141	4125	4125	4156	4266		4151 6	4151 6
Google distance	84063	83078	83988	74876	86079	77520	84608	73845	80046	86596		81469 9	
google Init	235	203	203	390	250	188	203	203	219	437		253 1	
matching	89078	88297	89203	80047	91313	82844	89735	78985	85250	91641		86639 3	5422 5
composition	1422	422	437	406	422	468	421	437	422	500		535 7	
Best solution?	n	n	n	n	n	n	n	n	n	n		0%	
total ants 30 iteration 100													
st 0 The best matching SO is SO Details	Service Key B193C170 BD53 11DF B671 DA4F88EF0FB8	WSDL URL http //localhost 8080/E privacy/services/ontoQuery3?wsdl#ontoQuery3HttpEndpoint	Name ontoQuery3	Description ontology query for privacy									
st 1 The best matching SO is SO Details	Service key 64617290-C0D1 11DF B290-CE6729998C50	WSDL URL http 8080/E privacy/services/ThirdPartyList7?wsdl#ThirdPartyList7HttpEndpoint	Name ThirdPartyList7	Description get third p									
st 2 The best matching SO is SO Details	Service Key 18780B90 BC45 11DF 8B90-C26C7E2D8C33	WSDL URL http 8080/E privacy/services/ComparyService1?wsdl#ComparyService1HttpEndpoint	Name ComparyService1	Description comparing P									
Our approach(find possible solutions)													
Before matching	1969	2906	2031	2219	1954	2907	1844	1875	2016	2032		2175 3	1840 9
Google distance	77410	76031	85422	77801	75220	81048	73139	90036	81949	83078		80113 4	
google Init	266	188	250	187	297	1266	203	234	250	203		334 4	
time for finding paths	16	16	16	16	16	16	16	16	16	16		16	
matching	78406	77078	86437	78953	76125	81031	74234	91063	83015	84203		81054 5	1275 5
solutions found	1453	1453	1453	1453	1453	1453	1453	1453	1453	1453		1453	
									Total available solutions			1453	
									Percentage			1	
Global Optimzation(find possible solutions)													
Before matching	1750	1828	1922	2281	1828	1875	1843	1812	2078	1813		1903	1676 7
moaco	2781	2718	2734	2703	2672	2985	2703	2688	2718	2703		2740 5	2740 5
Google distance	74562	84454	85748	83001	74068	85951	83063	76831	75176	76718		79957 2	
google Init	172	218	218	250	187	266	234	203	312	203		226 3	
matching	78468	88125	89641	86734	77875	90047	86891	80610	78953	80547		83789 1	4058 2
solutions found	608	587	604	602	601	612	577	607	587	603		598 8	
total ants 30 iteration 100									Total available solutions			1453	
iteration 200 (solutions / moaco time)	877 / 6281	843 / 6109	861 / 6203	59%					Percentage			0 41211287	
ST0 The best matching SO is SO Details	Service Key 7D7A08A0-0772 11DF 88A0 A9319EEE3647	WSDL URL http 8080/E privacy/services/ontoQuery?wsdl#ontoQueryHttpSoap11Endpoint	Name OntoQuery	Description A service for perform on									
ST1 The best matching SO is SO Details	Service key C4317E60 BCD8 11DF BE60 ABA9700FAAE7	WSDL URL http 8080/E privacy/services/partner1?wsdl#partner1HttpEndpoint	Name partner1	Description get a list of third party partner									
ST2 The best matching SO is SO Details	Service Key 87714A30 27CA 11DF AED4 8BECF0C08799	WSDL URL http 8080/E privacy/services/comparingService?wsdl#comparingServiceHttpEndpoint	Name comparing Service	Description comparin									

Figure 5.2 The results comparison form for the experiments running against 90 web service instances

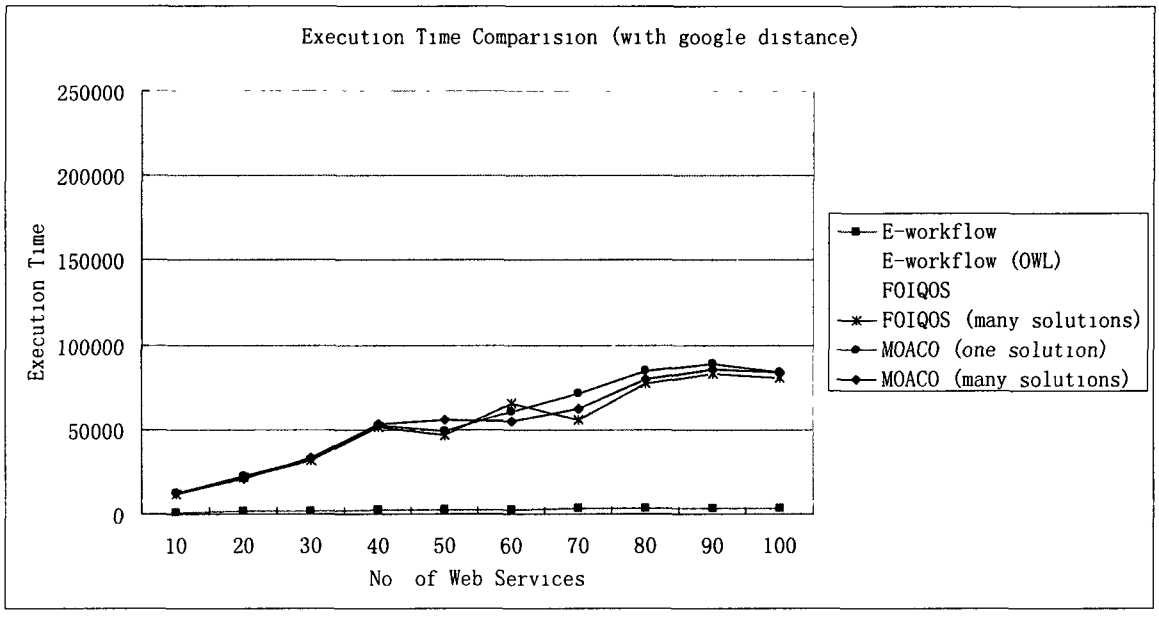


Figure 5.3 Total execution time comparison results.

