2005

Web service searching

Ismail Jaghmani
University of Windsor

Web Service Searching

By

Ismail Jaghmani

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2005

Your file Votre référence
ISBN: 0-494-09795-7
Our file Notre référence
ISBN: 0-494-09795-7

Abstract

With the growing number of Web services, it is no longer adequate to locate a Web service by searching its name or browsing a UDDI directory. An efficient Web services discovery mechanism is necessary for locating and selecting the required Web services. Searching mechanism should be based on Web service description rather than on keywords. In this work, we introduce a Web service searching prototype that can locate Web services by comparing all available information encoded in Web service description, such as operation name, input and output types, the structure of the underlying XML schema, and the semantic of element names. Our approach combines information-retrieval techniques, weighted bipartite graph matching algorithm and tree-matching algorithm. Given a query, represented as set of keywords, Web service description, or operation description, an information retrieval technique is used to rank the candidate Web services based on their text-base similarity to the query. The ranked result can be further refined by computing their structure similarity. Data types are matched by modeling the underlying XML schema as tree; each node in the tree represents an element in the schema. A tree-matching algorithm is implemented to compute the data type similarity. The experimental results demonstrated the flexibility, efficiency and effectiveness introduced by the proposed approach.

Keywords: XML, XML schema, schema matching, mapping, schema similarity, tree matching, WSDL, SOAP, Vector Space Model, WordNet, name similarity, node similarity, structural similarity, Information Retrieval, Graph Matching

Acknowledgements

I wish to express my appreciation and gratitude to the thesis supervisor Dr. Jianguo Lu for his guidance and encouragement throughout the course of this research work. Thanks are also extended to Mr. Xiaolei Yuan for data collection and Mr. Ju Wang for his constructive discussions.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Web service technology has won the support of major software vendors such as Microsoft, IBM, and Sun Microsystems. Integrated Drivers IDC estimated that software, services, and hardware business created by the demand for Web services could increase from $1.6 billion in 2004 to $34 billion by 2007 [70].

Web services are self-contained self-describing software components that can be published, accessed and even brokered over the Internet. A Web service is defined by the world wide Web consortium (W3C) [78] as *"A software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols."*

Web service elevates the Web functionality from document oriented to application oriented. It is motivated by two drawbacks in the current software development practice. One is that the plethora of the services provided on the Web nowadays is meant for human use, not for applications to access and integrate. The other drawback is that the existing distributed component models such as Common Object Request broker Architecture (CORBA), Distributed Component Object Model (DCOM) are based on standards other than Hyper Text Markup Language (HTTP) and Extensible Markup Language (XML), which means they are not easy to be accessed over the Internet, or go through the firewalls. Web service technology is meant to combine the better of the two approaches while avoiding the drawbacks. It is a new model of distributed computing that provides a language and platform-independent syntax. Web services allow the application functionality to be defined in reusable standard format providing an easy way to integrate business applications and reduce the time and cost for application development and maintenance.

Three key parts of a Web service are: Web Services Description Language (WSDL), XML Schema, and Simple Object Access Protocol (SOAP). While WSDL provides the

syntax to describe the interface of Web Services, XML Schema is the language used in WSDL to define the data types of input and output messages. SOAP is a transport protocol used for communicating messages and data for WSDL.

With the growing number of Web services, it is no longer adequate to locate a web service by searching its name or by browsing the Universal Description Discovery and Integration (UDDI) directory. An efficient Web service discovery mechanism is necessary for locating and selecting the required Web service. An automatic Web services discovery and composition is one of the main concerns in the area of software engineering [15].

UDDI [75] defines a centralized registry for service discovery that is based on keywords search and leaves many things open such as how to locate similar Web services. WSIL [76] is a different model that complements UDDI by providing a lightweight model to improve service discovery. However, neither UDDI nor WSIL represent services description, therefore, they are no help for discovering services based on what they provide. Both UDDI and WSIL rely on other service description mechanism such as WSDL [77].

The research problem is how to accomplish flexible, efficient and effective Web service discovery using WSDL specifications. The difficulty in solving this problem arises from the fact that WSDL is described using XML structure. Matching between two XML documents is turned out to be very expensive in term of computational time. In addition WSDL describes data types using XML schema that can be of a very complex structure.

In this work we describe a novel approach for searching Web services. The proposed approach provides three search criteria with two filtering modes. The filtering modes are text comparison and structure similarity. The text comparison-filtering mode treats the query and the target, documents as text and determines the similarity using information retrieval techniques. The structure similarity considers the structure of the query and the target and computes the similarity based on their structures. The search criteria are a

keywords search, an operation search and a Web service search. The keywords search takes a set of keywords as a query and returns a list of Web services. The operation search takes an operation description as a query and returns a list of operations. The Web service search takes a Web service as query and returns a list of Web services.

In particular, our goal is to build Web services search mechanism based on WSDL specifications with the following aspects:

- Speeding up the computational time by:
    - Combining bipartite graph matching with recursive tree matching
    - Using top-down approach
        - Matching process starts by comparing operations
        - Input parameters of the source operation are only compared with input parameters of the target operation
        - Output parameters of the source operation are compared only with output parameters of the target operations
    - Eliminating all irrelevant Web services using less computational cost filtering mode
    - Caching parameters
- Including most of data type syntax
    - Occurrence indicators, order indicators and group indicator
    - Considering the similarity between data types from different categories
- Providing a flexible search engine that provides keyword search, operation search, and Web service search
- Providing a detailed experimental evaluation on a set of over 1400 Web services

The remaining of this work is organized as follows. Section 2 describes the research background. Section 3 presents an overview of our approach. Section 4 describes text comparison. Section 5 describes structure similarity. Chapter 6 describes the conclusion and future work.

Chapter 2: Background

2.1 Preliminaries

Web services neither required to be described using XML nor required to carry XML message or be bounded to a protocol capable of carrying XML messages. However using such technologies provides a platform-independent mechanism for application written in different programming languages to communicate over the Internet. Distributed technologies such as DCOM, CORBA, and Java Remote Method Invocation (Java RMI) are complex to implement and most of them require runtime libraries to be installed in the communicating systems. In addition to that, some systems provide additional application level for services such as garbage collection and session management that increase their complexity [73].

With the introduction of XML the industry and the academia focus has been shifted to develop additional technology such as Document object Model (DOM), Simple API for XML (SAX), XML Path Language (Xpath), Extensible Markup Language Transformation (XSLT), Simple Object Access Protocol (SOAP), XML schema and WSDL. These developments offer a set of technologies for Web services, where services are described and exposed on the Web using WSDL and communicate with each other using protocol capable of carrying XML messages. The objects exchanged between services are defined using XML schema.

In this section, a review of technology standards related to Web services are introduced. The basic concepts of XML and XML schema, the WSDL structure and how it embraces the use of XML schema and SOAP are described. Furthermore the graph and the tree concepts are introduced.

2.1.1 XML and XML Schema

XML stands for Extensible Mark-up Language. It was released by the World Wide Web Consortium (W3C) on February 10, 1998[66]. XML design is similar to Hypertext Mark-

up Language (HTML) [68]. Unlike XML, HTML was designed to display data, and its centre of attention was on how data is represented. It describes the presentation of the data on the browsers. It defines the style of the document by defining tags for heading, text format, links, tables, etc. All HTML tags are not case sensitive and all of them are predefined. However, XML was designed to describe the structure of data, not its presentation. XML file can be displayed in different formats with different content using Cascading Style Sheet (CSS) [64] and XSLT [82]. Unlike HTML, XML tags are case sensitive and not predefined. The basic unit in an XML structure is called *element.* An element is defined by its opening tag *(<>)* and closing tag *(</>).* XML document consists of strictly nested hierarchy of elements with a single root (top-level element). All other elements in the document are either direct or indirect children of the root element. An XML document must be syntactically correct and all opening tags must have corresponding closing tags. An XML document can be easily displayed on the Web or transferred to another document using XSLT. In HTML any change in the document tag will lead to the change in the way the document is displayed by the browser. However, an XML document can be displayed in different format and any changes in the document tags do not necessarily change the way the document is displayed. The XML structure is self-describing; each tag either describes what kind of information it contains or how this information is going to be interpreted. The following XML structure describes information about a car: (figure 1)

```
<car >
        < type>
            Ford
        </type>
        <year>
          2004
        </year>
        <colour>
          black
        </colour>
</car>
```

Figure 1: Example of XML Structure

The <*type*> element describes the type of the car. The <*year*> element describes the year the car was manufactured and the <*colour*> describes the colour of the car. XML can be processed and created by any application; the only thing required to process an XML document is an XML parser. The flexibility and simplicity of defining an XML document makes it ideal to store, carry, publish and exchange data among different applications and platforms.

The structure of an XML document can be controlled using Document Type Definition (DTD) [67] or XML schema [80]. Different applications can communicate and extract information from the same XML document as long as they use the same DTD or the same XML schema. However, unlike DTD, XML schema supports data types and wider range of constrains. The XML schema was proposed by Microsoft and became W3C recommendation in May 2001. An XML schema is an XML structure. It is used to specify and describe the structure and the content of XML documents.

Independent organizations can agree on a common XML schema for exchanging XML messages. Each organization uses the standard XML schema to verify that the data they receive is valid. When an XML document is processed, the parser compares the XML document with its associated XML schema to ensure that the XML document confirms the rules specified in the schema. Each element that appears in an XML instance must have an element declaration in the schema.

An XML schema defines a type system and constrains to describe an XML document. It organizes types as *built-in type, simple type,* and *complex type.* It supports an extensive set of built-in types that covers most of the types supported by other programming languages *(e.g. string, int, float etc.).* The built–in types are basic atomic data types that are built into XML schema. The build-in types comprise of primitive type such as *int*, float and derived types such as *positiveInteger.* Other derived types can also be created by restricting built-in types. An XML schema has 19 built-in primitive data types and 25 built-in derived types. A new derived type can be constructed using simpleType or complex type. A simpleType is defined by constraining a built-in type using constraining

facets. For example, *string* type facets are *length, minLength, maxLength, pattern, enumeration, and whiteSpace.* A complex type is defined as a list of types where each type can be built-in, simple or complex.

Consider the following XML element:

<car>Ford </car>

The car element can be constrained to contain only a value of type string.

```
< element name="car" minOccurs="0"  maxOccurs="1" type="string"/>
```

Figure 2: Example of XML Schema Built-in Type

The XML schema illustrated in figure 2 indicates that the car element can contain any value as long as the type of the value is string. XML schema also allows users to define the cardinality of an element, that is the number of times an element can occurs. The cardinality can be specified by the attribute minOccurs (the minimum number of occurrences of an element) and the attribute maxOccurs (the maximum number of occurrences of an element). In the above car element, the cardinality specifies that the element is optional as its minOccurs is set to zero and its maxOccurs is one. The range of cardinality is between 0 and unbounded.

As described in figure 3, the car element can be further restricted by defining it as a simple type.

```
<element name="car" type= "carInfo"/ >
    <simpleType name= "carInfo" >
        <restriction base="string">
                <enumeration value="Ford" />
                <enumeration value="BMW" />
     </restriction>
     </simpleType>

                or

<element name="car" >
    <simpleType >
        <restriction base="string">
                <enumeration value="Ford" />
                <enumeration value="BMW" />
     </restriction>
     </simpleType>
</element>
```

Figure 3: Example of XML Schema Simple Type

Figure 3 illustrates two different syntaxes of XML schema simple type that can be used to restrict the value of the car element. Both syntaxes presented in figure 3 indicate that the *car* element is a simple type and it's value is restricted using enumeration facet to be only Ford or BMW. Neither a built-in type nor a simple type can contain children elements.

Complex type elements can contain children elements. For example, the car element can have children elements as follows: (figure 4)

```
<car>
        <type>  Ford </type>
        <year>  2004 </year>
        <colour> black</colour>
 </car >
```

Figure 4: Example of XML Complex Type

The above structure can be described using a complex type as follows:

```
<element name="car" >
  <complexType >
    <sequence>
      <element name="type" type="string" />
      <element name="year" type="date" />
      <element name="colour" type="string" />
    </sequence>
  </complexType>
</element>
            or

<element name="car" name= "carInfo" />

<complexType  name= "carInfo">
    <sequence>
      <element name="type" type="string" />
      <element name="year" type="date" />
      <element name="color" type="string" />
    </sequence>
  </complexType>
```

Figure 5: Example of XML Schema Complex Type Structure

Figure 5 illustrates two different syntaxes of XML schema to describe a complex type element. Both syntaxes indicate that the *car* element is a complex type with three children. There are three kinds of indicators that restrict the order of complex type children elements; namely *sequence, choice,* and *all.* The *sequence* element indicates that the children elements should appear in the specified order; the *choice* element indicates that only one child element should appear and the *all* element indicates that the children elements can appear in any order. The XML schema presented in figure 5 indicates that the type, year and colour elements should appear in the specified order.

In addition to built-in types, simple types and complex types, XML schema also defines a *group* element that provides a way of component reuse. For example, the schema in figure 5 can be written as:

```
<element name="car" >
    <complexType >
      <group name="carInfo"/>
   </complexType>
</element>

<group name="carInfo">
   <sequence>
        <element name="type" type="carType" />
        <element name="year" type="date" />
       <element name="color" type="string" />
    </sequence>
</group>

<simpleType name= "carType" >
     <restriction base="string">
            <enumeration value="Ford" />
            <enumeration value="BMW" />
       </restriction>
</simpleType>
```

Figure 6: Example of XML Schema Group Element Structure

The complex type element *car* references the group element *carInfo.* The first child of the group element references a simple type element *carType.*

An element can also reference another element using the *ref* attribute for example:

```
<element name="truck" ref="car"/>
```

For elements, types, groups to be referenced by another element, they have to be direct children of the root element.

2.1.2 SOAP

The Simple Object Access Protocol SOAP [72] was proposed to W3C by HP, IBM, Microsoft and many other organizations in May 2000. The latest version of SOAP is SOAP 1.2 and it became a W3C recommendation on June 24, 2003. The specification defines SOAP as "*a lightweight protocol intended for exchanging structured information*

in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics"[72]. Unlike CORBA and COM, SOAP is an XML based protocol. It is easy to implement and does not require any software packages to install. SOAP did not introduce any new schema language, instead it refers to XML schema for syntax validation. SOAP defines a way of communicating messages between applications implemented with different programming languages and running on different platforms. The SOAP framework consists of the following XML elements: *Envelop, Header, Body* and *Fault*. The SOAP *Envelope* element is the root element of the SOAP message. It encapsulates all other elements and is used to identify a SOAP message. If a message is carried using HTTP protocol the *Envelope* element will directly follow the HTTP header. The SOAP *Header* element is optional and it contains auxiliary information such as security features. The SOAP *Body* element is required and it represents the message carried by the SOAP. It can contain any number of elements. The SOAP *Fault* element is optional and it represents an error format. Each fault element must contain *faultCode* element followed by *faultString* element. The *faultCode* element is used to classify the error and the *faultString* element is used to provide human readable description of the error message.

2.1.3 WSDL

WSDL (Web Service Description Language) [77] was submitted to W3C by Ariba, IBM and Microsoft on March 15, 2001. It defines the mechanism of interacting with a particular Web service. It provides the available tasks in form of operations, input/output messages, and binding information. WSDL comprised of five major elements that describe three aspects of a Web service. The types, messages and portTypes elements, describe what tasks the service provides. The binding element describes how to connect to the tasks provided by the service. The service element describes where the service is located.

- **<definitions>** The definitions element, acts as a root for the rest of the WSDL structure.

- **<service>** The service element provides a name for the service, and encloses one or more *port* elements. Each *port* element specifies a location where the service can be accessed. A *port* is defined by associating a network address with a *port type*. The *binding* element describes the protocol and the data format for operations provided by the service. Multiple ports mean multiple transports for the same service. This allows the use of any network protocol capable of carrying XML data. For example, some endpoints may use both HTTP and SMTP.

- **<binding>** The *binding* element includes a name attribute that provides a unique name for the *binding* among all bindings defined in the WSDL document. The *binding* element describes how to access a Web service by connecting *port types* to a *port*. It defines what operations a service provides, and what protocol should be used to access them.

- **<portType>** Each *port type* defines a name attribute that provides a unique name for the *port type* among all *port types* defined in the WSDL document. *Port types* are reusable and can be bound to multiple ports. They are logical grouping of operations where each operation describes a sequence of messages that may be exchanged with the Web service. These massages are defined via *input* and *output* elements.
  There are four types of operations:

  One-way: Messages sent without a reply required.

  Request/response: The sender sends a message and the received sends a reply.

  Solicit response: A request for a response.

  Notification: Messages sent to multiple receivers.

  It is important to note that WSDL does not describe how, for example, solicit-response and notification types of operations are implemented.

- **<message>** Each *message* contains a *name* attribute that provides a unique name for the message among all messages defined in a WSDL document and one or more *part*

element. Each *part* element defines an operation's parameter. Each *part* element contains two attributes; *name* attribute provides a unique name among all *parts* of a message and *typing* attributes, which can be an *element* that refers to an element in the schema or type that refers to XML schema data type. If the data type is not a build-in data type, then it must point to a type described in the schema element.

- **<types>** *Types* element encloses data type definitions that are relevant to the Web service exchanged messages. It contains a *schema* element that describes data types using XML schema type system.

In addition to the above structure, WSDL defines a documentation element that can be nested in any of the above elements. The main purpose of the documentation element is to provide human readable information about the element that contains it.

Figure 7 describes currency converter Web services using WSDL:

```
<definitions>
    <types>
     <schema targetNamespace="http://tempuri.org/">
      <element name="USDConvert">
          <complexType>
              <sequence>
                  <element name="ConvertTo" type="string" />
                  <element name="Amount" type="double" />
              </sequence>
          </complexType>
      </element>
      <element name="USDConvertResponse">
              <complexType>
                  <sequence>
                      <element name="USDConvertResult" type="double" />
                  </sequence>
              </complexType>
      </element>
     </schema>
    </types>
    <message name="USDConvertSoapIn">
        <part name="parameters" element="USDConvert" /></message>
    <message name="USDConvertSoapOut">
       <part name="parameters" element="USDConvertResponse" /></message>
   <portType name="CurrencyConverterSoap">
        <operation name="USDConvert">
              <input message="USDConvertSoapIn" />
              <output message="USDConvertSoapOut" />
         </operation>
     </portType>
     <binding name="CurrencyConverterSoap" >
        <soap:binding transport=http://schemas.xmlsoap.org/soap/http
                                                    style="document" />
       <operation name="USDConvert">
       <soap:operation soapAction="http://tempuri.org/USDConvert"
                                                              style="document" />
         <input>
            <soap:body use="literal" />
         </input>
         <output>
              <soap:body use="literal" />
          </output>
       </operation>
    </binding>
     <service name="CurrencyConverter">
        <port name="CurrencyConverterSoap
              binding="CurrencyConverterSoap">
                <soap:address location="http://www31.brinkster.com/
         webcomponents/CurrencyConverter.asmx" />
         </port>
   </service>
   <definitions>
```

Figure 7: WSDL Description for Currency Converter Web Service

Figure 8 illustrates the relationship between WSDL elements:

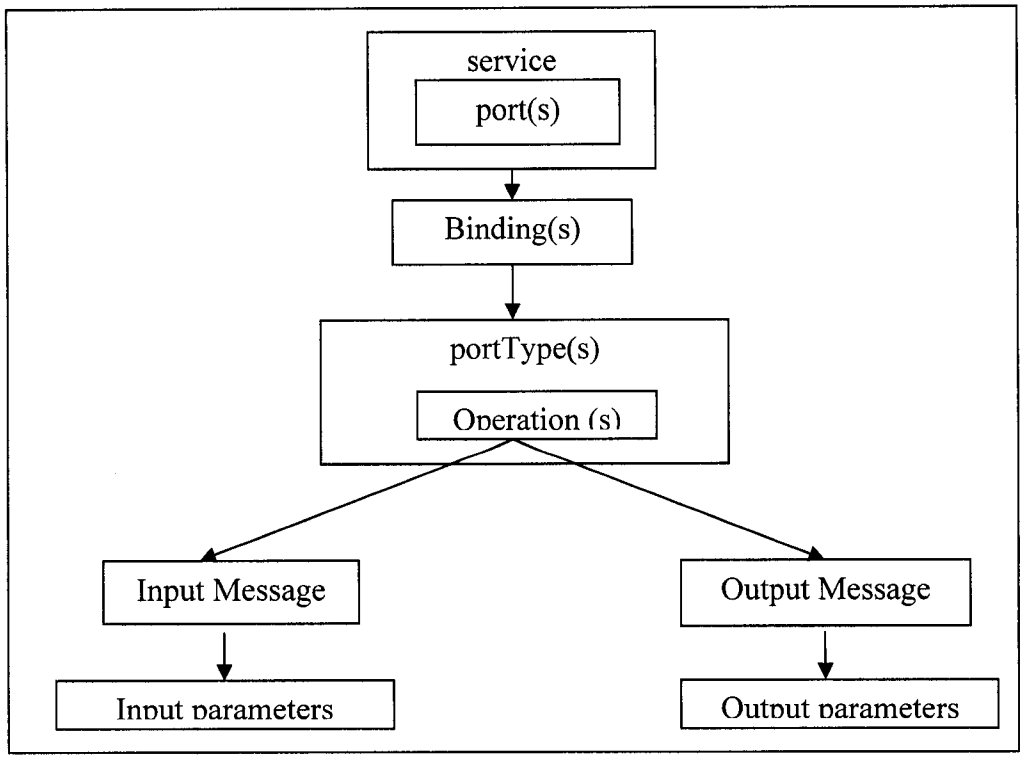

Figure 8: Relationship between WSDL Elements

In summary, a Web service is a network endpoint (ports) that provides an interface. The endpoint can be implemented in any programming language. The interface is bound to a concrete protocol and message format via one or more bindings, which are ways to communicate with the service. For example, a service may provide both a STMP and a HTTP interface. The binding lists the operations it supports, and what protocol to use to access that operation. The port type specifies what messages to send using the specified protocol. The messages are defined separately, which allows the reuse of the same messages. Each message consists of a number of parameters. Each parameter is a single object, defined in XML syntax.