Figure 5.3 shows the comparative results of the total execution times for all the 5 approaches. We tested the E-Workflow approach on both 41 term text-based sample domain ontology and the classic OWL based wine ontology. In the Figure 5.3, legend "E-workflow (OWL)" indicates the test result of E-Workflow approach on the wine ontology. The E-Workflow approach performs better than all other 4 approaches in terms of the execution time when the ontology is represented in a fast data structure, and without the use of a reasoning engine. However, when the E-Workflow approach used a small OWL ontology like the wine ontology, the performance dropped significantly. It performed even worse when the number of web services is more than 50. The wine ontology contains 2008 entities in total, including classes, properties, individuals, and values. While in the real world, a well-maintained domain ontology would contain far more than ten thousand entities. For example, a well known biomedical ontology, the Mesh ontology, contains

57985 entities in total. In our tests, it took 297 milliseconds to load the wine ontology locally; while it took 14801 milliseconds to load Mesh ontology locally. Furthermore, to load and query a domain ontology requires a lot of resources. In our test, we can load the Mesh ontology successfully, but we cannot complete a simple property query on it because it requires more memory exceeding the capacity of our test environment (3 GB memory). In the real world, for a large ontology, one may have to store the ontology persistently locally to improve the performance. However, a common way to use an ontology is to load it remotely through the internet. This provides a practical way for the ontology owners to maintain their ontology and consumers access the most up to date version. However, to load an ontology remotely would increase the time spent to load the ontology. In our tests, the wine ontology took about 2032 milliseconds to load remotely versus 297 milliseconds to load locally. The Mesh ontology took 125688 milliseconds to load remotely versus 14801 milliseconds to load locally. Thus, network delays increase the overhead of an ontology-based approach as well.

The proposed FOIQOS approach performs slightly better than the MOACO approach. As we specified in the previous chapters, the proposed approach and MOACO approach were implemented to use Google distance to calculate the semantic similarities. The Google distance algorithm needs to invoke search APIs provided by search engine service providers such as Google, Bing, or Yahoo, to get the web search results, and calculate the semantic similarity scores based on the returned search results and the Bayes' theorem. The performance of the Google distance algorithm is highly dependent on the performance of those search APIs, which is time consuming comparing to other steps involved in the whole discovering process. Google distance algorithm normally

consumes over 95% of the execution times spent for web service matching.

5.2.1 Overhead Examination

In order to examine overheads, Figure 5.4 shows the execution time comparison chart without the major overhead components: Google Distance and the Ontology engine. In this chart, we deducted the execution time for Google Distance algorithm from the total execution time. As well, we took out the E-Workflow approach performed on the wine ontology from this comparison to avoid the overheads introduced by Jena and Pellet reasoners on processing the ontology. We can see the differences compared to the chart shown in Figure 5.3. Without the major components contributing to overhead, the remaining pieces of overhead the FOIQOS approach is similar to the E-Workflow approach. FOIQOS performs even slightly better when finding all the possible solutions. When the number of web services increased, the remaining overhead of the MOACO approach increased significantly compared to E-Workflow approach and the proposed approach.