2.1.4 Bipartite Matching Concepts

In this section we introduce a brief description of some of graph concepts and how it can be used in Web service matching. A graph can be defined as a set of vertices (nodes) and edges (lines that connect the nodes), each of them connect some pair of vertices. A graph

is called a directed graph if its edges go from one vertex to another in a specific direction. A graph is called undirected if its edges have no direction. The in-degree of a vertex is the number of edges incident to it and the out-degree is the number of edges incident from it. A graph is called weighted graph if each edge is assigned a weight value. To model a problem as a graph, objects are viewed as vertices, and their relation as edges. A tree can also be modeled as a graph by considering the root element as a vertex that has zero in-degree and out-degree equal to the number of its direct children. A special type of graph is called a bipartite graph [56]. The bipartite graph is a graph where its vertices can be partitioned into two subsets such that edges are only connecting nodes from different sets. The bipartite graph has been extensively used to solve matching problems. One of the classical problems is the assignment of workers to tasks to increase efficiency such that every worker is assigned to at most one task and every task is assigned to at most one worker. This problem can be represented as a graph by representing workers and tasks as vertices where the edges represent a weight that reflects the effectiveness of a worker at a given task. If we separate the workers and tasks to two separate subsets, the graph becomes a bipartite graph and the problem becomes a bipartite graph matching problem. The solution to this problem is finding the maximum total weight such each worker only assigned to one task.

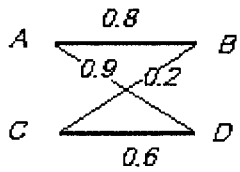

Figure 9: Bipartite Graph Matching

Using bipartite assignment matching, vertex A is matched to vertex B and vertex C is matched to D to maximize the total sum.

Another type of matching in a bipartite graph called the stable matching. Instead of optimizing the result to find the maximum total sum of the weight, the stable matching ensures that no pair will have higher weight than the current pair. A matching is stable if there is a vertex v and vertex u such that v can't be matched to another vertex u_1 with higher weight.

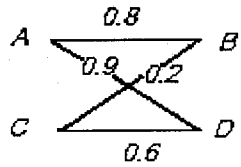

Figure 10: Bipartite Graph Stable Matching

The stable bipartite matching will match A to D and C to B. Even though the total sum is reduced but the matching is stable. The most common algorithm to solve bipartite graph matching problem is the Hungarian method [31] which grantee to find a solution in polynomial time.

2.1.5 WordNet and JWNL

WordNet is a lexical database containing the relations among English words. Its development began in 1985 by Princeton University [38,79,44]. WordNet has been used extensively in natural language processing [40,60]. The basic unit in WordNet is *synset*, representing a specific meaning of a word. A synset is the set of words that share the same sense (synonyms). The *synsets* are connected to each other with different types of relationships, such as *hypernym*; *y* is a hypernym of *x* if every *x* is a kind of *y* (e.g. *vehicle* is the *hypernym* of *car*). The synset includes nouns, verbs, adjectives, and adverbs. Each synset consists of synonym words and pointers to the hypernymy, hyponymy, antonymy, entailment, and meronymy/holonymy. The pointers represent the relation between a word in one *synset* and other *synsets*. The search process is first directed to an index file that contains the address of the *synset* in which the search word occurs. Depending on the search type (e.g Synonyms, Hypernym), the search can traverse many pointers from one *synset* to another until no further pointer encountered. The pointer traversing defines the path length of the search.

Java WordNet Library (JWNL) [69] is a Java API for accessing the WordNet relational dictionary. For example, *getImmediateRelationship (sourceWord, targetWord)* will looks at whether the target word is one of the words in one of the synsets list of the source word and returns its ranking location in the list. The *getSenseCount()* returns the word's number

of senses (sense count). The *findRelationships(sourceSynset, targetSynset, PointerType)* finds the relationships between *sourceSynset* and *targetSynset* based on the PointerType. For example a *pointerType* can be *hypernym* for a *hypernym* relation. The *getDepth()* API returns the depth of a relationship. A depth of relation is the path from the root (source word) to the target word. The larger the depth the less the compared words are related.

2.2 Related Work

Our work is directly related to information retrieval, software reuse, XML schema matching and Web services discovery and matching

2.2.1 Information Retrieval

Information retrieval is the process of searching for information that is relevant to the user needs within a collection of data. There are three information retrieval models [3, 42], the Boolean, the probabilistic and the vector space. The Boolean model is based on the "exact match"; the probabilistic and the vector space models are based on the "best match". Boolean retrieval model returns only fully matched information. The major problem with the Boolean retrieval model is that it is inflexible and unable to rank retrieved information according to their relevance to a query. It does not allow for a form of relevance ranking of the retrieved information. The Boolean model will exclude any information that does not precisely match the requested query [49, 42]
The probabilistic retrieval model [3] uses the statistical distribution of terms in the documents. It calculates the probability of the document being valued and returns a list of the information based on their probabilities.

The vector space model [48, 3] treats text and query as vectors in multidimensional space. The dimensions are the terms used to represent the text. Determining whether information is relevant for a given query requires computing similarity measures between the two vectors. For example, the cosine correlation similarity measures are to calculate the cosine angle between the two vectors. The more similar a vector representing a text is to a query vector, the more that text is relevant to the query. The result of the cosine

correlation is a value between 0 and 1. The value of the correlation similarity is used to rank the retrieved information by relevance. If the similarity value is below a predefined threshold value, the information is considered irrelevant and will not be retrieved.

One of the most used methods for measuring term frequency is the TF-IDF (Term Frequency-Inverse Document Frequency) [42,48]. It is the process of weighting the relevance of a term to a document. The number of times a term t appears in a document d is called term frequency and denoted as $tf(t,d)$. The larger is $tf(t, d)$, the more the t is related to document d. The times that the term t appears in the entire document is called the document frequency, denoted as df_t. The larger is df_t, the less t can discriminate between documents. Thus, for a given document d, the relevance of a term t to a document d is proportional to $tf(t,d)$, and inverse proportional to df_t.

2.2.2 Software Engineering

The software components retrieval have leveraged the searching process to a new level by not only searching based on keywords, but also matching software components for their reuse. Two software components are compared to determine whether one component can be substituted for another.

Luqi L. [34] has suggested that formal specification is suitable as basis for the retrieval and the reuse of software components. J. Jeng and H. Cheng [27] presented a foundation for using software specification matching for the retrieval of reusable software components. They defined an exact match, a relaxed match and a logical match at component and method levels.

H. Cheng and Y. Chen [6] established a semantic foundation to reason about the connection between a specification match and its usefulness for determining software reusability. They showed that the relaxed plug-in match is the most general reuse-ensuring match.

Zaremski and Wing [58, 59] defined and used a formal specification to describe the behaviour of software components to specify when two software components are related. They have presented a signature matching to locate software for reusable components. They considered function and module components, and defined function matching and module matching. The signature of the function is its type and the signature of the modules is a multi-set of user defined types and multi-set of function signature. For both function match and module match the exact and the relaxed match were considered.

2.2.3 XML Schema Matching

XML schema matching is a process of finding the correspondence between elements in schemas. It plays a crucial role in many application such XML schema integration and XML message mapping. XML schema matching is challenging problem due to the flexibility of XML schema itself. XML schema allows identical concepts to be described differently.

XML schema matching should consider both the syntax and semantic of the schema. The syntax of the schema includes the structure and the data they describe and the semantics includes the meaning of the data they describe [43]. The relations between names often involve pre-processing such as tokenization and auxiliary resources such as finding synonyms using dictionary. However, the structural relations vary according to how the schemas are presented. XML schema is usually modeled as a graph or tree, then graph or tree matching algorithms are used to find the structural correspondences.

A. Nierman and H. V. Jagadish [41] defined a tree edit distance-based measure that computes the structural similarity between two XML documents. The distance measure is utilized using different operations to transform one tree to another. The operations are *Relable*, *Insert, Delete, Insert Tree, and Delete Tree*. The edit distance between two trees is the sequence of steps that can be applied to transform one tree to anther. The operations are limited to sub-trees that were originally contained in the source or destination tree. A tree that has been inserted via Insert Tree may not have additional node inserted and a tree that has been deleted may not previously have had a node deleted. The cost of

transforming a source tree to destination tree via the transformation operations determines the similarity between the two trees. The lower is the cost, the more similar the two XML trees are in term of structure.

CUPID [35] is a general schema-matching model that is meant to match schemas such as relational schema and XML schema. Schema is considered as a set of elements that can be tables or columns in relational schema, or elements and attributes in XML schema. The matching process in CUPID covers both the linguistic matching and the structure matching. Linguistic matching is primarily based on the schema element's name. The linguistic matching includes normalization of schema elements by considering abbreviation, acronyms, punctuations using tokenization and expansion techniques. Elements are clustered into categories based on their names and types. A thesaurus is used to compare elements' similarity based on their synonym and hypernym relationship. However, a pre-match effort is needed to specify domain synonyms and abbreviation [8]. The structure matching is based on a tree structure. Two elements are similar if their leaves are similar, and the similarity of their leaves increases if they have similar ancestors. The process of matching is based on a bottom-up approach that pays more attention to the leaf elements.

COMA [7] is another hybrid system aimed to be a general-purpose schema matching. Similar to CUPID, COMA matching process includes linguistic and structural aspects of the schemas. However COMA combines a set of matchers to perform different schema matching. It maintains a library of different matchers that can be combined to produce the complete result. A new matcher can be added to the library. The matching process can result in multiple matching candidates based on the correspondence between the schema elements. The final choice depends on the user. COMA can also perform one or multiple iterations that can be combined with user feedback to improve the matching result. COMA currently supports three kind of matcher namely simple, hybrid, and reuse oriented. Each of these matchers exploits different parts of the schema information to determine the schema similarity.

LSD [9, 10] is a matching system that uses matching-learning techniques to match new schema to previously determined global schema. The user supplies the mapping from data source to the global schema. The pre-processing step looks to the data source to train the learner. The source data is the set of the schema needed to be matched. The learner is an object that can remember the pattern and the rules of matching which can be applied to match other data source. The idea is that, after the learner has been trained, it will have enough information to map subsequent data sources. There are several learners defined in LSD, each of which processes certain type of information from the schema.

Recent work on XML schema matching is the tree-matching algorithm introduced by Ju Wang [61]. The aim of his work was to match XML schemas with a large number of nodes. In addition to the mapping between tree nodes, his algorithm restructures the tree by identifying the approximate common substructure in the two trees. This common substructure is derived from a sub-tree by deleting a node. Consider the following sub-structures:

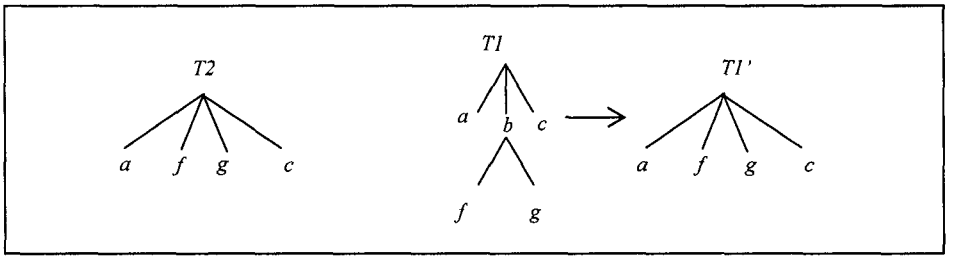

Figure 11: Common Sub-Structure

The sub-structures *T1* and *T2* are compared, the node *b* in *T1* is removed and its children become children of its parent node producing the structure presented by *T1'*. Matching *T2* with *T1'* will of course produce higher similarity than matching *T2* with *T1*. However the structure of *T1* has been changed largely to increase the similarity score. In addition to restructuring the compared trees, the algorithm does not consider any of an XML schema's order indicators in the matching process. The goal behind restructuring the tree and ignoring the order indicators was to increase the overall similarity. This approach pays less attention to the structure similarity and the execution time to increase the overall similarity.

2.2.4 Web Services Discovery

Recently, a considerable amount of research has been devoted for building a rich semantic description for Web services to enable effective and efficient discovery. A discovery of Web services consists of semantic matching between the description of the requested service and the description of the advertised services. The semantic description of Web services is modeled using ontologies to represent concepts in Web service and their relations. The Web service ontology defines a semantic Web service that describes the capability, the conditions, and the restrictions of a Web service. The ontology description usually attempts to build a Web service description language that is expressive, clear, flexible, and extensible. It would include functional capabilities, non-functional capabilities and information about the domain of the Web service. The following are semantic Web service frameworks developed to support Web service discovery:

DAML-S[1,2], OWL-S [36] are a formal language that supports the specification of semantics information in RDF [71] format. It is designed based on a set of a domain-specific semantics of ontologies. DAML-S is meant to support Web services discovery, invocation and composition under specific constrains. It characterizes the service as profile, model, and grounding.

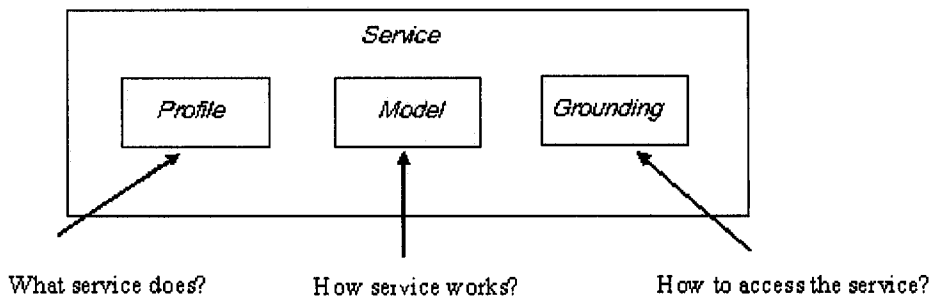

Figure 12: DAML-S Architecture

The service profile describes what the service does. It describes the functional and non-functional properties of the service including input types, output types, pre-condition, post-condition, name, and quality of services. The service profile is actually a summary

of the descriptions provided in the service model and service grounding. The service profile is intended for the purpose of advertisement; it includes only the functionality that is publicly provided. It includes three types of information: a text description primarily for the use by a human use, a functional description defines what the service provides and the conditions that have to be satisfied in order to successfully use the service, and functional attributes' address and properties. The attributes' address and properties are used to include information about the service other than the functional information (e. g. geographic scope, quality guarantees). The service mode describes how the service works. It defines what happens when the service is being executed. It is comprised of two components; process ontology and process control ontology. The primary entry of the process ontology called process. A process is a class that has input, output, preconditions and effect. The process control ontology is intended to monitor and control the execution of a process. Service grounding specifies how a Web service can be accessed. It provides information such as communication protocols and specifies details such as port number. The correspondence between profile, model, and groundings is not necessarily one-to-one, however, there must be at least one grounding. DAML-S is still immature and not supported by current tools and the cost of formally defining the services makes its adoption unlikely [62].

IRS-II [39] is a framework aimed to support heterogonous Web services publication, discovery and composition. It provides a publishing support, a client API, brokers and registry mechanism. IRS-II is based on UPML (unified problem method development language) [12]. The UPML framework is structured as classes of components where each class is described by means of ontology. A domain model describes the domain of an application such as vehicles, a medical disease. A task model provides a generic description of tasks to be solved such as input types and output types, the goal to be achieved and the pre-conditions. The problem solving methods provide abstract implementation-independent descriptions of reasoning processes, which can be applied to solve tasks in specific domains. The bridge specifies the mapping between different model components within an application.

In general IRS_II is comprised of three main components: IRS-Server, IRS-Publisher and IRS-II Client. These components are communicating using SOAP protocol.

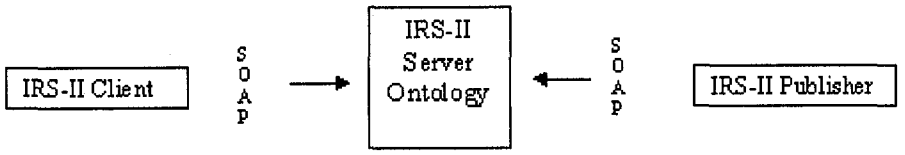

Figure 13: IRS_II Architecture

The IRS_II server contains the semantic Web service description. It provides two levels of descriptions: knowledge base level where the description is stored using domain model, tasks model and problem solving methods. The IRS-II publisher links the Web service to the semantics description inside the IRS_II server. Web services can be published using IRS-II java API where the developer has to specify the location of the IRS-II server via a host and port number and the problem solving methods using service name and ontology. The IRS-II client provides an interface for Web services invocation. The invocation process is achieved by asking the IRS-II client for a task to be located and invoked by the IRS-II broker.

WSMF [13] Web Service Modeling language provides a conceptual model that describes a Web service. WSMF is organized around two principles: strong decoupling of the components of e-commerce application and strong mediation. The strong decoupling is achieved via interfaces to keep the amount of interactions scalable. The strong mediation is to enable vast communication of Web services. WSMF consists of four main elements: ontology, goal repositories, Web service, and mediators. The ontology provides the definitions of terminologies used by the other elements. It defines formal semantics for terminologies to enable the reuse of these terminologies. The goal repositories define the problem solved by the Web service. This is generally what the client has when searching in a Web service. The goal repositories consists of pre-conditions that describe what the service requires to be executed; post-conditions describe what a service return as a response to the client input. A mediator is used to solve the interoperability of the Web

service. For example, a mediation of dynamic service invocation is when a Web service invokes another Web service to provide its functionality.

WSDA [23] is Web service discovery architecture that defines Web service discovery layer for describing interoperable interfaces, operations, and protocol binding. It is described in SWSDL [24, 25]. SWSDL is simplified version of WSDL. It provides a service as a set of related service interfaces. Each interface has an interface type which defines a set of operations and arguments. An operation is bound to one or more protocols and network endpoints via binding definitions. For example, a service can be structured as the following:

```
<service>
    <interface type = "http://gridforum.org/interface/Scheduler-1.0">
      <operation>
          <name>void submitJob(String jobdescription)</name>
          <allow> http://cms.cern.ch/everybody </allow>
          <bind:http verb="GET" URL="https://sched.cern.ch/submitjob"/>
      </operation>
    </interface>
</service>
```

Figure 14: SWSDL Service Description

The service is a scheduler type and its syntax and semantics of operations are specified at the location defined by the type attribute of the interface element. The name element defines the operation name as *submitJob* and its parameters of type *string*. The bind element specifies that the operation is bound to HTTP protocol. A service is identified by a URL and retrieved using HTTP Get request to the identifier.

WSAD includes service descriptions, service identification, and a query support. It supports XML data model for heterogonous content interaction. It defines four types of interfaces: presenter, consumer, minQuery and XQuery. The presenter interface allows a client to retrieve services. It defines an identifier for the services to be retrieved and a service description that is associated with the identifier. The identifier is described by URI and the retrieval mechanism is HTTP protocol. An HTTP request to the identifier will return a service description. The service description can be bound to a protocol to

connect to the service. The consumer interface works like a registry service. It allows the provider to publish a tuple set to the user. The tuple set comprise of a set of attributes normally link, type, context, time stamp and metadata. The link is a URI to the service description. The type describes what kind of context is being published. A time stamp defines the *tuple* lifetime. The metadata may describe any other information that has not been described by the attributes such as retrieval from UDDI. The *minQuerry* interface supports a query in select-all style. For example, *getTuple()* will return all tuples. The *XQuerry* interface supports the XML query language [81]. The presenter, consumer and *minQuerry* are bounded to HTTP protocol and the *XQuery* interface is bound to peer database protocol [24, 25].

SCDL [15] describes a capability matching for Web services. The structure of SCDL comprised of a set of elements, including *name* that defines the name of the Web service. *Ontology Description* is used to describe the used terms. *Types* are used to define the variable types. *Input* for declaring input variables and *output* for declaring the type of the output variables. *Pre-conditions* and *pre-constrains* to describe the conditions and constrains have to be imposed on the input variables. *Post-condition* and *post-constrains* to describe the conditions and constrains have to be imposed on the output variables. This structure is described using XML schema. Two specifications in SCDL are plug-in matched if their signatures match. Their signature match if there is a sub-sumption relation for every clause in the set of input conditions constrains of one specification and a clause in the set of input condition constrains in the other specification, and there is similar sub-sumption relation for their output condition constrains.

Bianchini Davi [5] Described ontology based methodology for e-service discovery. Their methodology supports both the publication phase and the searching phase. Their approach is designed to be fully compatible with UDDI in a way users can either use the UDDI API or the API provided by their approach. A service context is defined in term of location, time zone, and available channels in both location and time. Channels are characterized by *device*, and *network* for defining end-to-end link, *network interface* for defining how a device could be connected to the network, and *application protocol*

specifies the application protocol is supported by the device according to the network and the network interface. The functional description of the service can be defined according to WSDL. The non-functional description of the web services is characterized by a set of *Quality Parameters.* For example, a video on demand Web service would define parameters as frame-rate and colour depth. The searching process is described by a semantic analysis of the service functionality, context and quality. The functionality similarity is done using interface similarity analysis and behaviour similarity analysis. The interface similarity analysis is preformed through measuring the similarity between the set of coefficients defined in the interface. If WSDL is used to represent the interface specification, the interface analysis will compare all operations and their input and output parameters' information. A behaviour similarity analysis describes each major functionality provided by the service is associated with a state-chart diagram. The state-chart diagram describes how the execution of the service is preformed. The context similarity considers the comparison of location, time zone, and channel constrains. The quality similarity considers comparing the characteristics of quality of service parameters.

The thesis is based on LARKS, and agent matching system [54]. Larks is a matching agent that uses a sequence of filters based on specific models that perform both syntactic and semantic matching. The process of matching uses different filters to narrow the set of matching candidates. A context matching filter matches software agents based on their context. A profile comparison filter matches software agents based on their text using TF-IDF method and vector space model. A similarity matching filter matches software agents based on their semantic. A signature matching filter matches software agents based on their input and output parameters. A constrain matching filter matches software agents based on their pre-conditions and post-conditions.

Whiles the above approaches are promising to revolutionize Web service discovery by providing the rich formal descriptions, they are still immature and not supported by current tools and industrial community.

The UDDI project [75] is founded by IBM, Microsoft, and Ariba and currently more than 200 organizations are sponsoring the project [74]. Public UDDI operators are currently managed by IBM, Microsoft, HP, and SAP. Three versions of UDDI has been released so far; version 1.0 was released on September 2000, version 2.0 was released on June 2001 and version 3.0 was released on July 2002. UDDI uses XML, SOAP, WSDL and HTTP standards to provide a standard mechanism to publish and to locate a Web service.

UDDI framework consists of a registry containing the Web service information. It is organized around two specifications: the information specifications and the API specification. The former defines the data structure, while the latter defines the API for inquiring and publishing Web services. The UDDI data structure is comprised of four entity types: <businessEntity>, <businessService>, <bindingTemplate> and <tmodel> known as technical model. Service provider uses these entities to register information about the offered services. The type of information registered in UDDI registry is commonly known as white pages, Yellow Pages, and Green pages.

- White pages: contains basic business contact information. It allows to discover the services based on the contact information
- Yellow Pages: contains basic information that categorizes businesses. It allows others to discover Web services based on their category.
- Green pages: contains technical information about the offered services.

This information is represented in UDDI as an XML structure with the *businessEntity* element as top-level element. The *bussinesEntity* element describes a business that provides a Web service. It contains contact information, set of services description, and technical information. The services description is defined by one or more *bussinessServices* element. Each *businessServices* element represents a service description, name, category and technical description. The technical description is represented by one or more *bindingTemplate* entities. It consists of technical information about service entry point. In addition to the technical description, each *bindingTemplate* has a reference to *tModel* entities. The *tModel* entities are used to describe the behaviour of the service, what standards it follows, what specifications the service complies with, and how to invoke the services. It consists of related information that facilitates

communication between a client and a Web service. The *tModel* is also includes an entry pointing either to the service URL or to WSDL file description.

Each of the above core entities has a key that can be used to retrieve information about the corresponding entity.

The API specification is divided into two sections:

- The publisher API and,
- The inquiry API

The publisher API allows the service provider to access the UDDI registry to manage the information advertised about their business. It provides the functions required to create, update or delete service information. The inquiry API allows the user to locate and obtain information about an entity. It supports three pattern of inquiry: browse pattern, drill-down pattern, and invocation pattern.

The browse pattern starts with some general information, performing a search, and results in a huge list of records. This search is usually followed by drill-down pattern to select more specific information. The drill-down pattern requires prior knowledge of a core data structure entity key (one of the values returned by the browser pattern).

Passing an entity key as search criteria retrieves detailed information about the corresponding entity. The invocation pattern is used in case of failure in the service invocation.

Locating Web services in UDDI registry is largely based on a single search criterion. A potential user must identify a keyword such as business name, service name, or business location to extract information out of the UDDI registry. The search process generally starts with the browse pattern to extract general information, followed by a drill-down pattern to find specific detailed information.

Some research work has been focusing on WSDL description to build Web service searching systems. W. Yiqiao and S Eleni [63, 62] have described a method for web services discovery and matching that combines the structure and the semantic information of WSDL file. They defined a keyword search using vector space model and structure similarity based on the tree-edit distance algorithm [16]. In their approach, WSDL is

viewed as hierarchal structure with the data type lie in the lowest level of the hierarchy. They adopted a bottom-up approach where the similarity of two WSDL files starts by comparing their data types. The result of data types' similarity is a matrix of all possible pair-wise combinations of source and target of data types. The second step is to compare the source messages to the target messages. The result is matrix of all pair-wise combinations of source and target messages scores. The similarity of messages is based on the similarity of their parameters scores. The third step is to compare the services' operations. The result of operations similarity is based on the pair-wise combinations of the source and target operations. The similarity of two services is based on computing the pair-wise correspondence of their operations that maximize the total sum. Data types are compared based on their compatibility. Two data types are considered compatible with score of ten, semi–compatible with score of five or non-compatible with score of zero. If data types being compared are complex types, their elements are collected to produce lists of simpler data types. The total score is the highest matching score of their elements. If the data types being compared have the same grouping style a bounce score of ten is added to the total score.

The main drawback of this approach is that it compares all possible combination of data types. It does not distinguish between output data types and input data types. For example if there is an operation A with input data type as X and output data type as Y and operation B with input data type as Z and output data type as W, data type comparison will match all pairs (X, Z), (X, W), (Y, Z), and (Y, W).

In addition to comparing all pairs of data types and messages which is not required, the algorithm does not consider most of data type syntax such as maximum occurrence, minimum occurrence, sequence indicator, choice indicator, grouping and It does not compute the similarity between data types from different categories such as simple type to complex type.

Xin Dong [11] have described a search engine for Web service (Woogle). Their approach is based on operation search rather than WSDL search. Their algorithm is based on the

classical agglomerative clustering [28]. Similarity between two operations is based on the similarity between their vector *op(w, f, I, o)*. Where w is the text description of the Web services to which the operation belongs, f is the textual description of the operation, and *I* and *o*, are the input and output parameters respectively.

Chapter 3: Overview of Our Matching System

Web service technology makes it possible for developers to choose from either building all pieces of their applications or using Web services created by others. An individual organization does not have to supply every piece for a solution. It can compose a Web service from different providers to build the complete solution. A crucial step is to be able to efficiently locate and select Web service. This is particularly important in automatic Web service composition when the output of one service is passed as input to another service. As the Web service paradigm becomes more and more popular; the need for flexible Web service discovery becomes more essential. It is becoming one of the major challenges of Web service technology [4]. The searching process should be flexible enough to return a ranked list of Web services based on their closeness to the query.

In our prototype, a mechanism that includes text similarity and structural similarity of Web services is introduced.

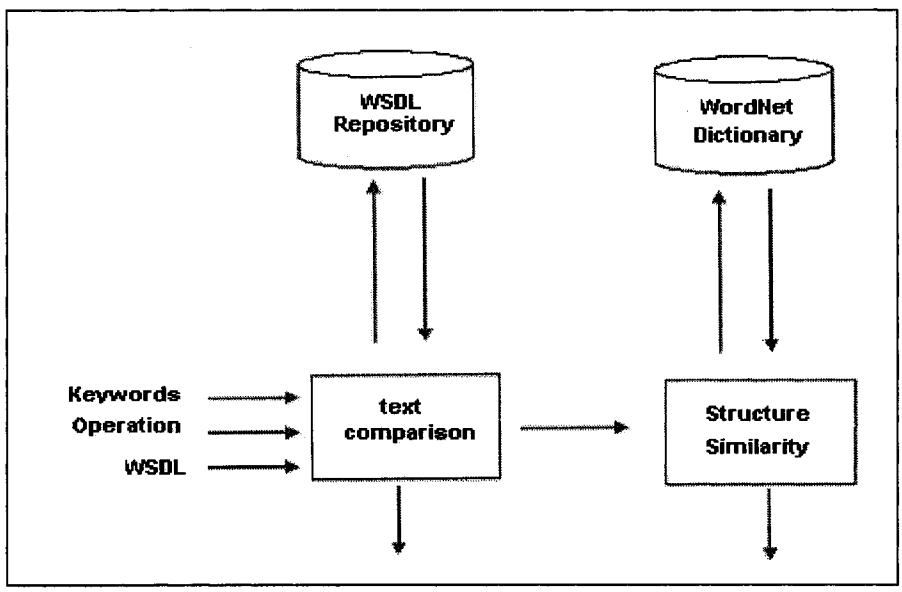

Figure 15: Web Service Searching Framework

A different algorithms are combined to produce flexible, effective and efficient Web searching framework that combines two filtering modes with three searching criteria.

3.1 Searching Criteria

A user can either search for Web services using a keyword search by providing a list of keywords, operation search by providing an operation description, or service search by providing a Web service description.

3.1.1 Keywords Search

The keyword search uses only the text comparison mode described in chapter 4. The query is determined by the keywords obtained from the user input.

3.1.2 Operation Search

In operation search, the query is an operation description obtained from the user input. The structure of the requested operation and the advertised operations are taken into consideration. The user enters information such as operation name, input parameters and output parameters. The search process compares all information provided by the user to all operations in the repository. The similarity is computed based on the similarity mode chosen by the user (Text Comparison or Structure Similarity). The result is a list of operation ranked based on their similarity to the query.

Definition 3.1 (operation search)

Request operation o, advertised Web Services W $\longrightarrow$ *List of similar operation*

$sim\ (o,\ W) = \{o' \in W : sim(o,\ o')\} \rightarrow L$

Given a request operation *o* and Web services collection W the searching returns L, a list of all operations similar to the query operation.

3.1.3 Service Search

The query is a URI pointing to the location of the Web services description. The system compares the requested service to all services advertised in the repository. The similarity

is computed based on the similarity mode chosen by the user (Text Comparison or Structure Similarity). The result is a list of similar Web services are ranked based on their closeness to the query.

Definition 3.2 (web services searching)

Request Web service w, advertised Web Services W $\longrightarrow$ *List of similar Web services*

$sim\ (w, W) = \{w' \in W : sim(w, w')\} \rightarrow L$

Given a request Web service *w* and Web Services repository W, the searching returns a list of Web service that are similar to the requested Web service.

3.2 Filtering Modes

The similarity filtering modes are organized as two increasingly stringent filters. Each filter narrows the set of matching candidates with respect to a given filter criterion.

3.2.1 Text Comparison

The text comparison filter measures the similarity of a request to advertisements based on the vector space model [48]. The vector space model is based on building *n* dimensional vectors for the query and the distinct terms in each candidate service. The query and the collection of services are transformed to text. They are tokenized, stemmed and their stop words are removed. The relevance of a document to a given query is based on computing a distance measure between the query and the document using the cosine similarity measure.

3.2.2 Structure Similarity

The structure similarity computes the similarity between the query and the advertised services based on the structure of their corresponding elements. The structure similarity returns operations and services that are similar in some way to the advertisements and hence would match if the request is slightly modified. There are two forms of operation similarity: operations similarity and partial operation similarity. Operation similarity

specifies all the information required for search. In partial operation similarity, the user can specify only a subset of the required information. For example, users may be more interested in the type of the output. Allowing users to define an input that can be matched to any input in the advertised operations is more useful in this case. Depending on the search criteria, the structure similarity will return a list of operations or Web services with a score value between [0, 1] describing how close the result is to the query. As the computational cost of the structure similarity is high, only Web services returned by the text comparison are passed to the structure similarity.

Chapter 4: Text Comparison

4.1 Introduction

The goal of the text comparison is to use the information retrieval techniques to measure the relevance between a query and documents. These techniques rely on unstructured text description of the query and the documents. The process of relating a query to a document includes analyzing the statistical information about terms that appear in the documents and how these terms are related to the query. This statistical information is used to compute a weight for each term based on the frequency of a term in an individual document and in a collection of documents. The weight measuring involves identifying terms in a stream of text by pre-processing documents using tokenization, removing stop words and stemming. After the pre-processing phase, all terms in the collection of documents are indexed for fast document retrieval. The index is only needed to be built once, stored on the hard disk and loaded to the memory as needed. The index contains every unique term in the collection of documents. Each term points to the list of documents that contains the term and its frequency in each document. The term frequency is an indication about documents relevance to that term and it is used as base for measuring document relevance.

One of the most used models to calculate the similarity between a query and documents based on the term weighting is the vector space model. The vector space model has been extensively investigated in the literature. The advantages of using vector space model in information retrieval are its effectiveness, efficiency, ranked retrieval, and terms are weighted by importance [18, 19]. In this thesis we use the vector space model for Web service retrieval and filtering.

The main rationale behind using text comparison in Web service searching is to provide fast Web service retrieval mechanism using keyword search and to filter irrelevant Web services before being processed by the structure similarity. The structured similarity requires a significant computational time. As the number of candidates Web services can

be extremely large, using vector space model as filtering method will significantly reduce query processing time.

4.2 Documents comparison

Documents comparison is similar to the most conventional search on the Internet. It measures the relevance of the query to a document using the vector space model. Vector space model is based on building *n* dimensional vectors where the query and each document in a collection of documents are represented by a vector of non-negative terms' weight. Once the terms' weights are determined, the similarity between a query and a document is measured using the cosine similarity function. When the angle θ between two vectors is close to zero the cosine approaches one and when the angle between them approaches ninety the cosine approaches zero. From the law of cosines:

$$cos\ \theta = \frac{\vec{q}.\vec{d}}{|q||d|} \qquad 4.1$$

Where $\vec{q}.\vec{d}$ is the dot product of the query vector $\vec{q}$ and a document $\vec{d}$. If we have two vectors q=(2,3,4,5) and d=(6,7,8,9) the dot product of the two vectors q.d=(2*6)+(3*7)+(4*8)+(5*9) . The |q| and |d| are the absolute values of the query vector and the document vector. The absolute value of $|q||d|=\sqrt{2^2+3^2+4^2+5^2} * \sqrt{6^2+7^2+8^2+9^2}$

Measuring the similarity between the query *q* and the document *d* using cosine function is as follows:

$$sim(q,\ d) = \frac{\vec{q}.\vec{d}}{|q||d|} \qquad 4.2$$

Equation 4.2 indicates that the similarity between a query and a document is the similarity between their vectors, which is equal to the dot product of the vectors divided by their absolute values. The numerator of equation 4.2 can be represented as: *q.d=(wi,d*

$* wi,q) + \ldots + (wn,d * wn,q)$ where wi,d is the weight of a term i in a document d and wi,q is the weight of term i in a query q.

$$sim(q,dj) = \frac{\vec{q} \cdot \vec{d}}{|q||d|} = \frac{\sum_{j=1}^{n} w_{j,q} * w_{j,d}}{\sqrt{\sum_{j=1}^{n} w_{j,q}^2} * \sqrt{\sum_{j=1}^{n} w_{j,d}^2}} \qquad 4.3$$

Where j is a term in n collection of terms. The importance of j in a document d, dependence on its statistics in d and its statistics on the entire collection of documents D. Assigning a weight for each unique term in each document determines the relevance of the term to the document. The most useful and widely used term weighting method is the TF-IDF (term frequency-inverse term frequency), which is entirely being based on a single term statistics. Given a document $d \in D$ where D is the set of document in the repository. Let $t_1, \ldots, t_n$ be terms occurring in the document d. The number of times a term t occurs in the document d is called the term frequency $tf(t,d)$ of the term t in the document d. The number of documents in which the term t occurs at least once is called document frequency $df(t)$ of the term t. The relevance of a document d based on a term t is proportional to the number of times the term t occurs in the document d and inverse proportional to document frequency $df(t)$ of the term t. The larger is $tf(t, d)$, the more likely the t is related to document d. The larger is $df(t)$ the less t can discriminate between documents

In the TF-IDF weighting method, the weight of a term t in a document d is of the form

$$Wt,d = tf(t,d) * idf$$

Where the idf is the inverse term frequency and it is computed as follows:

$$idf = \log D/df(t)$$

$$Wt,d = tf(t,d) * \log D/df(t)$$

If a term appears in every document, its inverse document is equal to zero. For example if the number of documents in the entire collection is 1000 and the number of documents contains a term *t* is 1000, then the inverse document is equal to log 1000/1000=0. The idf of a term is constant cross documents collection and needed to be calculated only once. The denominator of equation 4.2 is called the cosine normalization factor. It normalizes the length of documents such that document length has no effect on the similarity score [32]. Other normalization techniques are the Maximum *tf* Normalization and the Byte Length Normalization [51]. The Maximum *tf* Normalization modifies each term tf in the document by the maximum term max-tf in the document. Since some of the resulted values are low, the normalized values are usually recalculated. For example the Smart system increases the tf factor as $(0.5+0.5*\frac{tf}{\max-tf})$ and the INQUERY system as $(0.5+0.6*\frac{tf}{\max-tf})$ [51]. The Document Length Normalization modifies the term weighting based on document size [52]. For example the weight value can be calculated as $(\frac{tf}{ndl})$ where *ndl* is the normalized document length *ndl=document length/ average documents length* [14].