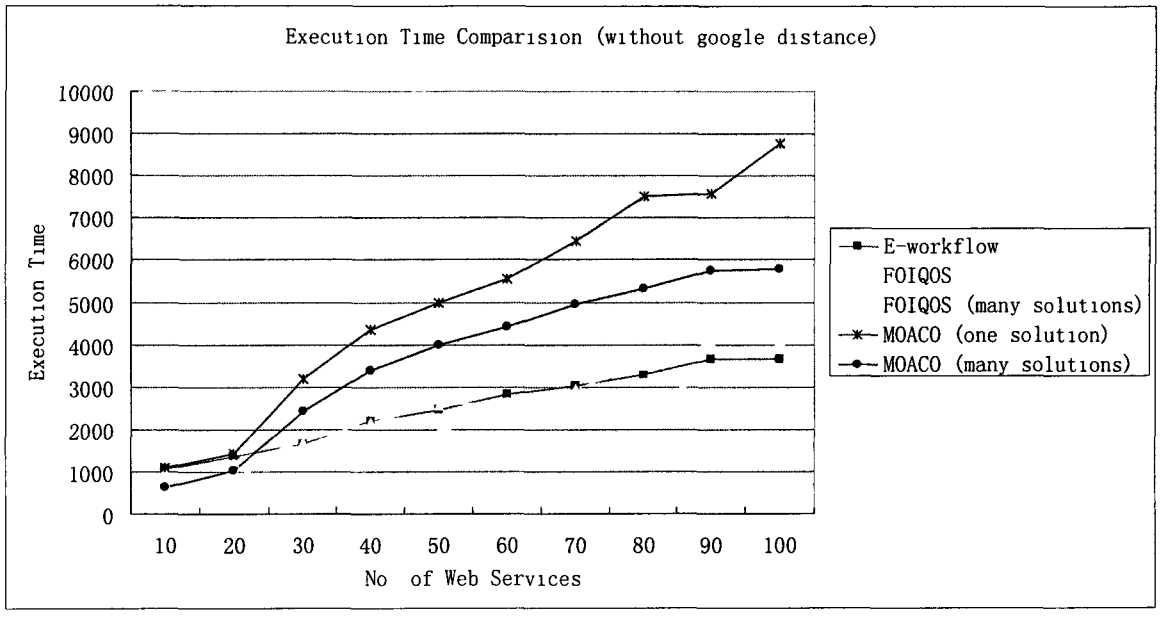


Figure 5.4 Total Execution time comparision without Google Distance

Table 5.1 and Figure 5.5 show the detailed execution times for each major steps during the FOIQOS discovering and composition process. The whole process can be divided into 3 steps: before matching, matching, and composition.

In the before matching step, the system loads and parses the inputs, searches the UDDI registry to get all the privacy services, and initiates the Google Distance APIs. In this step, loading and parsing inputs usuallly takes about 100 milliseconds, Google initiating usually takes 200 to 400 milliseconds; the time spent for searching the UDDI is the most time consuming part in this step and it increased significantly when the number of web service instances increased.

In the matching step, the system performs stemming and stop words removal for all the web service names and descriptions, calculates syntactic similarities by using the Q-Gram distance, calculates semantic similarities by using Google Distance algorithm,

calculates QoS similarities and finally get the best matching web service instances or find the possible composition solutions. In this step, the most time consuming part is the Google Distance. From the numbers in Table 5.1, we can see Google Distance usually took over 95% of the execution times in the matching step.

In the composition step, the system parses the WSDL files for each best matching web service instances and actually invoke the web services based on the input workflow. The execution time of this step is quite stable compared to other two steps.

From Table 5.1, we can see the time spent for the matching step dominates the total execution time. While the time spend for the Google Distance are dominant the time spend for the matching step. This explains why the execution times dropped when the number of web service instances increased from 40 to 50 and from 90 to 100. As mentioned before, the time spent for Google Distance depends on the performance of the search APIs provided by the search engine service providers. When those search APIs performed better, our total execution times could drop even with increased number of web service instances. Network delay could be another major factor that affects the performance of the Google Distance since the search services are all internet-based.

The E-Workflow approach follows the same major steps as discribed above. In the before matching step, instead of initiating Google Distance APIs, E-Workflow loads the domain ontology into memory during this step. This usually takes about 30 milliseconds to load the 41-term text based sample ontology, 297 milliseconds to load wine ontology, and 14801 milliseconds to load Mesh ontology. In the matching step, the E-Workflow approach queries the properties for the terms through the hash map (for the 41 term sample ontology), or by using Jena OWL APIS and Pellet reasoner for the wine ontology.

It usually takes less than 1 millisecond to query the hash map. The time spend to query the wine ontology increased from 7789 milliseconds to 190232 milliseconds when the total number of web service instances increased from 10 to 100. In the table 5.1 and the figure 5.5, column "Feature-based algorithm overhead" illustrates the overheads of the queries performed on the wine ontology.

Table 5.1 Detailed Execution times for each major step

No. Web Serv-ices	Before matching	Google Init (part of the Before Matching)	Feature-based algorithm overhead	Google distance (GD)	Matching incl. GD	Compos ition
10	632.7	245.4	7789.6	11374.7	11465.8	476.5
20	817.2	275.1	16631.2	21377.9	21562.5	428.1
30	1009.4	265.6	28203.6	29171.3	30868.7	471.9
40	1384.2	392	40851.6	47978.3	50482.9	464.2
50	1532.7	306.3	51411.2	43923.7	47031.2	461
60	1564.2	289.2	72126.4	55064.3	58582.7	471.8
70	1674.8	304.5	86767.2	64613.2	68920.5	448.4
80	1899.8	207.7	105598.8	76881.3	81700	793.8
90	1885.9	253.1	119797.8	81469.9	86639.3	535.7
100	2090.7	239	190232.2	77408.1	78612.6	526.8