4.2.1 Web Service as Document

Representing Web services and the query as text documents will enable using the vector space model for Web service searching. A keyword search can be used for fast Web service retrieval. An Operation and a Web service filtering can be used to prune off the irrelevant Web services to reduce the computational time required by structure similarity described in the next chapter.

To determine Web service relevance to a given query, the query and the Web service are converted to text documents. They are tokenized, stemmed and their stop words are removed. The tf-idf weighting method is applied for each term in the query and in the collection of Web services. Then, the query and the advertised Web services are represented as vectors. The similarity between the query vector and the Web services

vectors are determined by the angle between the query vector and each vector of the Web services.

Example 1:

Consider the following portion of Web services as the collection of Web services in the repository:

- **Temperature unit converter service:**

```
<portType name="ChangeTemperatureSoap">
  <operation name="ChangeTempUnit">
      <input message="ChangeTempUnitSoapIn" />
      <output message="ChangeTempUnitSoapOut" />
    </operation>
</portType>
```

- **Energy Unit converter service:**

```
<portType name="EnergyUnitSoap">
  <operation name="ChangeEnergyUnit">
      <input message="ChangeEnergyUnitSoapIn" />
      <output message="ChangeEnergyUnitSoapOut" />
    </operation>
</portType>
```

- **Currency converter service:**

```
<portType name="CurrencyConvertorSoap">
    <operation name="ConversionRate">
        <documentation><br><b>Get conversion rate from one
    currency to another </documentation>
        <input message="ConversionRateSoapIn" />
        <output message="ConversionRateSoapOut" />
  </operation>
</portType>
```

Consider running the following query on the above collection:

- **Query:** "temperature unit converter".

4.2.1.1 Tokenization

The first step in documents processing is the tokenization. Tokenization separates the tokens of a compound word in such way that every individual token is identified and treated as atomic unit (separate term). It separates compound words based on punctuation marks, abbreviation, and case. For example the string "getLatestStockValue" is tokenized to "get" "Latest" "Stock" "Value". There is no general agreement on how documents are tokenized. It is usually depend on the underlying applications [21]. As Web services are described using WSDL, which is an XML structure, a lot of information is not considered in the tokenization process. For example, tag names, namespace values and attributes names are ignored. This information is ignored because it occurs in all WSDL files and has no effect on the search result. Failing in removing this information will increase the index size. A tokenizer is implemented to parse all strings in the query and the advertised Web services.

Tokenizing the WSDL portions and the query presented in example 1 will produce the following texts:

- **Temperature unit converter:** "change temperature soap change temp unit change temp unit soap in change temp unit soap out"
- **Energy Unit converter:** "energy unit soap change energy unit change energy unit soap in change energy unit soap out"
- **Currency converter:** "currency converter soap conversion rate get conversion rate from one currency to another conversion rate soap in conversion rate soap out"
- **Query** "Temperature unit converter"

Note that tags names and attributes names are ignored. For example "<portType" and "<operation" do not appear in the tokenized text. Removing these keywords by the tokenizer is more efficient than adding them to the stop word. Also note that all terms are transformed into lower case

4.2.1.2 Removing Stop Words

Once the query and the advertised Web services are converted into a sequence of tokens, stop words are removed. The stop words are words that are meaningless and merely noise and can be eliminated without affecting the accuracy of the retrieval process. Usually English text is composed of the same few words and some of these words may not be useful for Web service retrieval process. Removing the stop words also reduces the index size and thus increases the indexing process. If we consider the following set of terms as part of the stop words {in, out, another, from, one, to}. The tokenized version of example 1 can be presented as the following:

- **Temperature unit converter (with 15% length reduction):** "change temperature soap change temp unit change temp unit soap change temp unit soap
- **Energy Unit converter (12% length reduction):** "energy unit soap change energy unit change energy unit soap change energy unit soap"
- **Currency converter (with 28% length reduction): "**currency converter soap conversion rate get conversion rate currency conversion rate soap conversion rate soap"
- **Query** "Temperature unit converter"

In this thesis, the stop words list provided by the Department of Computing Science at the University of Glasgow [65] was used. Note that this list has been modified to include terms that are related to Web service description.

4.2.1.3 Stemming

After removing the stop words the query and the documents are stemmed for term normalization. Stemming is the process of removing morphological variants and suffixes from terms (e.g. "ing","ed"). In literature several types of stemmers have been developed. Two of the most popular stemmers are Lovins, 1968; Porter, 1980. Both Porter stemmer [45, 46] and Lovins stemmer [33] are similar, however Porter stemmer is intended to reduce the number of processing steps in Lovins. Porter stemmer consists of five steps; for example step one deals with plurals and past participles such as removing 's', 'ies'

and "ed". Since the algorithm is performed in steps, it is possible that one term is transformed by more than one step. If a term has less than four letters no stemming is performed. For example, using Porter stemmer the terms "transform", "transformation" and "transforming" are stemmed to "transform". As both the query and the documents are stemmed, all terms with the same root as the query are returned. In this study Porter's stemmer has been implemented.

After removing the stop words, the stemming form of example 1 can be presented as:

- **Temperature unit converter:** "chang temperatur soap chang temp unit chang temp unit soap chang temp unit soap

- **Energy Unit converter:** "energi unit soap chang energi unit chang energi unit soap chang energi unit soap"

- **Currency converter:** "currenc convert soap convers rate get convers rate currenc convers rate soap convers rate soap"

- **Query** "Temperatur unit convert"

Note that all terms have been converted to their roots. For example the term "temperature" in the temperature unit converter document and the term "temperature" in the query are both converted to "temperatur"

4.2.2 Representing Web Service as a Vector

Assume we have the three documents presented above "Temperature unit converter", "Energy Unit converter" and "Currency converter" and we would like to use the vector space model to search for the query "temperature unit converter".

The total unique terms in the collection is eleven {chang, temperatur, soap, temp, energi, unit, currenc, convert, convers, rate, get}. This will produce vectors of length eleven.

The following table is used to calculate the weight of each term based on its frequency in a document and in the collection of documents.

terms	tf							wi=tf*idf			
	q	d1	d2	d3	dfi	D/dfi	idf	q	d1	d2	d3
chang	0	4	3	0	2	3/2	0.1761	0	0.7044	0.5283	0
temperatur	1	1	0	0	1	3/1	0.4771	0.4771	0.4771	0	0
soap	0	3	3	3	3	3/3	0	0	0	0	0
temp	0	3	0	0	1	3/1	0.4771	0	1.4314	0	0
energi	0	0	4	0	1	3/1	0.4771	0	0	1.9085	0
unit	1	3	4	0	2	3/2	0.1761	0.1761	0.5283	0.7044	0
currenc	0	0	0	2	1	3/1	0.4771	0	0	0	0.9542
convert	1	0	0	1	1	3/1	0.4771	0.4771	0	0	0.4771
convers	0	0	0	4	1	3/1	0.4771	0	0	0	1.9085
rate	0	0	0	4	1	3/1	0.4771	0	0	0	1.9085
get	0	0	0	1	1	3/1	0.4771	0	0	0	0.4771

Table 1: Term Weight Calculation Based on TF-IDF Method

Note that the larger the number of documents contain a term, the less the idf of the term. As the term "soap" appears in all documents, its idf is zero and consequently its weight is zero.

From table 1, the three documents and the query can be presented as vectors of eleven elements describing the weight of each term.

Query: (0, 0.4771, 0, 0, 0, 0.1761, 0, 0.4771, 0, 0, 0)

Temperature unit converter: (0.7044, 0.4771, 0, 1.4314, 0, 0.5283, 0, 0, 0, 0, 0)

Energy Unit converter: (0.5283, 0, 0, 0, 1.9085, 0.7044, 0, 0, 0, 0, 0)

Currency converter: (0, 0, 0, 0, 0, 0, 0.9542, 0.4771, 1.9085, 1.9085, 0.4771)

$|q| = \sqrt{0.4771^2 + 0.1761^2 + 0.4771^2} = 0.6973$

$|d1| = \sqrt{0.7044^2 + 0.4771^2 + 1.4314^2 + 0.5283^2} = 3.052$

$|d2| = \sqrt{0.5283^2 + 1.9085^2 + 0.7044^2} = 4.417$

$|d3| = \sqrt{0.9542^2 + 0.4771^2 + 1.9085^2 + 1.9085^2 + 0.4771^2} = 8.6505$

$q.d1 = 0.4771 * 0.4771 + 0.1761 * 0.5283 = 0.3206$

$q.d2 = 0.1761 * 0.7044 = 0.1240$

$q.d3 = 0.4771 * 0.4771 = 0.2276$

$sim(q,d1) = 0.3206/(0.6973 * 3.052) = 0.1506$

$sim(q,d2) = 0.1240/(0.6973 * 4.417) = 0.04$

$sim(q,d3) = 0.2276/(0.6973 * 8.6505) = 0.03$

The vector space model will sort the documents in decreasing order of their relevance to the query as:

1. Temperature unit converter
2. Energy Unit converter
3. Currency converter

Note that only portion of the Web services is used in the above example. In real application the complete document is considered. The query is either a keywords for a keyword search, operation description for operation search or a Web service description for Web service search. The Web services retrieved by the operation search and Web service search are passed to the structure similarity for further refinement.

4.3 Experiment Design and Result Analysis

This section presents the experimental design and the results analysis for evaluating the vector space model for Web service retrieval. It is important to emphasise that our objective is not to evaluate the vector space model. Our objective is to evaluate the use of vector space model for fast retrieval of Web services and for Web services filtering. Our experiments, study the effectiveness of the system in retrieving documents relevant to the user query.

The most common way to evaluate an information retrieval system is to measure how many relevant documents have been retrieved and how early in the ranking they were listed. The most used technique is the recall and the precision measure. In the ideal case, when all and only all the relevant document retrieved, the precision=recall=100%. However recall can be easily maximized by returning all possible documents. On the other hand, precision can be maximized by returning only few related documents. Measuring the precision and recall requires that documents are either relevant or irrelevant to the query. Human interaction required in determining a set of queries and which documents in the collocation is considered as relevant to specific query.

The vector space model is investigated based on effectiveness and time and space efficiency. The effectiveness is measured based on the type of documents retrieved with respect to a given query. It measures whether the retrieved documents are relevant to the query and whether all the relevant documents are retrieved.

In particular we are investigating the following points to evaluate the system:

- Measuring the recall and precision
- Top-K precision (measuring how early the relevant Web services appear in the ranked result)
- Response time to the query
- Time required for the pre-processing of the candidate Web services (tokenization, removing stop word, stemming, indexing)
- Size of the index compared with the original size of the Web services collection.

The precision and recall measures assume that the set of documents are either relevant or irrelevant. If we donate the set of relevant documents retrieved as A and the set of irrelevant documents retrieved as B, and the set of relevant documents that are not retrieved as C, the precision =A/(AUB) which is the ratio of relevant documents retrieved to the total number of retrieved documents . The recall = A/(AUC), which is the ratio of relevant documents retrieved to the total number of relevant documents in the collection. This is illustrated in figure 10:

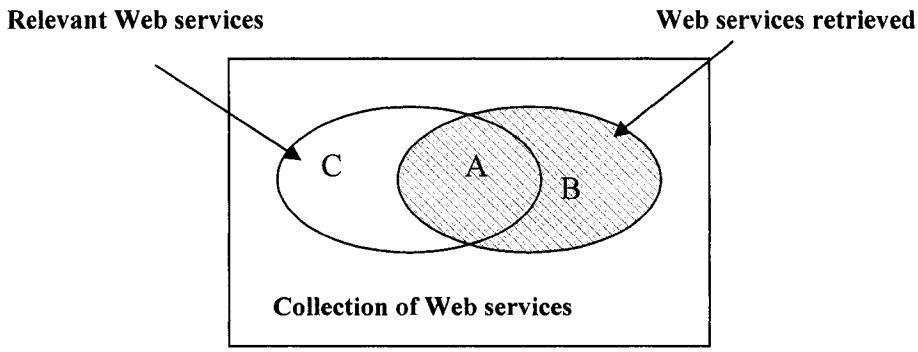

Figure 16: Precision and Recall Diagram

The precision and recall are inversely related, such that when the recall goes up the precision goes down and when the precision goes down the recall goes up. If the goal of a search is comprehensive retrieval (includes general terms), then we should be looking for higher recall, which consequently produces low precision.

Another way of evaluating information retrieval systems based on the precision and recall is to measure the precision at Top-k recall. For example, measuring the precision at k equal; 1, 5, 10 and 25 percent recall points. The top-k precision is calculated by considering only the top-k as returned value. For example, if the returned list is 100, a top 10% will measure the precision at the top returned 10. If all the top 10 are related to the query, the precision is equal to 100%. This approach measures how many relevant documents appear at the top of the returned result. It assumes that the user is interested in looking at the top k documents for a particular query. For all experiments we measured the recall and precision, and top-k precision for recall points; Top-1%, Top-5%, Top-10%, Top-25%, Top-50%, and Top-75%.

In order to thoroughly evaluate the system, we ran three kinds of experiments. The difference between these experiments is the query structure. The first kind of experiments uses keyword as a query. The second kind of experiment uses an operation description as query and the third kind of experiments uses Web service description as query.

All of our experiments are preformed on a computer with single x86 Family 6 Model 6 Stepping 2 AuthenticAMD ~1.539 GHz CPU and 753,136 KB RAM. The operating system is Microsoft Windows 2000 Professional. All programs are developed using Java J2RE 1.4 "j2re1.4.2_04"

4.3.1 Data Collection

A considerable amount of Web services have been used in evaluating the system. These Web services have been collected from a variety of resources. The data domain of the collection contains over 1,400 Web services description documents collected from over 900 hosts. The size of the collection is over 18 MB. The Web services cover various domains such as stock quotes, unit converters, weather forecast, currency exchange, etc. We have used the weather category as a base for our experiments. All queries are formulated to be related to weather services. The weather category contains 17 Web services presented in appendix A.

Note that operations and Web services exceeding a specified threshold will be passed to the structure similarity measure described in the next chapter.

4.3.2 Performance

Results have shown that the time required for the pre-processing of the candidate documents (tokenization, removing stop word, stemming, indexing) was 42204 Millisecond. The size of the index was 2.63 MB. The length of the index was 9,589 unique terms. We ran the following three kinds of experiments:

4.3.2.1 Keyword Search

The effectiveness of keyword search was evaluated using the term "weather" as a query. The keyword search has achieved precision=recall=100%. The response time to the query was 15 millisecond.

4.3.2.1 Operations Searching

In the operation search, the query is an operation description. The query is first transformed to text and than matched to all Web services in the repository. The result exceeding a threshold is passed to the structure operation similarity for further refinements. Three operations each from different weather Web service have been used as queries. The query is structured as in figure 11.

Operation name: GetWeatherReport

- **Input data type:**

```
<element name="GetWeatherReport">
    <complexType>
      <sequence>
        <element name="ZipCode" type="string" />
      </sequence>
    </complexType>
  </element>
```

- **Output data type:**

```
<element name="GetWeatherReportResponse">
  <complexType>
    <sequence>
      <element name="GetWeatherReportResult" type="string" />
    </sequence>
  </complexType>
</element>
```

Figure 17: Structure of Operation Search Query

Table 2 presents the operations similarity results:

Operations used as Query	Number of Services Retrieved	Number of Services above the Threshold	Precision	Recall	Response Time(ms)
USWeather: GetWeatherReport	17	16	100	94.1	374
WeatherForecast: GetWeatherByPlaceName	37	24	62.5	88.23	392
WeatherByZip: GetWeatherByZip	69	32	50	94.1	422

Table 2: Operation Search Result

The following measurements are calculated based of the average performance of the operation presented in the above table and only results exceeding the specified threshold 5% are considered in the measurements. The precision is equal to the number of relevant Web services above the threshold divided by the total number of Web services above threshold. The recall is equal to the number of relevant Web services above the threshold divided by the total number of relevant Web services in the repository. The total response time to the query was 396 millisecond. The overall precision is 70.83% and over all recall of 92.15%. The top-k precision has achieved 100% precision at Top-1 % recall, 100% precision at Top-5 % recall, 100% precision at Top-10 % recall, 94.4% precision at Top-25% recall, 86% precision at Top-50% recall and 78.7% precision at Top-75% recall. The following graph illustrates the distribution of precision at different recall points.

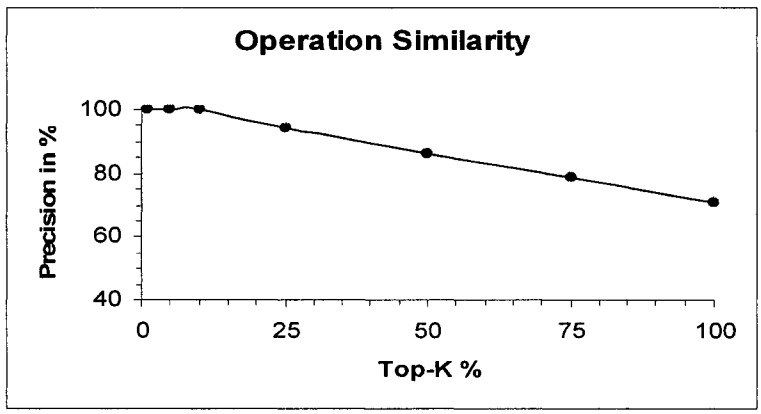

Figure 18: Operation Search Top-k Precision Graph

4.3.2.1 Web Service Search

In the Web service search the query is a Web service. The query is first transformed to a text and then matched to all Web services in the repository. Web services exceeding the threshold are passed to the structure similarity for further refinements. Three Web services each from the weather category have been used as queries.

Web service used as Query	Number of Services Retrieved	Number of Services above the Threshold	precision	recall	Response Time(ms)
USWeather	38	16	100	94.1	437
WeatherForecast	56	21	76.19	94.1	531
WeatherByZip	79	31	51.6	94.1	500

Table 3: Web Services Search Result

The following measurements are calculated based of the average performance of the Web services presented in table 3 and only results exceeding the specified threshold are considered in the measurements. The precision is equal to the number of relevant Web services above the threshold divided by the total number of Web services above threshold. The recall is equal to the number of relevant Web services above the threshold divided by the total number of relevant Web services in the repository. The response time to the query was 489.3 millisecond. The over all precision is 75.9% and over all recall of 94.1%. The top-k precision has achieved 100% precision at Top-1 % recall, 100% precision at Top-5 % recall, 100% precision at Top-10 % recall, 95.8% precision at Top-25% recall, 88.3% precision at Top-50% recall and 81.3% precision at Top-75% recall. The following graph illustrates the distribution of precision at different recall points.

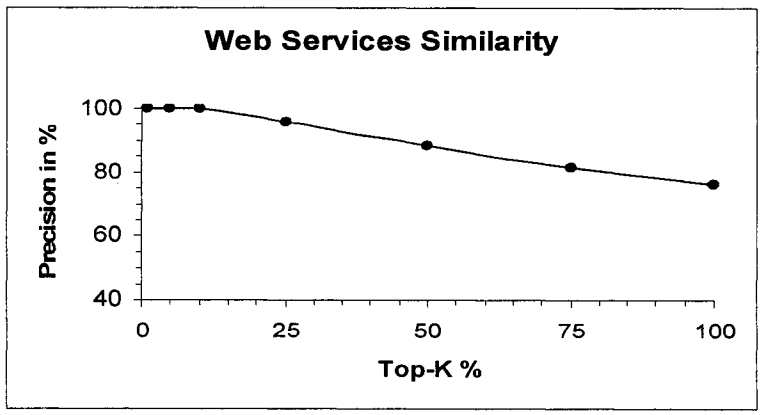

Figure 19: Web Service Search Top-K Precision Graph

4.3.3 Results Analysis

The experiments results have shown that the time required for the pre-processing and the indexing is relatively high. That is because the pre-processing phase runs on every term in the documents. However since the pre-processing is only computed once on the collection of documents, it does not largely effect the query processing. The size of the index is relatively small comparing to the original size of the candidate documents. It is 14.2% of the original size of the repository.

The query processing time for Web services search is higher than the operation search and the keyword search. This is expected as Web service file is larger than the size of the operation and consequently the pre-processing will require more time.

For a keyword search, the precision and recall achieved 100%. This is because a single keyword precisely identifies the query and there are no general terms that can raise the recall. Broad or general terms will achieve a comprehensive retrieval, and consequently reduce the precision. An operation search has shown a lower precision than the keyword search and that is due to the larger number of terms appearing in the operation description. Although the operation filtering has precision less than the keyword search, the Top-k precision analysis has shown that most of the relevant documents appeared on the top of the retrieved list. Web services search have achieved similar precision as the operation search. However, the number of Web services retrieved by the Web service search is larger then the operation search. This is expected as the Web service description has more terms than operation description. As Web service description usually use similar terms in all elements, both operation search and Web services search have retrieved most of the relevant Web services and ranked most of them at the top of the retrieved list. Text comparison has succeeded in filtering over 98% of the irrelevant Web services.

4.4 Conclusion

This chapter described how information retrieval techniques can be used for web service searching. Vector space model relies on terms statistics to measure the similarity between a query and documents. The more precise is the query terms, the higher the precision of the retrieved documents. Results have shown that vector space model can be used for fast retrieval and works well as filtering mechanism for Web services. The vector space model does not rely on term semantic and does not consider the structure of the Web service. It treats Web services as text documents and roughly prunes off Web services that are irrelevant for a given query.

Chapter 5: Structure Similarity

5.1 Introduction

With the growing number of Web services, it is inadequate to measure Web services similarity based only on text-base similarity. The semantic and the structural information are crucial components in identifying similar Web services. There is an increasing need to automatically identify the semantic and structural similarity of Web services for searching, clustering and composition. In order to compute the semantic and the structural similarity between two Web services, it is required to measure the relations among their corresponding elements. Particularly to specify rules for measuring the similarity between two elements and to identify how close two elements should be in order to be considered similar. The goal is to further refine the operations and the services similarity computed by the text comparison by comparing them based on their semantic and structure similarity. The structure similarity will return not only operations and web services that are exactly similar, but also operations and services that are similar in some way and hence would be considered similar if the query is slightly modified.

In this chapter, the structure similarity of two Web services is measured based on the similarity of their operations. The similarity of two operations is based on the similarity of their names, input and output parameters. The semantic similarity is measured using WordNet dictionary [79]. The structure similarity is measured using a tree matching algorithm. The returned list is ranked between 0 and 1 based on there closeness to the query.

5.2 Web Services Similarity

WSDL files expose the services they offer over the Internet using interfaces to operations. Among other things, operations are the most important component of Web service and the focal point of interacting with Web services. In the following, we take an abstract of view of Web service as a collection of operations. i.e., a Web service w is defined as follows:

$w = \{O_1, O_2, ..., O_n\}$

Where w is a Web service, Oi $(i>= 1, i<=n)$ is an operation

Web services similarity is computed based on their operations similarity, which in turn based on their input and output parameters. However, the Binding element of WSDL file is not considered in the similarity measure as it describes how users and applications can communicate with operations.

Measuring the similarity between two Web services based on their operations can be modeled as a bipartite graph-matching problem. This can be preformed by comparing each operation in one Web service to all operations in the other Web service. The result is two sets of operations where each operation in one set has similarity weights with all operations in the other set. The maximum sum of the similarity between the two Web services is computed using the Hungering assignment algorithm described in section 2.1.4.

The total similarity score is computed as follows:

$$sim(w, w') = (max \sum_{i \in w, j \in w'} sim(o_i, o_j)) / |w|$$

Where i and j are the indexes of the operations in the source Web service w and the operations in the target Web service w' respectively. $|w|$ denotes the total number of operations in the source Web service.

5.3 Operations Similarity

An operation is considered as a sequence of three components *(name, input type, and output type).* The structural similarity of two operations is computed based on the mentioned components.

Definition 5.1 (operation similarity)

$$sim(O, O') = sim(O_{name}, O'_{name})*0.5 + sim(O_{type}, O'_{type})*0.5$$

Where O_{name} is the name of the source operation and O'_{name} is the name of the target operation. The final similarity score is normalized to a range between 0 and 1. O_{type} is the source operation parameters and O'_{type} is the target operation parameters. The similarity of operations' parameters is computed as follows:

$$sim(O_{type}, O'_{type}) = sim(T_{ip}, T_{ip'})*0.5 + sim(T_{op}, T_{op'})*0.5$$

Where T_{ip} is the input parameters and T_{op} is the output parameters.

Names similarity is computed as described in section 5.3.1. Input parameters and output parameters similarity is computed as described in section 5.3.2. The final similarity score is a value ranges between 0 and 1.

5.3.1 String Similarity

String similarity method is used to compute the similarity between any two names. A name can be an operation name defined by the attribute name in the operation tag or element name defined by the attribute name in the element tag. The similarity of two strings relies on pre-processing steps such as tokenization and elimination the stop words. If a string is defined as a set of terms $\{t_1, ..., t_n\}$, where a term is a single word. Two strings $S=\{t_1,...,t_n\}$ and $S'=\{t'_1,...,t'_m\}$ are similar if their terms are similar based on definition 5.2.

Each term in the resulting string is used as atomic unit in finding the similarity between two name fields. The relation between terms is measured using the WordNet Dictionary described in section 2.1.5.

In this study, only the synonyms and the hypernym are considered. Considering all relations is computationally expensive and will not contribute much to the similarity measures. Two terms are semantically similar if their WordNet synsets are connected. The strength of a relationship is calculated as follows:

$$Sem_{sim}(t,t') = \frac{\left(\frac{S_c - S_n + 1}{S_c}\right)}{depth} \qquad 5.1$$

Where $Sem_{sim}(t,t')$ is the semantic similarity between the source term t and the target term t'. S_c is the sense size and it represents the number of senses of the source term. S_n is the sense ranking number of the target term in the source senses and it represents how early in the returned list is the target term appears. The *depth* represents the path between the source term and the target term. The depth of a synonyms relation is always equal to one since they are directly connected.

This formula gives more importance to the most frequent sense as they appear at the beginning of the sense list [79]. This formula has been applied to both synonyms and hypernyms relations. As described in section 2.1.5 the JWNL APIs [69] is used to access the WordNet dictionary to obtain the values of the parameters mentioned above.

Example:

Measuring the relations between **car** and **automobile** is as follows:

The **noun** "car" has 5 senses in WordNet.

1. **car**, auto, **automobile**, machine, motorcar -- (wheeled motor vehicle; usually propelled by an internal combustion engine; "he needs a car to get to work")
2. **car**, railcar, railway car, railroad car -- (a wheeled vehicle adapted to the rails of railroad; "three cars had jumped the rails")
3. cable car, **car** -- (a conveyance for passengers or freight on a cable railway; "they took a cable car to the top of the mountain")
4. **car**, gondola -- (car suspended from an airship and carrying personnel and cargo and power plant)
5. **car**, elevator car -- (where passengers ride up and down; "the car was on the top floor")

From the above, the term automobile appeared as a first sense in the term car senses and the total car senses are five. The similarity of the car to the automobile is as follows:

$$Sem_{sim}(car, automobile) = \frac{\frac{5-1+1}{5}}{1} = 1$$

The total result is evaluated to 1 as the automobile is one of the most frequent used synonyms for car. For example measuring the similarity between car and cable car will result in similarity score equal to 0.6 since the cable car is the third in the list of most frequent used synonyms

The above procedure is applied to every two terms in the compared strings.

The result is two lists of terms where each term in one list has a similarity weight with every term in the other list. The final score is determined by applying the Hungarian algorithm on the two lists.

After computing the semantic similarity of operations' names, the semantic and the structure similarity of their input and output parameters is computed. The goal is to

determine which parameter in the source operation corresponds to which parameter in the target operation that maximizes the total sum of parameters' similarity weight.

5.3.2 Parameters Similarity

In WSDL, parameters are defined using *message* elements. Each massage can be either input message or output message. Each message defines one or more *part* element. Each *part* element defines an *operation's* parameter. The messages are described separately to allow messages reuse. Since each massage can be either input or output, more than one operation may use the same input/output parameters. Caching these parameters will speed up the computation time. Given that, measuring parameters similarity consumes most of operations similarity time, a cache hit will significantly reduce the time required for computing the similarity. When parameters are compared, we first consult the cache. If the result is in the cache, it is returned without any further computations. If it isn't in the cache, the parameters similarity is computed and stored in the cache.

The parameter (part element) consists of two attributes; *name* attribute and *typing* attribute. The name attribute defines the name of the parameter and the *typing* attribute defines the type of the parameter. The *typing* attribute can be either an *element* with a value referencing an element in the types' element of WSDL or *type* with a value as built-in data type.

Consider the following messages:

- **message 1:**

```
<message name="ChangeForceUnitHttpGetIn">
    <part name="ForceValue" type="string" />
</message>
```

- **message 2:**

```
<message name="ChangeForceUnitSoapOut">
    <part name="parameters" element="ChangeForceUnitResponse" />
</message>
```

- **type 1:**

```
<element name="ChangeForceUnitResponse">
 <complexType>
   <sequence>
     <element name="ChangeForceUnitResult" type="double" />
   </sequence>
 </complexType>
</element>
```

Figure 20: WSDL Messages Structure

In message 1 the *name* of the parameter is "*ForceValue*" and the *typing* attribute is a type indicating that the type is *built-in* type with a value of *"string"*. However in massage 2 the *name* parameter is *"parameter"* and the *typing* attribute is an element pointing to the type 1 which must be defined in the types' element of WSDL file. In this case, the *name* of the parameter is considered as "*ChangeForceUnitResponse*" and the *type* as a *complex type*.

As described in section 2.1.3, WSDL defines types using *types'* element, which contains the schema element. The *schema* element organizes data types as sets of "element", "simpleType", or "complexType". An "element", a "simpleType" or a "complexType" that is direct child of the schema (global element) represents a particular data type of an input/output parameter. Therefore, element, simpleType, and complexType that are direct children of the schema are technically data types. A data type can also reference any other data type and can be referred by other elements more than once.

Let O and O' be two structure operations

$O = (Ip_1, Ip_2, \ldots Ip_n;\ Op_1, Op_2, \ldots Op_m)$ *for source operation* O

$O' = (Ip'_1, Ip'_2, \ldots Ip'_n;\ Op'_1, Op'_2, \ldots Op'_m)$ for target operation O'

Where *Ip* denotes the input parameters and *Op* denotes the output parameters.

The similarity between two parameters is computed as follow:

$$sim(O,O') = ((\max \sum_{i \in O, j \in O'} sim(Ip_i, Ip'_j)) / \max(|Ip|, |Ip'|)) * 0.5 + ((\max \sum_{i \in O, j \in O'} sim(Op_i, Op'_j)) / \max(|Op|, |Op'|)) * 0.5$$

The similarity of operations' parameters is measured based on the similarity of their input parameters and the similarity of their output parameters. The final score is normalized to a value range between 0 and 1.

The similarity of any two parameters is based on the similarity of their names (identifiers) and the similarity of their type. The similarity of their names is computed based on the string similarity described in section 5.3.1. The similarity of their data type is based on the structure of the XML schema describing their types and is computed based on the similarity of two nodes in XML schema. There are two steps in computing nodes similarity; first is modeling the two schemas as trees and second is measuring the similarity between nodes in the trees.

5.3.2.1 Schema Modeling

If the *schema* element is modeled as a root of tree, all data types and referenced data types will be represented as direct children (known as global elements) of the root. The similarity between two data types becomes the similarity between two global elements.

A *labelled tree* is used to represent the structure of the schema. The schema <*schema*> element is parsed and its elements are translated into *nodes* with the name of element as the label of the node. Each element in the schema (*"element"*, *"complexType"*, *"simpleType"*, *"group"*, *"sequence"*, *"all"* and *"choice"),* is represented by a node in the tree. For example, the root element has its label as 'schema". It is important to notice that the order indictors are also represented as nodes.

The *Order indictors* are used to describe the order in which their children elements should occur. The *all* indictor indicates that its direct children elements can appear in any order, but must appear once and only once. The *choice* indicator indicates that only one of its direct children elements can appear. The *sequence* indicator indicates that all of its children elements must appear in the specified order.

Based on their type, elements may be either non-terminal (non-leaf nodes) or terminal (leaf nodes). For example an element that is a built-in type (i.e. float) will be modeled as a leaf node and an element that is a complexType type will be modeled as non-leaf node.

The tree structure reflects the nesting relations of the schema elements, which in return reflects the structure of data types. As data types in WSDL are direct children of *schema* element, the root of the tree is always 'schema'. The label of a node determines the importance of its children order. For example the order of direct children of the schema element is irrelevant as each element is an atomic unit that describes the structure of a particular data type. On the other hand, the order of children of a sequence node must be considered in the similarity measure.

During the modeling, both reference and group definitions are considered. The reference definition is a mechanism to simplify XML schema structure through enabling the reuse by sharing common segments. There are two methods of reference in XML schema specification; data type referencing and name referencing. Data type referencing is created by the clause "*type=dataTypeName*" where "*dataTypeName"* is a *complexType* or a *simpleType*. The name referencing is created by the clause "*ref=elementName*"

where *"elementName"* is a name of another element. All referred types must be global elements. The group definition provides a way of component reuse. It groups a set of related elements using the tag <*group name= "groupName"* >. The group element can be referenced by any other element using *"groupName"*.

5.3.2.2 Nodes Similarity

XML Schema similarity has attracted a lot of attention due to the extensive adoption of XML-based representation of data. As described in section 2.2.3, several algorithms have been developed to measure the similarity between XML schemas. Some of these algorithms measure the similarity of XML files based on common DTD [41]. Others are developed for general-purpose schema matching including the relational schema and XML Schema [7,35]. Since these algorithms did not specifically developed for XML schema, they do not consider all of XML schema properties and most of them require human interaction ore globally defined schema.

In this work, we propose XML schema similarity algorithm. The aim of this algorithm is to match nodes (sub-trees) instead of the matching the over all schemas. As we are targeting a small XML trees, any changes to the structure of a node will largely affect the accuracy of the result. Thus the proposed algorithm gives special importance to the node structure by considering all the properties of XML schema structure. Each node has structure that defines the properties of an element including name, category, type, max occurrence, and min occurrence.

The similarity between the names of any two nodes (elements) is computed based on the string similarity described in section 5.3.1. As described in section 2.1.1, XML schema allows the specification of minimum and maximum occurrences with range from 0 to unbounded. It is unnecessary and cumbersome to compare all the cardinalities in this range. Thus, the total similarity of nodes is reduced by factor of 10% if their occurrence attributes do not match. For example if the total similarity of nodes names and data type is 1, the total score will be reduced to 0.9 if there occurrence attributes do not match.

The type similarity of two elements is measured based on their category. In XML schema elements are organized into three categories; *built-in type, simple type, and complex type.* The similarity between any two types τ and τ' is computed based on the following rules:

- If τ and τ' are built-in type, their similarity is measured based on their types similarity obtained from cardinality table.
- If τ and τ' are simple types, their similarity is measured based on base their type and their facets. *Constrain facets are considered only if the two elements have the same base.*
- If τ and τ' are complex type, their similarity is measured based on their list types $\{\tau_1, ..., \tau_n\}$ and $\{\tau_1', ..., \tau_n'\}$
- If τ is a built-in type and τ' is a simple type, only the base type of the simple type is considered. However penalty is applied.
- If τ is a built-in type and τ' is a complex type, the build-in type element is compared with all complex type list of types $\{\tau_1', ..., \tau_n'\}$.
- If τ is a simple type and τ' is a complex type, the simple type is compared with complex type list of types $\{\tau_1', ..., \tau_n'\}$

- **Similarity of Built-in Data Types**

There are forty-four built-in types (e.g. int, float,...), including nineteen primitive and twenty-five derived. For example, a built-in type parameter can be defined as follows:

<element name= "temperature" type= "float"/>

The above element defines a parameter of type float with a name as temperature. Instead of measuring the similarity between each two built-in types, a compatibility table obtained from [61] is used. The use of the compatibility table is to reduce the matching time. It divides the built-in types into a set of classes based on their relationships as described by XML schema specifications. It assigns a relationship weight between any two classes. The built-in types are organized into seven classes; *binary, Boolean, dateTime, float, idRef, integer, and string.* The complete list of each class is presented in

appendix C. The similarity weight between any two classes is determined based on table 4.

	binary	boolean	dataTime	float	idRef	integer	string
binary	1						
boolean	0.2	1					
dataTime	0.3	0	1				
float	0.8	0.1	0.2	1			
idRef	0	0	0	0	1		
integer	0.6	0.6	0.6	0.9	0	1	
string	0.6	0.6	0.8	0.6	0.9	0.7	1

Table 4: Cardinality Table for XML Built-in Types

The similarity between two built-in data types is computed in two steps: the first step is to transform any two built-in types τ *and* τ' to a class type.

Definition 5. 3 (transformation similarity)

$sim_{Trans}(\tau, \tau') = \exists$ *a transformation function* δ *such that sim* $(\delta(\tau), \delta(\tau'))$ *is maximized*

Where $sim_{Trans}(\tau, \tau')$ *is the similarity of types* τ *and* τ'

Example:

Consider comparing *nonNegativeInteger* with *negativeInteger* and *nonNegativeInteger* with *double.* First the types are mapped to a class type using transformation function δ. If the two types belong to the same class, their similarity score is evaluated to 1, as shown in (a). If they belong to two different classes, their similarity score is computed based on table 4, as shown in (b).

(a) Similarity of two types belong to the same class

δ (*nonNegativeInteger*) $\rightarrow$ *integer*

δ (negativeInteger) $\rightarrow$ *integer*

$Sim(integer, integer)=1$

(b) Similarity of two types belong to the same class

δ (*nonNegativeInteger*) $\rightarrow$ *integer.*
δ (*double*) $\rightarrow$ *float.*

$sim(integer, float)=0.9$

- **Similarity of Simple Data Types**

A simple type is defined in term of its base type *b* and it's constrain facet list *l(c)*, where $b \in$ *Built-in* data type. There are over twelve constraining facets that can be applied to a simple type. As described in section 2.1.2, these facets depend on the base type. For example, the constraining facets for string base are {*length, minLength, maxLength, pattern, enumeration, whiteSpace*}.

Example:

```
<element name="car" type= "carType"/>
  <simpleType name="carType">
        <restriction base="string">
              <enumeration value="Ford" />
              <enumeration value="BMW" />
        </restriction>
  </simpleType>
```

Figure 21: XML Schema Constraining Facets Example

The above example defines a simple type element. Its name is a car and its base is string. This definition indicates that only the "Ford" and "BMW" are accepted as input values.

The similarity of two simple types is determined by the similarity of their base and their facets.

Definition 5.4 (simple type similarity) given two simple types T= (b, l(c)) and T'= (b', l'(c)),

$sim_{simpleType}(T, T') = sim(b, b') + sim(l(c), l'(c))$

The total similarity of two simple types is based on measuring the similarity of their bases and the similarity of their constraining facets. As the *base* ∈ *Built-in type,* measuring the similarity of their bases is considered as measuring the similarity of two built-in types. However the constraining facets similarity is determined based on the similarity of their constrains lists l(c) and $l'(c)$. Constrains lists are compared only and only if they belong to the same base.

- **Similarity of Complex Data Types**

Complex type element is a tuple (τ, R), *where* $\tau = \{\tau_1, ..., \tau_n\}$, $n \geq 1$, $\tau_i \in$ *{built-in type, simple type, complex type}* and R is an order relationship $R \in$ *{sequence, all, choice},* that define the order of the list elements $\{\tau_1, ..., \tau_n\}$. Each element $\tau_i \in$ *{Built-in type, Simple type, Complex type}.*

It is important to notice that a complex type is recursively defined when τ_i is a *Complex type*. This recursive process will terminate at a point when all elements in τ have types $\tau_i \in$ *{Built-in type , Simple type}.* It is also important to notice that the length of the list type τ is not determined. The similarity between two complex type elements τ and τ' do not require that the len(τ) is equal to len(τ'). The following example shows a recursive structure of a complex type:

```
<element name="car" >
  <complexType >
    <sequence>
      <element name="type" type="carInfo" />
      <element name="color" type="string" />
    </sequence>
  </complexType>
</element>

<complexType name="carInfo">
  <all>
      <element name="carType"   type="string" />
      <element name="year"      type="date" />
  </all>
</complexType>
```

Figure 22: Complex Type Element Structure Example

Note that the car element is a complex type that has one of its element recursively defined by referencing another complex type "carInfo". Note that car element has a sequence relation for its children elements indicating that "carType" and color should be in the specified sequence. The "carInfo" is defined as a complex type with two children. However, its children defined using an all indicter indicating that the order of the children is not important.