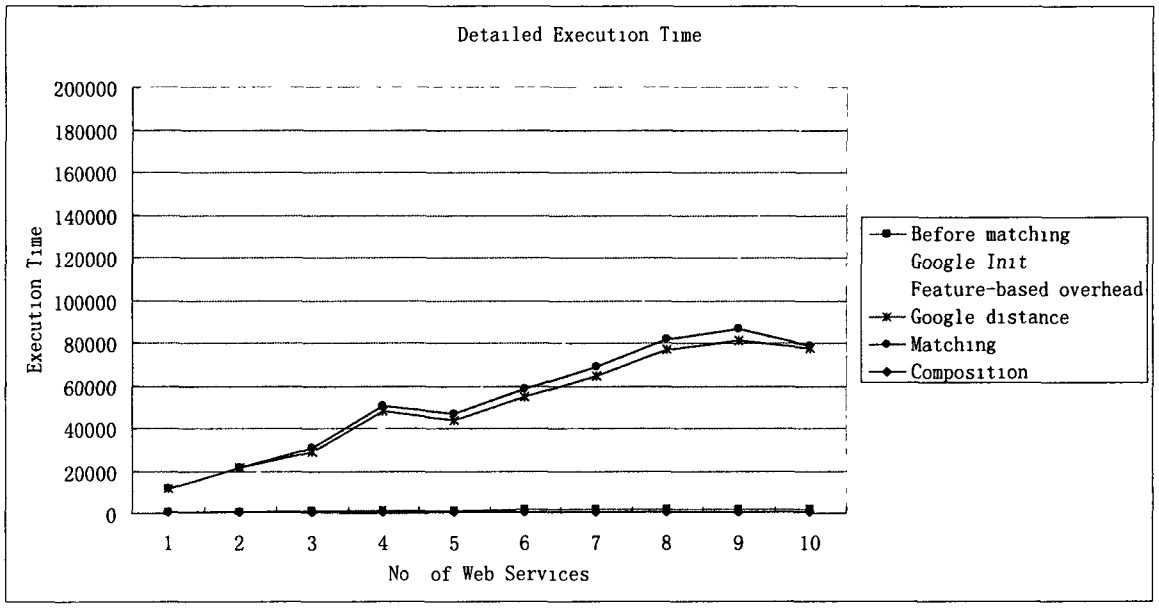


Figure 5.5 Detailed Execution times for each major steps (comparison chart)

Besides the Google Distance and the common tasks, such as parsing inputs and data preprocessing, shared by all the approaches, the major execution time difference among those approaches also depends on the times spent for calculating the similarity scores. Figures 5.6 and Table 5.2 depict the algorithm calculation times comparison for all the approaches.

The FOIQOS and the E-Workflow composition approaches calculate all the similarity scores based on the defined formulas. They spent almost no time on algorithm calculations. On the other hand, the MOACO algorithm actually is a learning process. It has to repeat hundreds of times to train the program to get the desired results. From both Tables 5.2 and Figure 5.6, we can see the MOACO approaches took more time than the FOIQOS approaches and E-Workflow approach. When the number of web service instances increase, the calculation times for MOACO increases significantly.

Table 5.2 Algorithm calculation times for each approach

No. Web Services	Feature-based	FOIQOS one solution	FOIQOS many solutions	MOACO one solution	MOACO many solutions
10	0	0	0	93.6	84.4
20	0	0	0	454	394.9
30	0	0	0	1420.3	1109.4
40	0	0	0	2200.1	1730.9
50	0	0	0	2599.9	2021.9
60	0	0	0	2897.1	2223.4
70	0	0	0	3564.1	2515.6
80	0	0	0	3840.7	2586
90	12.7	0	16	4151.6	2740.5
100	3.2	0	1.6	4984.3	2968.9

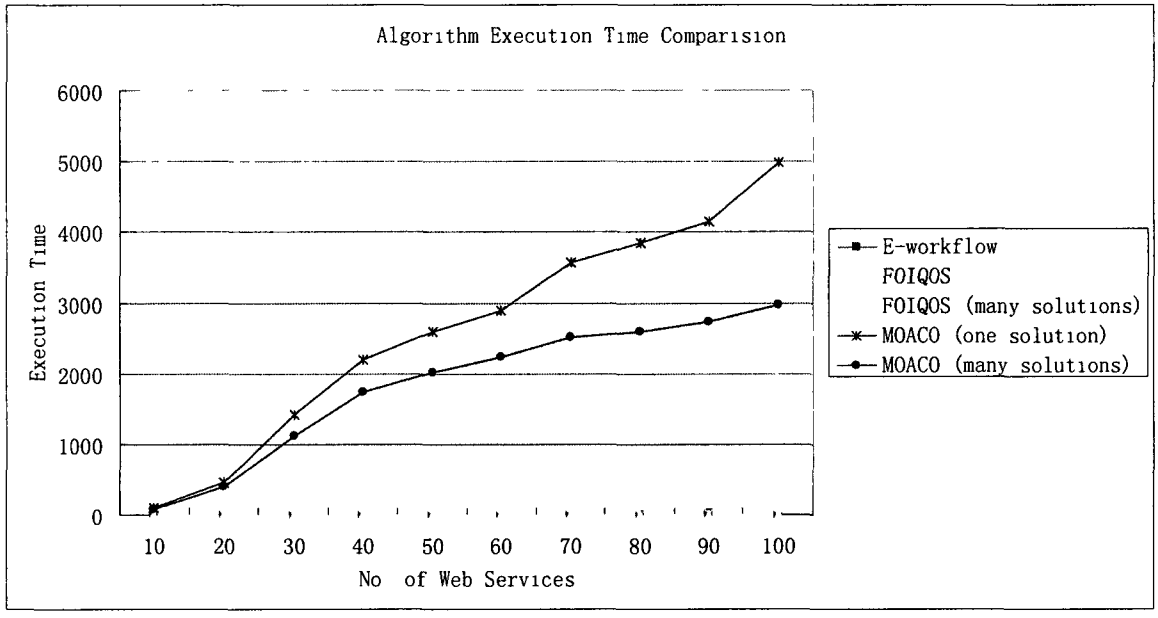


Figure 5.6 Algorithm calculation times for each approach (comparision chart)

5.2.2 Best Composition Solution

Table 5.3 compares the capability of each approach to find the best composition solution. Since all the implemented approaches use similar methods and criteria when filtering the web service instances in terms of the syntactic similarities and semantic similarities, we define the best composition solution as the following:

For the composition without global QoS constraints:

- each web service instance in the solution must be syntactically and semantically similar to the related web service type / template defined in the composition workflow. We call those web service instances candidate web services;
- each web service instance in the solution must have best QoS scores among all the candidate web services belongs to its web service type / template. A desired web service instance in the solution should have less Time value, less Cost value, and greater Reliability value. The QoS score is the overall QoS values based on Time, Cost, and Reliability values. It is calculated by using the formulas defined in each approach;

For the composition with global QoS constraints, other than the two conditions listed above, the composition solution must meet the defined global QoS constraints as well. The experimental results show the E-Workflow composition approach always did not find the best composition solution, while the FOIQOS approach can always find the best composition solution. The MOACO approach can find the best composition solution but not always. When the number of web service instances increased, the capability of MOACO to find the best solution decreased.