The similarity of two complex type elements is based on the relationship of their children elements. The comparison process will use the following relationship rules to compute the similarity of any two complex type elements:

Source	Target	Applied Relationship
Sequence	Sequence	Sequence
Sequence	Choice	Choice
All	All	All
All	Sequence	Sequence
All	Choice	Choice
Choice	Choice	Choice
Choice	all	All
Choice	Sequence	Sequence

Table 5: Complex Type Relationship Indicators

- **Similarity of All Elements**

The *all* relationship indicates that its children elements can appear in any order. The similarity between two complex types τ and τ' with *all* relationship is computed based on definition 5.5.

Definition 5.5 (all similarity)) given two complex types T= (l(c)) and T'= (l'(c)) with all relation

$$sim_{reorder}(T,T') = (\max \sum_{i\in\tau, j\in\tau'} sim(\tau_i, \tau_j)) / \ max(len(T), len(T'))$$

Each element τ_i in one list is compared to every element τ_j' in the other list. The maximum sum is calculated using the assignment Hungarian algorithm, then, the sum is divided by the larger length of the two lists.

Consider the following example:

```
<complexType>                          <complexType>
  <all>                                  <all>
    <element name= "email" type= "string"/>    <element name= "email" type= "string"/>
    <element name= "phone" type= "string"/>    <element name= "phone" type= "string"/>
    <element name= "fax" type= "string"/>      <element name= "fax" type= "string"/>
  </all>                                 </all>
</complexType>                         </complexType>
```

Figure 23: XML Structure of All Indicator

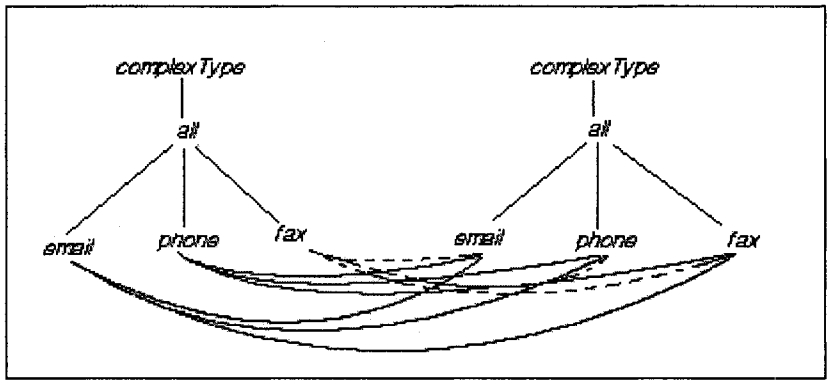

Figure 24: Comparison of All Indicator

Each element in one complex type is compared to all elements of the other complex type. The total similarity is the maximum sum of the similarity scores of all elements such that each element in the source is matched with only one element in the target.

- **Similarity of Sequence Elements:**

The *sequence* relationship restricts the order of its children to be in the specified sequence. The similarity of sequence children is computed as follow:

Definition 5.6 (sequence similarity) given two complex types $T=l(\tau)$ *and* $T'=l'(\tau')$ *with sequence relation*

$$Sim_{sequence}(T,T') = \sum_{i \in \tau} sim(\tau_i, \tau'_i) / \max(len(T), len(T'))$$

A mapping function will map each element τ_i in the source τ to the corresponding element τ'_i in target τ'.

Consider the following example:

```
<complexType>                          <complexType>
  <sequence>                             <sequence>
    <element name= "email" type= "string"/>    <element name= "email" type= "string"/>
    <element name= "phone" type= "string"/>    <element name= "phone" type= "string"/>
    <element name= "Fax" type= "string"/>      <element name= "Fax" type= "string"/>
  </sequence>                            </sequence>
 </complexType>                         </complexType>
```

Figure 25: XML Structure of Sequence Indicator

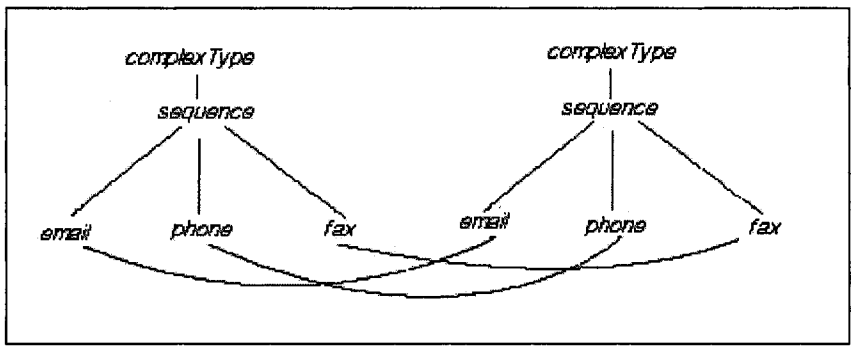

Figure 26: Comparison of Sequence Indicator

The similarity is computed as follow:

Total similarity=(sim(email,email)+sim(phone,phone)+sim(fax,fax))/3.

- **Similarity of Choice Elements:**

  The *choice* relationship indicates that only one element of its children can appear. In the *choice* relationship, elements are compared using all relationship rules. Only the pair that scores the maximum value is considered as the final score.

Definition 5.7 (choice similarity) given two complex types T= (l(τ)) and T'= (l'(τ')) with choice relation

$$sim(T,T') = max(sim(\tau_i, \tau'_j))$$

Consider the following example:

```
<complexType>                          <complexType>
  <choice>                               <choice>
    <element name= "email" type= "string"/>    <element name= "phone" type= "string"/>
  <element name= "address" type= "string"/>    <element name= "email" type= "string"/>
  </choice>                                    <element name= "fax" type= "string"/>
 </complexType>                              </choice>
                                            </complexType>
```

Figure 27 : XML Structure Choice Indicator

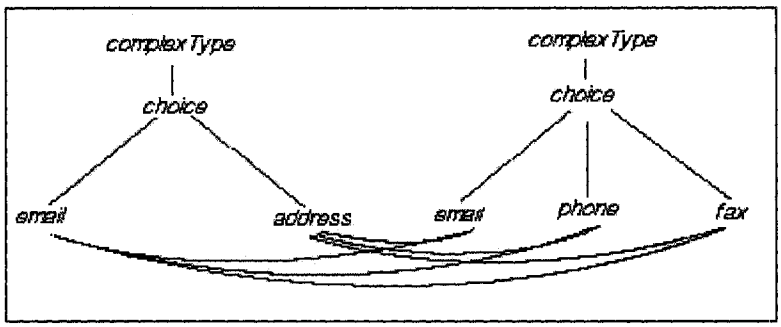

Figure 28: Comparison of Choice Indicator

The highest similarity score of any pair is considered as the final similarity score. The highest similarity pair is determined using the Hungarian stable matching algorithm.

- **Similarity Between Built-in Data Type and Simple Type:**

Simple type element consists of a base and facets. However, when compared with a built-in type, the facets constrains are ignored. The comparison process considers only the base. Both types are compared as built-in data types. As the two data types belong to different categories a penalty factor α is applied.

- **Similarity between non-complex Type and Complex Type:**

Comparing two complex types is determined based on their children $\{\tau_i, ..., \tau_n\}$ and their relationship R. However, the similarity between non-complex type element τ' and a complex type τ is computed by comparing non-complex type τ' to the complex type children $\{\tau_1, ..., \tau_n\}$. For example if τ' and τ_i are both build-in type, then the similarity between two built-in type elements is applied. If τ' is a built-in type and τ_i is a complex type, then we recursively compare τ' to all elements of τ_i .the recursive process will terminate when all elements are built-in types and simple types. The total score is calculated as follows:

Definition 5.8(non-complex type to complex type similarity)

$$sim(\tau, \tau') = \sum(sim(\tau, \tau_i'))/len(\tau')$$

Consider the following two elements:

```
<element name="name" type= "string">

<element name=personalInfo>
        <complexType>
            <all>
                <element name= "name" type= "string"/>
                 <element name= "contactInfo" />
                    <complexType>
                        <all>
                            <element name= "email" type= "string"/>
                            <element name= "phone" type= "string"/>
                        </all>
                    </complexType>
                </element >
            </all>
        </complexType>
    </element >
```

Figure 29: XML Structure of Built-in Type and Complex Type

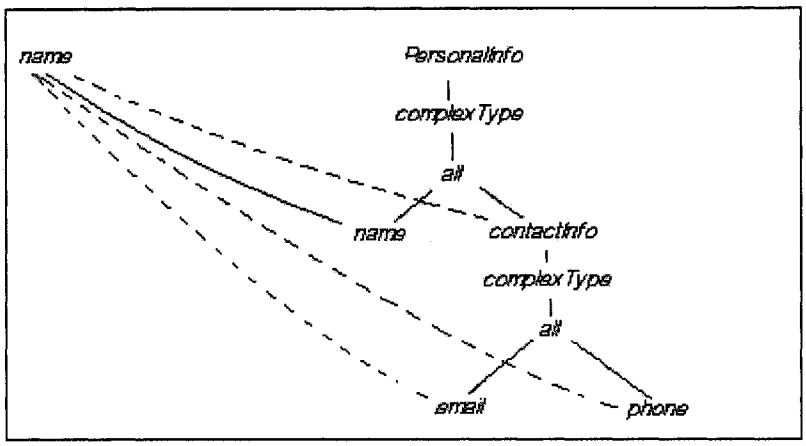

Figure 30: Comparing Primitive Type Element to Complex Type Element

$$sim(name, personalInfo) = (sim(name, name) + sim(name, contactInfo))/len(personalInfo)$$

$$sim(name, contactInfo) = (sim(name, email) + sim(name, phone))/len(contactInfo)$$

It is important to notice that as the depth of the tree of the complex type grows or the number of its children increases, the total score decreases.

5.4 Time Complexity Analysis

There are three major steps in computing the structural similarity of Web services. The first is the operation similarity. The second is the parameters similarity. The third is the data types similarity.

5.4.1 Operations Similarity

Given two web services w and w', each containing a collection of operations:

$$w = \{Oi, ..., On\}$$
$$w' = \{O'i, ..., O'm\}$$

Where n and m are the number of operations in the source and the target Web services respectively. Each operation in the source Web service is compared with all operations in

the target Web service resulting in time complexity equal to $O(m \times n)$. However computing the total sum using the Hungarian Method for Weighted Bipartite Graph requires polynomial time complexity equal to $O(n+m)^3$ [31].

5.4.2 Parameters Similarity

Given two operations O and O', each containing collections of input parameters and output parameters:

$O = (Ip_1, Ip_2, \ldots Ip_n ; Op_1, Op_2, \ldots Op_m)$ *for source operation* O

$O' = (Ip'_1, Ip'_2, \ldots Ip'_{n'} ; Op'_1, Op'_2, \ldots Op_{m'})$ for target operation O'

Where Ip denotes the input parameters and Op denotes the output parameters. The complexity of comparing the input parameters of the source operation to the input parameters of the target operations is equal to $O(n \times n')$, where n is the number of input parameters of the source operation and n' is the number of input parameters of the target operations. The total sum of input parameters similarity is computed using the Hungarian Method for Weighted Bipartite Graph resulting in time complexity equal to $O(n+n')^3$. Similarly, the time complexity of the output parameters similarity is equal to $O(m+m')^3$.

5.4.3 Data Types Similarity

Given two XML types u and v, their structural similarity is determined based on their category. Consequently, their time complexity is determined based on their structure.

Case 1: Both u and v are either built-in types or simple types (do not have children). Clearly, the time complexity in this case is a constant.

Case 2: Both u and v are complex types. In this case, the complexity depends on their children ordering relationship (all, sequence or choice) and the number of children elements of u and v. Assume the number of children of u is equal to n and the number of

children of v is equal to m, then if their ordering relation is choice or all, the complexity is equal to $O(m \times n)$. If their relation is a sequence, then the complexity is equal to $O(n)$. However computing the total sum using the Hungarian Method for Weighted Bipartite Graph will require $O(n+m)^3$ time complexity.

Case 3: Type u is non-complex type and type v is a complex type. In this case u is compared with all children of v resulting in complexity equal to $O(n)$

From the above analysis, the worst case for our algorithm is a polynomial time complexity equal to $O(n)^3$.

5.5 Experiments

This section presents the experimental design and the results analysis for evaluating the performance of the similarity measure algorithm. In particular we are measuring the recall, precision, Top-K precision and the response time to the query. We ran three kinds of experiments to evaluate the system. The first kind of experiment uses operation description as a query. The second kind of experiment uses partial operation description as query and the third kind of experiment uses Web service description as query. All experiments are preformed on the same set of queries, machine and data collection used in chapter 4.

5.5.1 Performance

5.5.1.1 Operation Similarity

Operation similarity measures the relevance of operations in the collections of Web services to query operation. Web services are broken down into operations and the query operation is compared to each operation in the collection. It is possible that not every operation in a Web service relevant to the query and operations in the same Web service may have different ranking scores.

Measuring the performance of operations similarity requires identifying all operations that are relevant to the query operation. We have specified a total of 36 operations 18 of them are relevant to the queries operations. All operations similarity measurements are preformed on this set of operations. The total set of operation and their Web services are presented in appendix B.

The same set of operations descriptions presented in the text comparison is used as queries to measure the structure similarity. The result is a list of operations names proceeded by the name of the Web services that contains the operation. Table 5.3 presents the experiments result:

Operations used as Queries	Number of operation	above Threshold	Precision	Recall	Response Time(ms)	Response time(ms) Caching
USWeather: GetWeatherReport	36	19	94.7	100	5344	5015
WeatherForecast: GetWeatherByPlaceName	36	20	95	100	3750	3750
WeatherByZip: GetWeatherByZip	36	20	90	100	2422	2403

Table 6: Operation Similarity Results

The average performance of the operations exceeding a threshold of 25 % is as follows: The response time to the query without using the cache was 3838 millisecond. The response time to the query using the cache was 3722 millisecond. The precision is 93.23 % and recall of 100%. The top-k precision has achieved 100% precision at Top-1 % recall, 100% precision at Top-5 % recall, 100% precision at Top-10 % recall, 90% precision at Top-25% recall, 93.3% precision at Top-50% recall and 95.5% precision at Top-75% recall. The following graph illustrates the distribution of precision at different recall points.

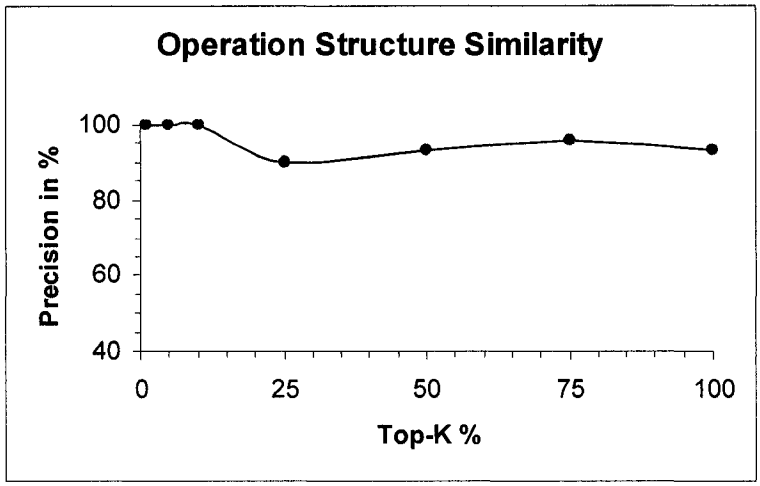

Figure 31: Operation Structure Similarity Precision and Recall Graph

5.5.1.2 Operation Partial Similarity

In previous discussion of operation similarity users have to specify operation name, input parameters and output parameters to be able to search for similar operations. A more practical approach is to allow the user to specify only a sub set of the required information. For example, users may have difficulty determining the type of the input because they are more interested in the type of the output. Allowing users to define an input type that matches any type in the advertised operations is more useful in this case.

In this section we will investigate the partial operation structure similarity using either operation name or parameters structure as query. This kind of search does not exploit full aspects of the structure similarity; however it gives a general assessment about operations related to the query. The user can provide more information for more precise similarity.

Table 7 described the results of structure similarity using only the operations name as query.

Queries as operation names	Number of operation	above Threshold	Precision	Recall	Response Time(ms)
USWeather: GetWeatherReport	36	18	100	100	1734
WeatherForecast: GetWeatherByPlaceName	36	19	94.7	100	1954
WeatherByZip: GetWeatherByZip	36	20	90	100	1969

Table 7: Operation Name Similarity Results

The average performance of the operations exceeding a threshold of 25 % is as follows: The response time to the query was 1885 millisecond. The over all precision is 94.9% with over all recall of 100%. The top-k precision has achieved 100% precision at Top-1 % recall, 100% precision at Top-5 % recall, 100% precision at Top-10 % recall, 93.3% precision at Top-25% recall, 92.6% precision at Top-50% recall and 95.3% precision at Top-75% recall. The following graph illustrates the distribution of precision at different recall points.

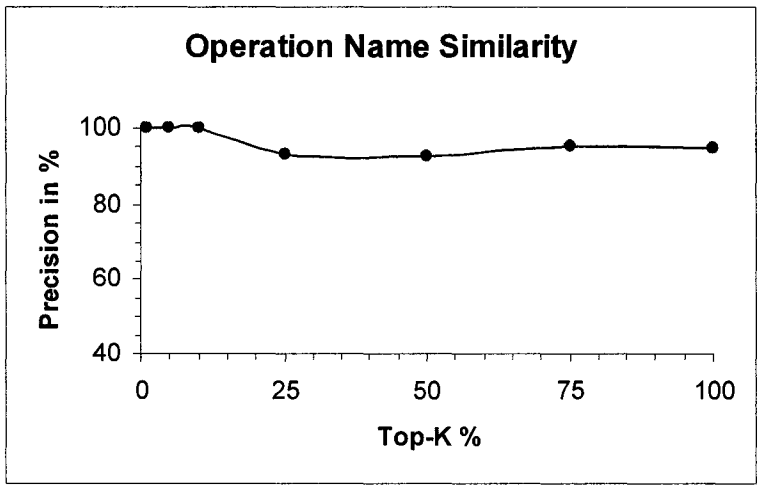

Figure 32: Operations Name Structure Similarity Precision and Recall Graph

Table 8 described the results of structure similarity using only the operations parameters as query.

Queries as operation parameters	Number of operation	above Threshold	Precision	Recall	Response Time(ms)
USWeather: GetWeatherReport	36	17	53	50	4703
WeatherForecast: GetWeatherByPlaceName	36	17	47	44	3063
WeatherByZip: GetWeatherByZip	36	13	53	72	1625

Table 8: Operation Parameters Similarity Results

The average performance of the operations parameters exceeding a threshold of 25 % is as follows: The response time to the query was 3130 millisecond. The over all precision is 50 % and over all recall of 55%. The top-k precision has achieved 100% precision at Top-1 % recall, 100% precision at Top-5 % recall, 66% precision at Top-10 % recall, 65% precision at Top-25% recall, 64% precision at Top-50% recall and 58% precision at Top-75% recall. Figure 5.16 illustrates the distribution of precision at different recall points.