Table 5.3 Capability to find the best solution

No. Web Services	E-workflow composition	FOIQOS	MOACO
10	0%	100%	80%
20	0%	100%	60%
30	0%	100%	50%
40	0%	100%	50%

50	0%	100%	30%
60	0%	100%	40%
70	0%	100%	30%
80	0%	100%	20%
90	0%	100%	0%
100	0%	100%	30%

The E-Workflow composition approach only considers the absolute distance when calculating QoS similarities. The problem with the absolute distance is that it ignores the negative distance. That means when a SO's QoS value is slightly less than the defined minimum ST's QoS value, the system will give it a higher score when compared to a SO's QoS value which is much greater than the defined minimum QoS value. That's definitely not the desired result. Another problem with the E-Workflow composition is that it defines Minimum, Average, and Maximum values for each ST (Service Template / Type)'s QoS parameters. The similarities were calculated based on the absolute distance between ST_{min}, SO_{min}, and ST_{avg}, SO_{avg}, and ST_{max}, SO_{max}. This seems reasonable, however, it is not practical. When considering the quality of the services, the service requestors actually only care about the acceptable values to him. For example, considering the QoS parameter time, when a service requester defines time in a ST, usually, it only cares about the longest process time (t1) acceptable to it and possibly it will define the preferred process time (t2) as well. With the E-Workflow approach, the algorithm will tend to choose a SO whose process time is between t1 and t2 rather than choose a service whose process time is shorter than t2. That selection is undesirable because regarding the QoS parameter time, the shorter the better.

---

st time: 10, 20, 50; so time: 5, 10, 15
st cost: 10, 15, 30; so cost: 10, 10, 10

st Reliability: 0.85, 0.9, 1.0; so Reliability: 0.9, 0.95, 0.99
dcd time: 0.8753966455665646; dcd cost: 0.9319259231277157; dcdReliability: 0.9582839885514031;
qos d: 0.9212132321989214
st time 10, 20, 50, so time 10, 15, 20
st cost: 10, 15, 30; so cost· 10, 15, 20
st Reliability 0 85, 0 9, 1.0, so Reliability· 0 85, 0 9, 0.95
dcd time 0 915241126759053; dcd cost 0 9736718881674383; dcdReliability 0 983047550507815,
qos d· 0 9568434638599167
ST1, The best matching SO is: SO Details -
Name:Third Party List, Description:Return the third party list of the input web site, Input:web site, Output:third party list, Time Min:10, Cost Min:10, Reliability Min:0.85, Time Avg:15, Cost Avg:15, Reliability Avg:0.9, Time Max:20, Cost Max:20, Reliability Max:0.95.
QoS score: 0.9568434638599167

Figure 5.7 Illustration of E-Workflow Operational Matching Step

In Figure 5.7 is a fragment of raw service discovering results by using E-Workflow composition approach. From the above fragment, we can see the algorithm chooses the SO with the highest QoS score (italicized) as the best matching SO for ST1. However, when we compare to the SO highlighted in boldface, we can see the bolded SO actually has lower time values, lower cost values, and higher reliability values. However the algorithm gave the higher QoS score to the SO shows the algorithm cannot choose the best web service instance with the better QoSvalues, just because its minimum, average, and maximun values are closer to ST's minimum, avergage, and maximum values. That shows how the E-Workflow composition approach cannot choose the best web service instance with the better QoS values in the last filter stage of the service discovering process and thus it cannot find the best composition solution.

The MOACO approach, as mentioned before, is a learning algorithm. We put 30 ants into the directed graph, each ant's start position is determined by a probability

formula and the next node on the path is determined randomly. We repeat the program 100 / 200 times. When there is less number of web service instances, the program takes less time to walkthrough the whole graph and has higher probility to fnd the best composition solution; while when the number of web service instances increased significantly, with 30 ants and 100 iteration times, the program may not be able to walk through the whole graph to find all the composition solutions, and of course it even harder for it to find the best composition solution. The probability for MOACO approach to find the best composition solutions is low and irregular when there is a large number of web service instances.

5.2.3 Global QOS Constraints and Possible Composition Solutions

The MOACO approach aims to find possible composition solutions that meet the defined global QoS constraints. To compare with the MOACO approach on this perspective, I designed and implemented the FOIQOS algorithm to realize the same functionalities.

Table 5.4 Possible solutions comparision

WS instance No.	Total path	Algorithm	Time spend (ms)	Paths found	%
10	17	Our approach	0	17	100%
		MOACO	42.2	17	100%
20	44	Our approach	0	44	100%
		MOACO	98.6	43.8	100%
30	130	Our approach	0	130	100%
		MOACO	1109.4	127.8	98%
40	375	Our approach	0	375	100%
		MOACO	1730.9	327.1	87%
50	556	Our approach	0	556	100%
		MOACO	2021.9	411.4	74%
60	728	Our approach	0	728	100%
		MOACO	2223.4	487.9	67%
70	1064	Our approach	0	1064	100%
		MOACO	2515.6	605.6	57%

80	1370	Our approach		0	1370	100%
		MOACO (Iterations)	100	2586	576.2	42%
			200	5890	798	58%
90	1453	Our approach		16	1453	100%
		MOACO	100	2740.5	598.8	41%
			200	6197	783.3	54%
100	2104	Our approach		1.6	2104	100%
		MOACO	100	2968.9	716.4	34%
			200	6906.3	1042	49.5%

Table 5.4 shows the comparison of the MOACO approach and the FOIQOS approach in terms of the capabilities to find the possible composition solutions that meet the global QoS constraints. The results show the FOIQOS approach can always find all the possible composition solutions in almost no time. The MOACO approach can find almost all the solutions when the number of web service instances is less than 30. Its capability to find the possible composition solutions dropped significantly when the number of web service instances increased. Increasing the iteration times can improve the capability to find the possible solutions, but it will increase the execution time significantly.

Chapter 6

Conclusions and Future Work

With the increased usage of web services, demand for better service discovering mechanisms and dynamic service composition approaches is similarly increasing. Since the early 2000s, the scientific literature shows active contributions to the web service discovering and composition areas. Service discovering mechanisms mainly consider the functional properties (i.e. inputs, outputs) and operational properties (i.e. QoS). Further, dynamic web service composition approaches mainly fall into the realm of workflow composition or AI planning. In this thesis, we illustrated the problems that exist with discovery approaches using ontologies, and highlighted the issues that exist in several methods at the syntactic, semantic, and QoS operational similarity matching stages. Further, the thesis proposed to use Google distance as an alternative to using domain ontologies at the semantic matching stage. To evaluate the quality and performance of the FOIQOS approach, we implement the E-Workflow composition approach, and a recent AI-planning approach named MOACO, and compared the experiment results.

Basically, the service discovering process falls into two logical steps: WS matching (meet the functional requirements, e.g. IOPE), and WS selection (meet the non-functional requirements. e.g. QoS). Matching approaches use the syntactic similarity and semantic similarity stages to filter out the unwanted service instances step by step. FOIQOS and E-Workflow use the QoS operational similarity step to select the best matching service instances as the actual concrete web services that will be invoked in the actual composition. The MOACO approach first finds the matching web service instances

for each involved web service type (WS matching), then, based on the matching results, it builds a directed graph and applies its selection algorithm. The E-Workflow composition approach can be used in different ways, as it gives users the flexibility to use syntactic, semantic, and QoS operational similarities stages as three separate ranking methods. The user can choose one of them as the actual ranking method. So actually, there may not be a stepwise final best selection algorithm. To make it comparable to the automatic FOIQOS and MOACO approaches, we got rid of human interaction and applied the similar selection process (Syntactic selection, Semantic selection, and QoS selection) as in the FOIQOS approach but with the E-Workflow formulae and methods. In each selection step, we applied the corresponding ranking method defined in the E-Workflow composition approach. Specifically, the E-Workflow approach applied Q-gram algorithm in the syntactic selection step, a feature-based algorithm in semantic selection step, and its own QoS distance formulas in QoS selection step. The FOIQOS approach employs the Q-gram algorithms in the syntactic selection step, Google distance in the semantic selection step, and its own QoS distance formulas in the QoS operational selection step. The MOACO approach did not specify the algorithm that was used for Syntactic and Semantic selections in [Fang et al, 2009]. In our implementation, it shared the same Syntactic and Semantic selection algorithms with the FOIQOS approach, but set the global QoS constraints and used its native MOACO algorithm in the QoS selection step.

The experimental results show the FOIQOS approach outperforms the E-Workflow and MOACO approaches in terms of the accuracy i.e. best selection of web services to compose. FOIQOS can always find the best composition solutions and all the possible composition solutions when considering global QoS constraints. In terms of

execution times, the performance of the FOIQOS approach is better than both MOACO and the E-Workflow approaches.

We compare a feature-based algorithm with the Google Distance algorithm. Theoretically a feature-based algorithm should be relatively more accurate in a specific domain. However, it needs a pre-defined domain ontology, and highly relies on the quality of the domain ontology to get accurate semantic matching results. With a polluted or poorly maintained domain ontology, it is hard for a feature-based algorithm to always get the accurate semantic similarity scores. In the cases where no property is defined for a queried term, or the queried term is not defined in the ontology at all, or even worse the properties are not specified correctly, the feature-based algorithm used in the E-Workflow approach needs complementary rules or methods to get more accurate results. Google distance is not domain specific. It can be used to compute semantic distance between any two terms, and can still get the accurate results. The drawback of the Google Distance algorithm is its dependency on the web search provider, and it takes more time because of network delay. However, compared to a feature-based algorithm on an OWL ontology, the Google distance algorithm actually performs better. The performance of a feature-based algorithm drops significantly when the size of the ontology increases. Whereas the performance of the Google distance algorithm is relatively stable.

When computing the QoS similarity, the E-Workflow composition and most of the existing web service composition approaches, e.g. the approach defined in [Taher, 2005], only consider the absolute distance when computing the similarity distance. They define min, avg, max values for each ST's QoS parameters. The algorithm will tend to choose a web service instance whose QoS values (SOmin, SOavg, SOmax) closest to the values

defined in the web service template (STmin, STavg, STmax) rather than choose a web service instance with the best QoS values (shorter time, less cost, and better reliability).

The FOIQOS approach only defines acceptable values for each ST. Instead of calculating the absolute distance, it discards the SOs with the negative distances (which means at least one of the QoS values are not acceptable). As well, it introduces failure rate to represent reliability (failure rate = 1 - reliability). This is to make sure that for values of all the three QoS parameters, the smaller values are always better than larger ones. With this QoS algorithm, the program can always find the SOs with the best QoS values. And because it discard the SOs with unacceptable QoS values, that will speed up the selection process.