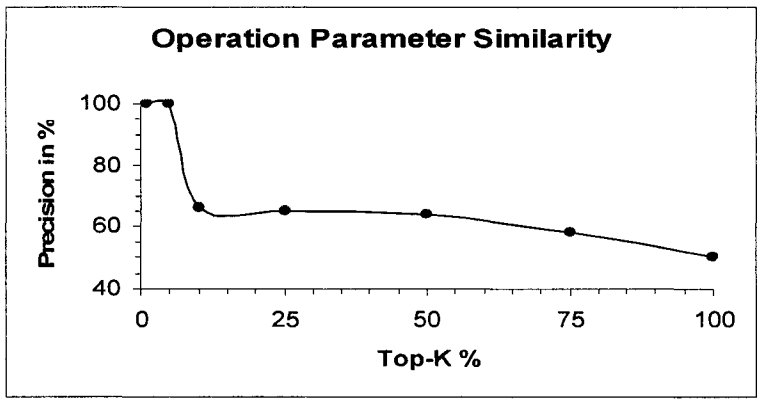

Figure 33: Operations Parameters Structure Similarity Precision and Recall Graph

From figure 26 and figure 27, both name similarity and parameter similarity have 100% precision at very low recall point. However at higher recall points name similarity has much higher precision

5.5.1.3 Web Services Similarity

In Web services similarity, the structure of all Web services pass the text comparison filter is compared with the query. In addition to the precision, recall and top-k precision we investigated the relationship between the response time, file size and the number of operations in the Web service. The affect of caching has been also investigated. Only results exceeding a threshold of 25% are considered.

Query	File size bytes	# of oper.	Number Services	above Threshold	Prec.	Recall	Response time(ms)	Response time(ms) Caching
USWeather	4544	3	16	14	100	87.5	16313	10953
WeatherForecast	10482	6	21	17	70	75	78640	40844
WeatherByZip	4954	3	31	25	56	87.5	24422	18500

Table 9: Web Services Structure Similarity Results

The following measurements are based on the average performance. The response time to the query without using the cache was 39791 millisecond. The response time to the query using the cache was 23432 millisecond. The over all precision is 75.3% and over all recall of 83.3%. The top-k precision has achieved 100% precision at Top-1 % recall, 100% precision at Top-5 % recall, 100% precision at Top-10 % recall, 88.6% precision at Top-25% recall, 83.5% precision at Top-50% recall and 75.6% precision at Top-75% recall. The following graph illustrates the distribution of precision at different recall points.

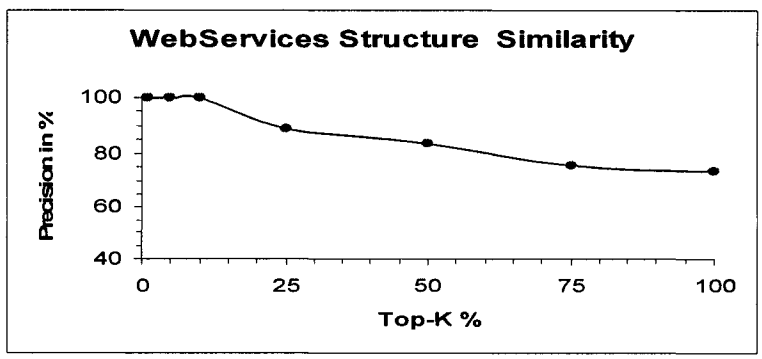

Figure 34: Web Services Structure Similarity Precision and Recall Graph

In addition, the relations between the WSDL sizes, the number of operations and the execution time have been also measured.

Table 10 and figure 29 illustrate the relation between the number of operations and the execution time for USWeather Web service.

Number of Operation in the Target Service	Average Execution time with Caching	Average Execution time with No Caching
1-5	199	280
5-10	589	828
15-20	2281	4211

Table 10: Relation between Number of Operations and Execution Time for USWeather

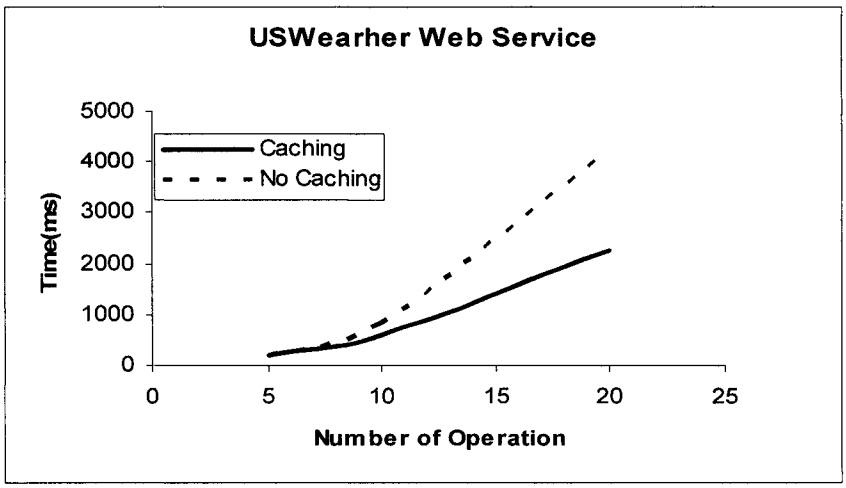

Figure 35: Relation between Number of Operations and Execution Time for USWeather

Table 11 and figure 30 illustrate the relation between the file size in kilo bytes and the execution time for USWeather Web service.

File size	Average execution time with Caching	Average execution time with No Caching
1-5k	31	47
5-10k	310	425
10-15k	500	750
20—25k	1570	2625

Table 11: Relation between File Size and Execution Time for USWeather

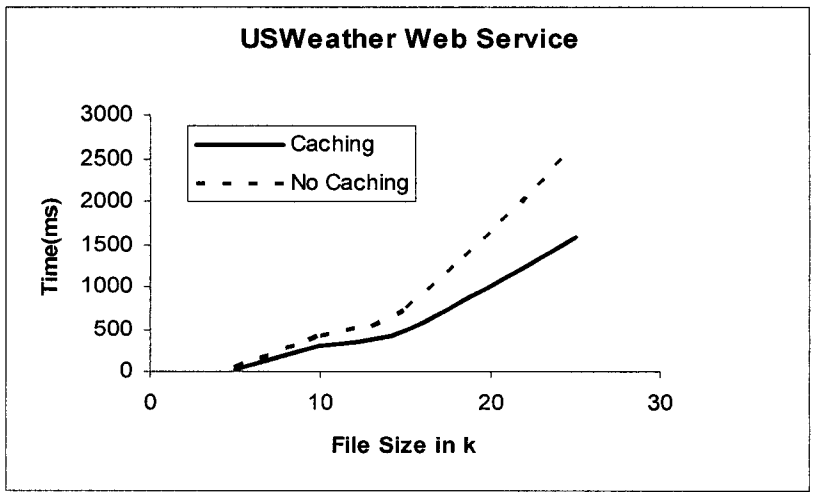

Figure 36: Relation between File Size and Execution Time for USWeather

Table 12 and figure 31 illustrate the relation between the number of operations and the execution time for WeatherForecast Web service.

Number of Operation in the Target Services	Average execution time with Caching	Average execution time with No Caching
1-5	199	280
5-10	589	828
15-20	2281	4211

Table 12: Relation between Number of Operations and Execution Time for WeatherForecast

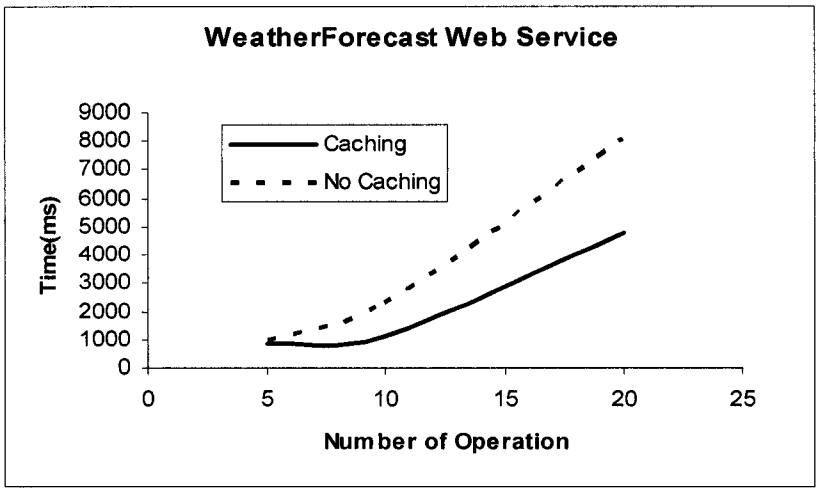

Figure 37: Relation between Number of Operations and Execution Time for WeatherForecast

Table 13 and figure 32 illustrate the relation between the file size in kilo bytes and the execution time for WeatherForecast Web service.

Target Web Service File size	Average execution time with Caching	Average execution time with No Caching
1-5k	31	47
5-10k	310	425
10-15k	500	750
20—25k	1570	2625

Table 13: Relation between File Size and Execution Time for WeatherForecast

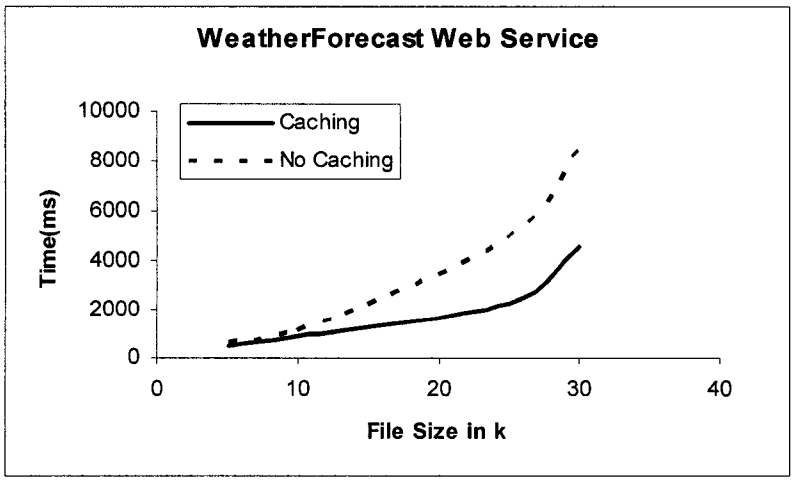

Figure 38: Relation between File Size and Execution Time for WeatherForecast

Table 14 and figure 33 illustrate the relation between the number of operations and the execution time for WeatherByZip Web service.

Number of Operation in Target Web Services	Average execution time with Caching	Average execution time with No Caching
1-5	199	280
5-10	589	828
15-20	2281	4211

Table 14: Relation between Number of Operations and Execution Time for WeatherByZip

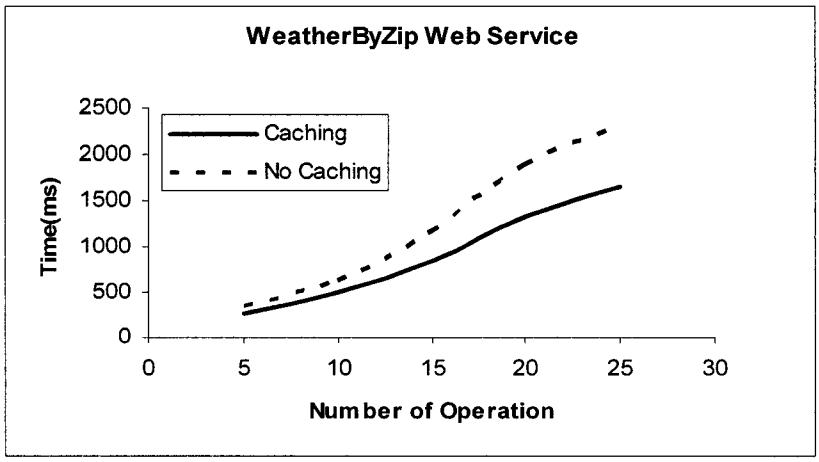

Figure 39: Relation between Number of Operations and Execution Time for WeatherByZip

Table 15 and figure 34 illustrate the relation between the file size in kilo bytes and the execution time for WeatherByZip Web service.

File size	Execution time with Caching	Execution time with No Caching
1-5k	31	47
5-10k	310	425
10-15k	500	750
20—25k	1570	2625

Table 15: Relation between File Size and Execution Time for WeatherByZip

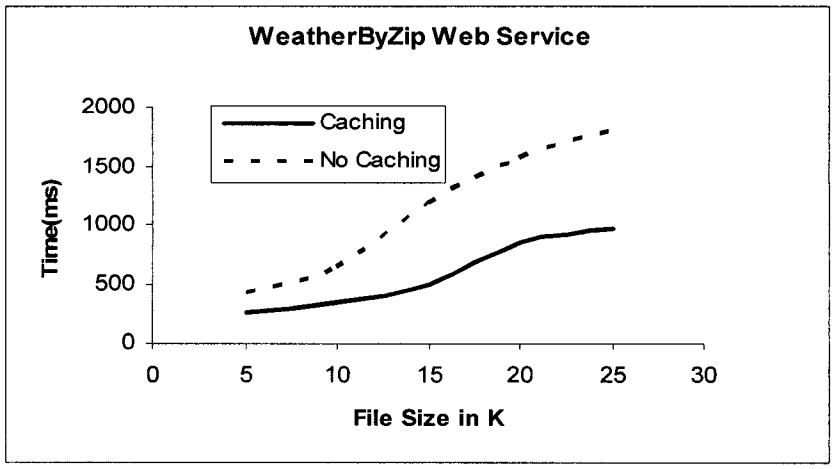

Figure 40: Relation between File Size and Execution Time for WeatherByZip

From the above tables and figures, the execution time increases with the file size or the number of operation of the target Web services. As the size of file increase, the number of operations in the file increases. The higher the number of operations in a Web services, the larger the possibility of parameters reuse, the more effective the cache is.

5.5.2 Results Analysis

The experiments results have shown that the time required for operations similarity is much less than the time required for the Web services similarity. That is expected as a Web service may contain more than one operation. The cache has almost no effect on the operation similarity and that is due to the collection of operation selected for the experiments. As the set of operation has been selected form different Web services and from different categories to measure the effectiveness of the system, it is not expected that these operations would use the same parameters. The query response time for operation search is less than 4 seconds and operation search has archived over 90% precision with 100 % recall and most of the related services have been ranked at the top of the returned list. The partial operation search has shown that operation name similarity achieved higher recall and precision and less execution time than the operation parameters similarity. Web service structure similarity response time is higher than the operation search response time. The average response without cache is less than 40 seconds and with using the cache is less than 25 seconds. Web service search has achieved an average over 75% precision and over 80% recall. The effect of the cache is apparent with Web services containing more than 15 operations or file size over 10 k.

5.6 Conclusion

In this chapter we have described the structure similarity measure for operation search and Web service search. Bipartite graph matching and tree matching algorithm have been used to measure the similarity of operations and Web services. The names similarity has been computed using WordNet dictionary. The input parameters of the source operation are only compared to the input parameters of the target operation and the output parameters of the source operation are only compared to the output of the target operation. A caching mechanism has been used to increase the computational time. XML schema syntax such as element cardinality and order indicators and group style has been considered. Extensive evaluation of the system has shown the system preformed well in term of efficiency and effectiveness.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

There are five contributions of this work:

- A novel approach for Web service searching based on bipartite graph matching
- A new algorithm for XML schema matching based on recursive tree matching
- Increasing the speed of the matching process by utilizing parameters caching
- Introducing a search engine that provides three searching criteria with two filtering modes
- Extensive experiments on matching real life Web services and comparative analysis

The experiments results of text comparison have shown that the time required for the pre-processing and the indexing of Web services collections was relatively high. However since the pre-processing is only computed once on the collection of documents, it does not largely effect the query processing. The size of the index was only 14.2% of the original size of the total collection. Both operation filtering and web services filtering have achieved high precision and recall and were able to rank the relevant results at the top of the retrieved list. Text comparison has succeeded in filtering over 98% of the irrelevant Web service. In the structure similarity, experiments results have shown that operation similarity has achieved over 90% precision with 100 % recall and web service similarity have achieved over 75% precision and over 80% recall. The response time for operation query is much less than the response time for a Web services query. The cache has almost no effect on the operation structure similarity; however, it has enormous effect on Web service similarity especially web services with large number of operations. The partial operation search has shown that operation's name similarity achieved higher recall and precision and less execution time than the operation's parameters similarity.

6.1 Future Work

The system can be extended to include signature matching. The signature matching is crucial for automatic Web services composition, where the output of one operation is automatically passed to another operation. The signature matching should return a Boolean matching score that indicates two operations can be integrated or not. For example the use of WordNet dictionary and type cardinality tables will not be effective in this case. The type matching sub system can also be extended to include subtypes. The subtype measure will be able to identify when a type is included in anther type based on the structure of the two types and consequently determines whether the two types can be substituted. Other Web service discovery benchmark can be implemented and compared with the results obtained from our system to identify the weakness and advantages of the system.

References

1. A. Ankolekar et al: DAML-S: *Semantic Markup for Web Services*, Proceedings of the International Semantic Web Workshop, 2001.

2. A. Ankolekar et al., DAML-S: *Web Service Description for the Semantic Web.* The semantic Web–ISWC 2002: Proc. 1st Int'l Semantic Web Conf. (ISWC), Springer-Verlag, Berlin, 2002, pp. 348–363.

3. N.J. Belkin and W.B. Croft. *Information filtering and information retrieval: two sides of the same coin?* CACM, Dec. 1992. 35(12), pp. 29-37 (8)

4. Boualem Benatallah and Fabio Casati: *An Overview of Standards and Related Technology in Web Services.* Distributed and Parallel Databases, 12, 135–162, 2002

5. D. Bianchini, V. De Antonellis, B. Pernici, P. Plebani: *Ontology based methodology for e-Service discovery*, Information Systems Journal

6. B. Cheng and Y. Chen, "*A Semantic Foundation for Specification Matching*," in *Foundations of Component-Based Systems*, Eds. M.Sitaraman and G. Leavens, Cambridge University Press, 2000. pp. 91-109

7. Hong-Hai Do and E. Rahm. *COMA-A system for flexible combination of schema matching approaches.* Proceedings of the 28th VLDB Conference, Hong Kong, China., 2002

8. Hong-Hai Do, S. Melnik and E. Rahm. *Comparison of schema matching evaluations.* Proceedings of GI-Work "Web and Database", Oct. 2002.

9. A. Doan, P. Domingos, and A. Levy. *Learning Source Descriptions for Data Integration.* In Proceedings of the International Workshop on the Web and Databases (WebDB), Dallas, Texas, 2000. pp. 81-92, 2000

10. A. Doan, P. Domingos, and A. Halevy. *Reconciling schemas of disparate data sources: A machine-learning approach.* In proc. SIGMOD Conference, 2001

11. X. Dong, A. Halevy, J. Madhavan, E. Nemes and J. Zhang: *Similarity Searching for Web Services.* Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

12. D. Fensel, et al: *The Unified Problem-solving Method development Language UPML.* Knowledge and Information Systems Volume 5, Issue 1, Pages: 83 - 131 Springer-Verlag New York, Inc. New York, NY, USA, March 2003

13. D. Fensel and C. Bussler. *The Web Service Modeling Framework WSMF.* http://informatik.uibk.ac.at/users/c70385/wese/wsmf.bis2002.pdf (viewed April 2004)

14. M. J. Fisher, J. E. Fieldsend and R. M. Everson. *Multi-Objective Optimisation for Information Access Tasks*, viewed January 28, 2005 at http://www.dcs.ex.ac.uk/people/mjfisher/MOOIA.pdf

15. Xiang Gao; Jian Yang; Papazoglou, M. P; *The Capability Matching of Web Services.* Multimedia Software Engineering, Proceedings. Fourth International Symposium on , 11-13 Dec. 2002 Pages:56 - 63

16. M. Garofalakis, and A. Kumar. *Correlating XML Data Streams Using Tree-Edit Distance Embeddings*. In Proceedings of ACM PODS'2003. San Diego, California, June 2003, pp. 143-154. ACM Press.

17. David Grossman: *Information Retrieval.* Viewed December 2004 at http://ir.iit.edu/~dagr/cs529/files/ir_book/

18. David Grossman: *Retrieval Strategies and Vector Space Model, Implementation details*. viewed in December 15, 2004 at: http://ir.iit.edu/~dagr/cs529/files/handouts/03VectorSpaceImplementation-6per.PDF

19. Grossman, Frieder and Goharian, *Boolean Vector Space*. Viewed December 15 2004 at: http://www.eng.auburn.edu/~gilbert/Comp7120/Concept-50/IR-BooleanVectorSpace.pdf

20. S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. *Approximate XML Joins*. In ACM SIGMOD, Madison, WI, Jun. 2002.