With the MOACO approach, a service requestor can set the global QoS constraints. The algorithm can find a set of possible composition solutions which satisfy the global QoS constraints. Based on the pheromone left on each solution it can find an optimized solution as the best composition solution. However, due to the fact that MOACO is a learning-based algorithm, the amount of the possible solutions it can find and the execution times it spends depend on the number of the ants and the total iteration times. With limited iteration times, it may not always find the best composition solution. MOACO does not always find all the possible composition solutions. The overall performance drops quickly when the number of web service instance increases. A second and modified FOIQOS approach implemented an improved graph walk-through algorithm to accommodate the global QoS constraints requirements. It can always find all the possible composition solutions in a very short time. The FOIQOS approach outperforms the MOACO approach in every aspect.

6.1 Future Work

Currently, there is no standard on how to present web service's operational properties in OWL-S. So in the thesis, the dynamic composition is based on WSDL, and the inputted composition workflow and web service templates are defined in XML formats, whereas the standard of the semantic web is using OWL / OWL-S. As for future work, we will use OWL-S to define the composition workflow and service templates. As well, all the web services will be specified in both WSDL and OWL-S. The service discovering and composition will be based on OWL-S instead of WSDL.

Recall the mention of the absence of commercial UDDI APIs to directly access QOS information. Thus current approaches using QOS information embedded in the UDDI require multiple steps to locate appropriate service such that the time required on service discovery would be longer than our implementation using UDDI categories and then local similarity matching. However recently, researchers (e.g. Paramala and Saini, 2011) have proposed more efficient methods for implementing UDDI-QOS APIs than those in Blum and Carter [2004] and Blum [2004] who previously described methods to use industry UDDI implementations to store QoS information. It would be useful to compare their similarity matching methods with our methods and approach.

Another interesting research project in the future would be applying automatic reasoning methods to generate the composition workflow and the service templates based on the request submitted by the service requestor.

7. REFERENCES

[Andrews, 2003] Andrews, T., "Business Process Execution Language for Web Services (BPEL4WS) 1.1.", 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel

[Bennicke, 2003] Bennicke, M., Langendorfer, "Towards automatic negotiation of privacy contracts for Internet services", Networks, the 11th IEEE International Conference, pp. 319 – 324, 2003

[Berners-Lee, 1998a] Berners-Lee, T., "Uniform Resource Identifiers (URI): Generic Syntax", 1998, http://www.isi.edu/in-notes/rfc2396.txt

[Berners-Lee, 2000] Berners-Lee, T., "Semantic Web - XML2000", 2000, http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html

[Berners-Lee, 2001] Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", 2001, http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2

[Berners-Lee,1998b] Berners-Lee, T., "Rules and Facts: Inference Engines vs. Web", Available: http.//www.w3.org/DesignIssues/Rules html, 1998, last updated 2009.

[Blum 2004] Adam Blum, "UDDI as an Extended Web Services Registry ", SOA WORLD MAGAZINE, Available: http://webservices.syscon.com/read/45102.htm

[Blum and Carter, 2004] Adam Blum and Fred Carter, "Representing Web Services Management Information", Available: http://www.oasisopen.org/committees/ download.php/5144/

[Cardoso, 2002] Cardoso, J., Sheth, A., Miller, J., "Workflow Quality of Service",

International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference, 2002

[Cardoso, 2003] Cardoso, J., Sheth, A., "Semantic E-Workflow Composition", Journal of Intelligent Information Systems, Vol. 21, No. 3, pp. 191 - 225, 2003

[Chapman, 2006] Chapman, S., "String Similarity Metrics for Information Integration", 2006, http://www.dcs.shef.ac.uk/~sam/stringmetrics.html

[DLML, 2003] DLML, http://co4.inrialpes.fr/xml/dlml/, 2003

[Dorigo, 1999] Dorigo, M., and Caro, G.D., "The Ant Colony Optimization metaheuristic," New Ideas in Optimization, McGraw Hill, London, 1999, pp.11-32.

[Evangelista, 2006] Evangelista, A., Kjos-Hanssen, B., "Google Distance Between Words", University of Connecticut, 2006.

[Fang, 2009] Fang, Q., Peng, X., Liu, Q., Hu, Y., "A Global QoS Optimizing Web Services Selection Algorithm Based on MOACO for Dynamic Web Service Composition", ifita, vol. 1, pp.37-42, 2009

[Fensel, 2001] Fensel, D., "Ontologies: Silver Bullet for Knowledge Management and Electronic commerce", 2001

[Gilleland, 2006] Gilleland, M., "Levenshtein Distance, in Three Flavors", 2006, http://www.merriampark.com/ld.htm

[Grigoris, 2004] Grigoris, A., Frank, H., "A Semantic Web Primer", 2004.

[Grigoris, 2004] Grigoris, A., Frank, H., "Web Ontology Language: OWL", 2004.

[Gruber, 1993] Gruber, T. R., "Toward Principles for the Design of Ontologies Used for Knowledge sharing", 1993, http://ksl-web.stanford.edu/KSL_Abstracts/KSL-93-04.html

[Heather, 2001] Heather, K., "Web Services Conceptual Architecture", 2001.

[Hong Kong University, 2004] Hong Kong University WWW, "A work mechanism of a simple search engine," 2004, http://ihome.ust.hk/~egwws/comp336/lab4/lab4.htm

[Jinghua, 2005] Jinghua, Z., "Study on Applying Semantic Web to Intelligent Information Retrieval", 2005.

[Karunamurthy, 2006] Karunamurthy, R., Khendek, F., Glitho, R., "A Novel Business Model for Web Service Composition", IEEE International Conference on Services Computing, pp. 431-437, 2006

[Keller, 2003] Keller, A., Ludwig, H., "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", Journal of Network and Systems Management, Special Issue on E-Business Management, Volume 11, Number 1, 2003.

[Kolari, 2005] Kolari, P., Ding, L., Shashidhara, G., Joshi, A., Finin, T., Kagal, L., "Enhancing Web privacy protection through declarative policies", Policies for Distributed Systems and Networks, Sixth IEEE International Workshop, pp. 57 – 66, 2005

[Lancaster University, 2004] Lancaster University WWW, "What is Stemming?" 2004, http://www.comp.lancs.ac.uk/computing/research/stemming/general/

[Lesp´erance, 1997] Lesp´erance, Y., Reiter, R., Lin, F., Scherl, R., "GOLOG: A logic programming language for dynamic domains," Journal of Logic Programming, vol. 31, no. 1-3, pp. 59– 83, 1997.

[Li, 2005] Li, B., Tang, X., Lv, J., "The Research and Implementation of Services Discovery Agent in Web Services Composition Framework", Machine Learning and Cybernetics, 2005

[Liu, 2005] Liu, J., Gu, N., Zong, Y., Ding, Z., Zhang, S., Zhang, Q., "Service

Registration and Discovery in a Domain-Oriented UDDI Registry", Proceedings of the Fifth International Conference on Computer and Information Technology, 2005

[McIlraith, 2002] McIlraith, S., Son, T., "Adapting Golog for Composition of Semantic Web Services", Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning, pp. 77-88, 2002

[Nie, 2006] Nie, T., Yu, G., Shen, D., Kou, Y., "An Approach for Composing Web Services on Demand", Computer Supported Cooperative Work in Design, 10th International Conference, 2006, pp. 1-6.

[Parimala, 2011], Parimala N. and Saini, A. Decision Support Web Service, Distributed Computing and Internet Technology, Lecture Notes in Computer Science, 2011, Volume 6536/2011, 221-231.

[Ponnekanti, 2002] Ponnekanti, S. R., Fox, A., "SWORD: A developer toolkit for Web service composition", Proceedings of the 11th World Wide Web Conference, pp. 24-32, 2002

[Serhani, 2005] Serhani, M.A., Dssouli, R., Hafid, A., and Sahraoui, H. "A QoSbroker based architecture for efficient web service selection", Proc. of the IEEE Int. Conference on Web Services, (ICWS'05), IEEE 2005, pp. 113 – 120.

[Systinet Corp., 2003] Systinet Corp., " Web Services: A Practical Introduction to SOAP Web Services", 2003,

[Taher, 2005] Taher, L., Basha, R., Khatib, H. E., "Establishing Association between QoS Properties in Service Oriented Architecture", International Conference on Next Generation Web Services Practices, pp6, 2005

[Tan, 2005] Tan, H., Guo, J. "Some methods to depress the risks of the online

transactions", E-marketing & e-businesses, Proceedings of the 7th international conference on Electronic commerce, ACM International Conference Proceeding Series; Vol. 113, pp. 217 – 220, 2005

[Technology Investigation Center, 2003] Technology Investigation Center, "P3P - Platform for Privacy Preferences Project by the W3C", 2003, http://www.tic.state.ar.us/Reports/p3preport.htm

[Teknomo, 2006] Teknomo, K., "City Block Distance", 2006, http://people.revoledu.com/kardi/tutorial/Similarity/CityBlockDistance.html

[Tran, 2008] Tran, V.X., WS-QoSOnto: A QoS Ontology for Web Services, 2008 IEEE Symp. on Service Oriented System Engineering. pp. 233-238.

[Thissen, 2006] Thissen, D., Wesnarat, P., "Considering QoS Aspects in Web Service Composition", Computers and Communications conference, pp.371 – 377, 2006

[Tversky, 2005] Tversky, A. "Features of Similarity." Psychological Review Vol. 84, No. 4, pp. 327-352, 2005

[Van Veldhuizen, 1999] Van Veldhuizen, D. A., "Multiobjective Evolutionary Algorithms: Classifications, Analyses and New Innovations," Ph. D. thesis , Air Force Institute of Technology Wright Patterson AFB, OH, USA .

[Vitanyi, 2004] Vitanyi, P., Cilibrasi, R., "Automatic Meaning Discovery Using Google", 2004, http://www.arxiv.org/abs/cs.CL/0412098

[W3C Recommendation, 2004a] W3C Recommendation, "Resource Description Framework (RDF): Concepts and Abstract Syntax", 2004

[W3C Recommendation, 2004b] W3C Recommendation, "OWL Web Ontology Language Overview ", 2004

[W3C Recommendation, 2005] W3C Recommendation, "RDF Vocabulary Description Language 1.0: RDF Schema", 2005

[W3C, 2005] W3C, "P3P Public Overview", 2005, http://www.w3.org/P3P/

[W3Schools, 2005] W3Schools, "XML Tutorial", 2005, http://www.w3schools.com/xml/default.asp

[Wang, 2006] Wang, X. Vitvar, T. Kerrigan, M., and Toma, I. "A QoS-Aware Selection Model for Semantic Web Services," in Proc. of ICSOC 2006, pp. 390-401. June 2006.

[Wu, 2003] Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B., "Automatic Web services Composition using SHOP2", Workshop on Planning for Web Services, 2003, 7 pages.

[Ye, 2006] Ye, L., Zhang, B., "Discovering Web Services Based on Functional Semantics", IEEE Asia-Pacific Conference on Services Computing, pp. 348-355, 2006