21. B. Habert et al: *Towards Tokenization Evaluation.* In Proceedings of LREC-98, pages 4:27-431, 1998.

22. J. Hendler and D. McGuinness, *The DARPA Agent Markup Language*. IEEE Intelligent Systems, vol. 15, no. 6,Nov./Dec. 2000, pp. 72–73.

23. Wolfgang Hoschek. *The Web Service Discovery Architecture*. In Proc. of the Int'l. IEEE/ACM Supercomputing Conference (SC 2002), Baltimore, USA, November 2002. IEEE Computer Society Press.

24. Wolfgang Hoschek: *A Unified Peer-to-Peer Database Framework for XQueries over Dynamic Distributed Content and its Application for Scalable Service Discovery.* PhD Thesis, Technical University of Vienna, March 2002.

25. Wolfgang Hoschek: *A Unified Peer-to-Peer Database Protocol.* Technical report, DataGrid-02-TED-0407, April 2002 http://dsd.lbl.gov/~hoschek/publications/DataGrid-02-TED-0407.pdf

26. R. Hull, M. Benedikt, V. Christophides, and J. Su. *E-services: A look behind the curtain.* Proc. of the 22nd ACM Symposium on Principles of Database Systems (PODS) (pp. 1–14). San Diego, CA, USA: ACM Press. 2003

27. J. J. Jeng and B. H. Cheng. *Specification matching for software reuse*: a foundation. Proceedings of ACM software Engineering Note, pages 97-105, 1995

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

29. M. Klein and A.Bernstein. *Searching for Services on the Semantic Web Using Process Ontologies.* Proceedings of the First International Semantic Web Working Symposium. Stanford CA. July 30-August 1-2001

30. András Kornai: *How many words are there?.* Glottometrics 2002/4 61-86. Viewed December 20 2004 at http://www.kornai.com/Papers/hmwat.pdf

31. H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Research Logistic Quarterly, pp. 83-97, 1955

32. D.L. Lee, Chuang Huei and K. Seamons: *Document ranking and the vector-space model Software.* IEEE, Volume: 14, Issue: 2, Mar/Apr 1997 Pages: 67 – 75

33. J. B Lovins: *Development of a stemming algorithm.* Mechanical Translation and Computational Linguistics 1968; 11,22-31

34. L. Luqi. *Normalized specifications for identifying reusable software.* Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow, Dallas, Texas, United States IEEE Computer Society Press, Pages: 46 - 49 , 1987

35. J. Madhavan,, P Bernstein, and E. Rahm: *Generic schema matching with Cupid.* Proceedings of VLDB 49-58, 2001

36. D.L. Mcguinness, R. Fikes, J. Hendler, L.A. Stein: *DAML+OIL: an ontology language for the Semantic Web.* Intelligent Systems, IEEE, Volume: 17, Issue: 5, Sept.-Oct. 2002 Pages: 72 – 80

37. M. Mecella, B B. Pernici, and P. Craca: *Compatibility of e-Service in a Cooperative Multi-Platform Environment.* In Proc. Of the second VLDE International Workshop on Technologies for e-service, Room September 15-2001

38. George A. Miller: *WordNet: a lexical database for English.* Communications of the ACM. Volume 38, Issue 11 (November 1995) Pages: 39 - 41

39. E. Motta, J. Domingue, L. Cabraland, and M. Gaspari. *IRSII: A Framework and Infrastructure for Semantic Web services.* In The SemanticWeb - ISWC 2003, volume 2870 of LNCS, pages 306 – 318. Springer, 2003.

40. Hwee Tou Ng and Hian Beng Lee: *Integrating multiple knowledge sources to disambiguate word sense: an exemplar-based approach.* Proceedings of the 34th conference on Association for Computational Linguistics Santa Cruz, California Pages: 40 – 47 1996

41. A. Nierman and H. V. Jagadish. *Evaluating Structural Similarity in XML Documents.* In Int'l Workshop on the Web and Databases (WebDB), Madison,WI, Jun. 2002.

42. N. Oren. *Reexamining tf.idf based information retrieval with genetic programming.* Proceeding of the 2002 Annual Research Conference of the South Africa.2002, pp. 224 – 234

43. K. Passi, L. Lane, S. Madria, B.C. Sakamuri, M. Mohania and S. Bhowmick: *A Model for XML Schema Integration.* In Proc. of *3rd Intl. Conf. EC-Web*, Springer LNCS 2455, pp. 193-202, Aix-en-Provence, France, Sept. 2002.

44. T. Pedersen, S. Patwardhan, and J. Michelizzi: *WordNet::Similarity - Measuring the Relatedness of Concepts.* the Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), July 25-29, 2004, San Jose, CA

45. M. F. Porter: *An algorithm for suffix stripping.* Morgan Kaufmann Multimedia Information and Systems Series, Readings in information retrieval. Pages: 313 – 316, 1997

46. M. F. Porter: *The Porter Stemming Algorithm.* Viewed Feb 2004 at http://www.tartarus.org/~martin/PorterStemmer/

47. E. Rahm and P. A. Bernstein: *A Survey of Approaches to Automatic Schema Matching.* VLDB Journal 10: 4, 2001

48. G. Salton, A. Wong, and C. S. Yang: *A vector space model for automatic indexing.* Communications of the ACM, Volume 18, Issue 11, November 1975, Pages 613-620

49. G. Salton , E.A. Fox and H. Wu: *Extended Boolean Information Retrieval.* Communications of the A CM, 1983, 26(11), pp. 1022-1036.

50. G. Salton, Automatic Text Processing: *The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, Reading, MA, 1989.

51. Amit Singhal, Chris Buckley, and Mandar Mitra: *Pivoted document length normalization.* Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, August 1996

52. A. Singhal and M. Kaszkiel: *A case Study in Web Search using TREC Algorithm.* In the tenth international conference on World Wide Web, Hong Kong, May 2001

53. Sycara, K.; Widoff, S.; Klusch, M.; Lu, J., *LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace.* Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic, 5, pp. 173-203, 2002.

54. K. Sycara, J. Lu, M. Klusch and S. Widoff. *Matchmaking among heterogeneous gents on the Internet.* Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace, March, 1999

55. R. Steigerwald, L. Luqi, V. Berzins: *A tool for reusable software component retrieval via normalized specifications.* System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on, Volume: ii, 7-10 Jan. 1992 Pages:18 - 26 vol.2

56. Steven L. Tanimoto, Alon Itai and Michael Rodeh: *Some Matching Problems for Bipartite Graphs.* Journal of the ACM (JACM). Volume 25 , Issue 4 Pages: 517 - 525 (October 1978)

57. A. Tsalgatidou and T. Pilioura: *An overview of standards and Related Technology in Web services.* In: International journal of Distributed and Parallel Databases, Special Issue on E-services, 12(2), Kluwer, Sep 2002, 135-162

58. A. M. Zaremski and J. M. Wing: *Signature matching, a tool for reusing software libraries.* ACM Transactions on Software Engineering and Methodology, Vol. 4, No. 2, April 1995, Pages 146-170

59. A. M. Zaremski and J. M. Wing: *Specification Matching of Software Components.* ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, October 1997, Pages 333-369

60. Paola Velardi, Paolo Fabriani, Michele Missikoff: *Using text processing techniques to automatically enrich a domain ontology.* Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001

61. Ju Wang: Matching *XML Schemas by a New Tree Matching Algorithm*: Master thesis. University of Windsor. Computer Science Department. 2004

62. Y. Wang and E. Stroulia.: Semantic *structure matching for accessing web-service similarity.* Proceedings of the First International Conference on Service Oriented

Computing (ICSOC2003), volume 2910 of Lecture Notes in Computer Science, pages 194–207. Springer-Verlag, 2003

63. Y. Wang and E. Stroulia. *Flexible interface matching for web-service discovery.* In Proceedings of 4th International Conference on Web Information Systems Engineering (WISE2003).

64. *Cascading Style Sheets, level 2, CSS2 Specification*, W3C Recommendation 12-May-1998 http://www.w3.org/TR/REC-CSS2/

65. *Stop words list:* Department of Computing Science at the University of Glasgow. Viewed Jan.23 2005 at http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words

66. *Extensible Markup Language (XML) 1.0 (Second Edition)* W3C Recommendation 6 October 2000 http://www.w3.org/TR/2000/REC-xml-20001006

67. *Guide to the W3C XML Specification ("XMLspec") DTD*, Version 2.1 http://www.w3.org/XML/1998/06/xmlspec-report.htm

68. *HTML 4.01 Specification W3C Recommendation* 24 December 1999 http://www.w3.org/TR/REC-html40/

69. *JWNL (Java WordNet Library):* http://sourceforge.net/projects/jwordnet/

70. *IDC Integration Drivers.* http://www.intelligenteai.com/020528/509razor.jhtml (viewed May 28, 2002)

71. *Resource Description Framework (RDF) Schema Specification version 1.0*: http://www.w3.org/TR/2000/CR-rdf-schema-20000327/

72. *SOAP/1.1 Note and the SOAP Version 1.2 Recommendation documents.*http://www.w3.org/TR/soap/

73. *The birth of Web service: October 2002 issue of MSDN Magazine* http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx (retrieved Oct. 2004)

74. *The Evolution of UDDI UDDI.org White Paper*: http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf

75. *Universal Description Discovery & Integration (UDDI) Version 3.0*, Published Specification, 19 July 2002 http://uddi.org/pubs/uddi_v3.htm#_Toc12653676

76. *Web Services Inspection Language (WSIL) Version 1.0,* November 2001, http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html

77. *Web Services Description Language (WSDL) 1.1 W3C Note* 15 March 2001 http://www.w3.org/TR/wsdl

78. *Web Services Architecture, W3C Working Draft* 14 May 2003 http://www.w3.org/TR/2003/WD-ws-arch-20030514/#id2608426

79. *WordNet – a lexical database for the English language*: Princeton University http://wordnet.princeton.edu/

80. *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation 28 October 2004 http://www.w3.org/TR/xmlschema-2/

81. *XQuery 1.0: An XML Query Language* (11 February 2005) http://www.w3.org/TR/xquery/

82. *XSL Transformations (XSLT) Version 1.0* W3C Recommendation 16 November 1999 http://www.w3.org/TR/xslt

Appendixes

A: Weather Category List of Web services

	Service Name	Web Services Location
1	AirportWeather	http://live.capescience.com:80/ccx/AirportWeather
2	DOTSFastWeather	http://ws2.serviceobjects.net/fw/FastWeather.asmx
3	GET_Weather	http://www11.brinkster.com/bgx/webservices/GET_Weather.asmx
4	GlobalWeather	http://www.webservicex.com/globalweather.asmx
5	HurricaneServiceService	http://weather.terrapin.com/soap/servlet/rpcrouter
6	ndfdXML	http://www.nws.noaa.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php
7	ndfdXML	http://weather.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php
8	Service	http://www.ejse.com/WeatherService/Service.asmx
9	USWeather	http://www.webservicex.com/usweather.asmx
10	WeatherByZip	http://www.innergears.com/WebServices/WeatherByZip/WeatherByZip.asmx
11	WorldWeatherByICAO	http://www.innergears.com/WebServices/WorldWeatherByICAO/WorldWeatherByICAO.asmx
12	WeatherInformationServiceService	http://www.ops-cij.gr.jp:8081/axis/services/weatherInformationService
13	WeatherService	http://www.learnxmlws.com/services/weatherretriever.asmx
14	WeatherService	http://www.lostsprings.com/weather/WeatherService.asmx
15	WeatherFetcher	http://glkev.webs.innerhost.com/glkev_ws/WeatherFetcher.asmx
16	WeatherForecast	http://www.webservicex.net/WeatherForecast.asmx
17	WeatherService	http://www.stanski.com/services/worldweather/weather.asmx

B: Operations List

	Service Name	Web Services Location	Operation Name
1	StockServices	http://glkev.webs.innerhost.com/glkev_ws/StockServices.asmx	GetQuotes
2	DOTSFastWeather	http://ws2.serviceobjects.net/fw/FastWeather.asmx	GetWeatherByZip
3	DOTSFastWeather	http://ws2.serviceobjects.net/fw/FastWeather.asmx	GetWeatherByIP
4	DOTSFastWeather	http://ws2.serviceobjects.net/fw/FastWeather.asmx	GetWeatherHistoricalByZip
5	DOTSFastWeather	http://ws2.serviceobjects.net/fw/FastWeather.asmx	GetWeatherByWMOID
6	GET_Weather	http://www11.brinkster.com/bgx/webservices/GET_Weather.asmx	Get_WeatherReport
7	GlobalWeather	http://www.webservicex.com/globalweather.asmx	GetWeather
8	ForceUnit	http://www.webservicex.net/ConvertForec.asmx	ChangeForceUnit
9	TorqueUnit	http://www.webservicex.net/ConvertTorque.asmx	ChangeTorqueUnit
10	CountryInfoLookupService	http://cs.uga.edu:8080/axis/services/urn%3acountryInfoLookup	CountryInfoLookup
11	CurrencyConverter	,http://www31.brinkster.com/webcomponents/CurrencyConverter.asmx	USDConvert
12	USWeather	http://www.webservicex.com/usweather.asmx	GetWeatherReport
13	MediCareSupplier	http://www.webservicex.net/medicareSupplier.asmx	GetSupplierByZipCode
14	DOTSEmailValidate	http://ws2.serviceobjects.net/ev/EmailValidate.asmx	ValidateEmail
15	WeatherByZip	http://www.innergears.com/WebServices/WeatherByZip/WeatherByZip.asmx	GetWeatherByZip
16	WorldWeatherByICAO	http://www.innergears.com/WebServices/WorldWeatherByICAO/WorldWeatherByICAO.asmx	GetWeatherByICAO
17	WeatherInformationServiceService	http://www.ops-cij.gr.jp:8081/axis/services/weatherInformationService	getWeather
18	WeatherService	http://www.learnxmlws.com/services/weatherretriever.asmx	LogOn
19	WeatherService	http://www.learnxmlws.com/services/weatherretriever.asmx	LogOff
20	WeatherService	http://www.learnxmlws.com/services/weatherretriever.asmx	GetWeather
21	WeatherFetcher	http://glkev.webs.innerhost.com/glkev_ws/WeatherFetcher.asmx	GetWeather
22	WeatherFetcher	http://glkev.webs.innerhost.com/glkev_ws/WeatherFetcher.asmx	GetLicWeather
23	WeatherForecast	http://www.webservicex.net/WeatherForecast.asmx	GetWeatherByZipCode
24	WeatherForecast	http://www.webservicex.net/WeatherForecast.asmx	GetWeatherByPlaceName
25	FreeFaxService	http://www.OneOutBox.com:80/cgi-bin/soap/outbox.cgi	SendFreeFAX

26	LocalTime	http://www.ripedev.com/webservices/LocalTime.asmx	LocalTimeByZipCode
27	NewsService	http://www.dotnetpro.de/xmlwebservices/news.asmx	GetLatestNews
28	NewsService	http://www.dotnetpro.de/xmlwebservices/news.asmx	GetLatestNewsSince
29	Phonebook	http://www.barnaland.is/dev/phonebook.asmx	Search
30	GeoPlaces	http://www.codebump.com/services/placelookup.asmx	GetPlacesWithin
31	USZip	http://www.webservicex.com/uszip.asmx	GetInfoByZIP
32	Service	http://www.ejse.com/WeatherService/Service.asmx	GetWeatherInfo
33	Service	http://www.ejse.com/WeatherService/Service.asmx	GetExtendedWeatherInfo
34	Service	http://www.ejse.com/WeatherService/Service.asmx	GetIraqWeatherInfo
35	WorldTime	http://upload.eraserver.net/circle24/worldtime/worldtime.asmx	GetTime
36	ZipcodeLookupService	http://www.winisp.net/cheeso/zips/ZipService.asmx	CityToZip

C: Sample List of Web services from our repository

Web services are collected using google APIs by searching for each file with extension WSDL

http://www.atomic-x.com/xmlservices/HyperlinkExtractor.asmx?wsdl
http://www.atomic-x.com/xmlservices/dnslookupservice.asmx?wsdl
http://ga-lms.cs.ait.ac.th:8081/axis/services/LMSService?wsdl
http://services.bio.ifi.lmu.de:1046/prothesaurus/services/BiologicalMarkupService?wsdl
http://services.bio.ifi.lmu.de:1046/prothesaurus/services/BiologicalNameService?wsdl
http://ws.strikeiron.com/GlobalAddressVerification?WSDL
http://ws.strikeiron.com/GlobalAddressVerification?WSDL
http://ws.strikeiron.com/IndianAddressVerification?WSDL
http://ws.strikeiron.com/StrikeIronDirectoryService?wsdl
http://www.bs-byg.dk/bzip2.wsdl
http://www.tradeshowdatabase.com/soap/service?wsdl
http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl
http://ws.strikeiron.com/MarketIndices?WSDL
http://www.xignite.com/xfunddata.asmx?WSDL
http://ws.strikeiron.com/ZacksSummary?WSDL
http://www.webservicex.net/WeatherForecast.asmx?WSDL
http://ws.strikeiron.com/DoNotCall?WSDL
http://www.xignite.com/xInvestorRelations.asmx?WSDL
http://www.quisque.com/fr/chasses/crypto/crypta.asmx?WSDL
http://www.xignite.com/xrates.asmx?WSDL
http://sms.idws.com/soap/smsservice.dll/wsdl/ISMSService
http://www.seshakiran.com/QuoteService/QuotesService.asmx?wsdl
http://wsdl.wsdlfeeds.com/odp.cfc?wsdl
http://www.webservicex.com/uklocation.asmx?WSDL
http://www.webservicex.com/hcpcs.asmx?WSDL
http://live.capescience.com/wsdl/FOPService.wsdl
http://digilander.libero.it/mamo78/KRSS_DAML_Service.wsdl
http://www.webservicex.com/country.asmx?wsdl
http://glkev.webs.innerhost.com/glkev_ws/StockServices.asmx?WSDL
http://www.stgregorioschurchdc.org/wsdl/Calendar.wsdl
http://www.esynaps.com/WebServices/DailyDiblert.asmx?WSDL
http://www.nims.nl/soap/oms.wsdl
http://www.SoapClient.com/xml/SQLDataSoap.wsdl
http://www.SoapClient.com/xml/SQLDataSoap.wsdl
http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl
http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
http://www.OneOutBox.com/wsdl/FreeFaxService.wsdl
http://www.drbob42.co.uk/cgi-bin/Euro42/wsdl/IEuro
http://www.foxcentral.net/foxcentral.wsdl
http://www.gxchart.com/webchart.wsdl

D: XML Schema built-in data types and categories

	Datatype	Category
primitive	string	string
	boolean	boolean
	float	float
	double	float
	decimal	float
	duration	dataTime
	dateTime	dataTime
	time	dataTime
	date	dataTime
	gYearMonth	dataTime
	gYear	dataTime
	gMonthDay	dataTime
	gDay	dataTime
	gMonth	dataTime
	hexBinary	binary
	base64Binary	binary
	anyURI	string
	QName	string
	NOTATION	string
derived	normalizedString	string
	token	string
	language	string
	IDREFS	idRef
	ENTITIES	string
	NMTOKEN	string
	NMTOKENS	string
	Name	string
	NCName	string
	ID	idRef
	IDREF	idRef
	ENTITY	idRef
	integer	integer
	nonPositiveInteger	integer
	negativeInteger	integer
	long	integer
	int	integer
	short	integer
	byte	integer
	nonNegativeInteger	integer
	unsignedLong	integer
	unsignedInt	integer
	unsignedShort	integer
	unsignedByte	integer
	positiveInteger	integer

Vita Auctoris

NAME:	Ismail Jaghmani
PLACE OF BIRTH	Zawia, Libya
YEAR OF BIRTH	1968
EDUCATION:	Bright Star University of Technology, Libya 1985-1990 B. in Sc Engineering
	Concordia University, Montreal Canada 1997-1999 M. Sc. In Engineering
	Concordia University, Montreal, Canada 2001-2002 G. D. Computer Science
	University of Windsor, Windsor, Canada 2002-2005 M. Sc. Computer Science