RC22999 (W0311-129) November 17, 2003
Computer Science

IBM Research Report

A Survey of Public Web Services

Su Myeon Kim
KAIST EECS Department
Yusung-Gu Gusung-dong 373-1, Taejon
Korea

Marcel-Catalin Rosu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

A Survey of Public Web Services

Su Myeon Kim*
KAIST. EECS Dept.
Yusung-Gu Gusung-dong 373-1, Taejon
KOREA
+82-42-869-3586
smkim@nclab.kaist.ac.kr

Marcel-Catalin Rosu
IBM T.J. Watson Research Center
19 Skyline Drive, NY
USA
+1-914-784-7242
rosu@us.ibm.com

ABSTRACT

Enterprise IT infrastructures and their interfaces to private partners and to the general public are migrating toward a service-oriented architecture, using Web Services (WS) as a de-facto implementation protocol. As a result, WS-generated traffic is expected to increase and have a considerable impact on the Internet. Despite the high amount of interest in WS, there have been relatively few studies regarding their characteristics**.**

In this survey, we analyze publicly-accessible WS using the information that we collected over a 3 month period. We study the evolution of WS and their geographic distribution, and message characteristics and response times of each WS. We closely analyze two popular WS sites: Amazon and Google. Some of our initial results contradict common intuition. The number of public WS has not increased dramatically, although there are signs which indicate intensive ongoing activities in the WS domain. The geographic distribution of public WS is largely skewed: about three fifths of public WS are located in USA. In contrast to existing Web content, WS response messages are just a little bigger than request messages, and the sizes of WS responses and their variation are smaller than those of the existing Web objects.

Categories and Subject Descriptors

A.1 [**Introductory and survey**]: - *Web Services, SOAP Traffic, Geographical distribution*

General Terms

Survey, Documentation, Measurement, Experimentation

Keywords

Web Services, SOAP, WSDL, UDDI Business Registry, Measurement, Web Services Traffic Characteristics.

1. Introduction

Enterprise IT infrastructures are currently migrating toward a service-oriented architecture, using Web Services (WS) as a de-facto implementation protocol. As the need to meet the increasing expectations of customers and business partners for real-time information exchange continues to rise, companies are motivated to integrate disparate systems within their organizations and also to interface with other organizations. A service-oriented architecture provides a framework for seamlessly interconnecting applications and software components, supporting loosely coupled software resources, such as distributed applications and objects. Ideally, remote business services can be invoked and/or installed as local components in a different application, all without writing a single line of low-level code [1]. Thus, WS help companies to greatly improve the flexibility and interoperability of their infrastructures.

**This research was done during a six-month visit at IBM T.J.Watson supported by a BK21 fellowship from the government of Korea.*

WS consist of a protocol stack of emerging standards characterized by a high degree of flexibility, connectivity, accessibility, and interoperability. By supporting service-oriented and component-based application architectures, WS provide a distributed computing technology for revealing the business services of applications on the Intranet as well as on the Internet using open and standard-based XML protocols and formats. The use of standard XML-based protocols makes WS platform-, language-, and vendor-independent, and so an ideal protocol for a service-oriented architecture.

WS are classified into three main categories according to their usage. First, there are WS designed only for use in an organization's Intranet, such as services for Enterprise Application Integration. Second, there are inter-organizational WS, which are operated by an organization but shared with other selected organizations. Third, there are public WS, which are intended for public use and therefore open to any other organizations. In this survey, we focus on the third category – publicly-accessible WS.

In spite of the wide acceptance of WS in computing infrastructures, there have been few studies on WS characteristics. Due to their inherent flexibility and interoperability, WS are expected to be adopted in every kind of IT infrastructure. In addition, since WS are supposed to be a new dominant communication protocol on the Internet, their impact on the Internet traffic may be significant. For instance, it is commonly accepted that message sizes on the Internet will grow significantly by the adoption of XML, the most basic element in the WS stack. These kinds of WS characteristics are of interest to researchers, developers, and network service providers.

In this paper, we analyze public WS in various ways, using publicly available information that we collected weekly, between August 8th and November 7th 2003 from an UDDI Business Registry (UBR). First, we study the evolution of the WS population and its geographic distribution. Second, we determine several characteristics of public WS such as preferred message styles, and distributions of complex and elementary types. Third, we develop a methodology for estimating WS message sizes. Fourth, we examine the liveness and response times of public WS, by probing their service ports. Lastly, using our methodology, we analyze the WS of two popular sites - Amazon and Google – and compare the message sizes predicted by our methodology with the message sizes observed during interactions with the two sites.

Our initial results contradict common intuition. First, the number of public WS has not increased dramatically, although there are certain signs which indicate that many intensive activities are ongoing in the WS domain. Second, the geographic distribution of public WS is largely skewed with about three fifths of public WS located in USA. Lastly, in contrast to existing Web content, response messages are just a little bigger than request messages and both the sizes of WS response messages and their variation are small. We expect our results to benefit WS applications and tools developers, and to improve our understanding of this emerging research area. This survey is part of an ongoing research and upcoming analysis results will be published on our web site [2].

The remainder of this paper is organized as follows. Section 2 provides a brief overview of WS usage in the service-oriented architecture and of the three most important elements in the WS stack: SOAP, WSDL, and UDDI. Section 3 describes our methodology for data collection and for estimation of WS message sizes, and the results of our analysis and experiments. Section 4 applies the techniques previously developed to the Amazon and Google WS. Section 5 is a brief overview of the related work. Section 6 is dedicated to conclusions and future work.

2. Web Services

WS specifications are relatively new and still evolving. Although WS are composed of many standards, four technologies are considered as the core ones: Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI). XML is a structured self-describing, data-neutral format that can be used to represent complex data as a simple text document. It has already been accepted as the universal language for information exchange. In this section, after describing the WS call process, we briefly describe three WS core standards: SOAP, WSDL and UDDI.

Figure 1 shows the WS call process. First, the service provider defines, designs, and implements a WS. In this process, the service provider generates a 'description file' which describes the new WS using WSDL. Second, the service provider publishes the new WS in a public UDDI registry using the UDDI WS-based API. Third, the client queries the UDDI registry and finds the WS. Once found, the client retrieves the WS description in WSDL. Lastly, the client invokes the WS one or more times using SOAP.

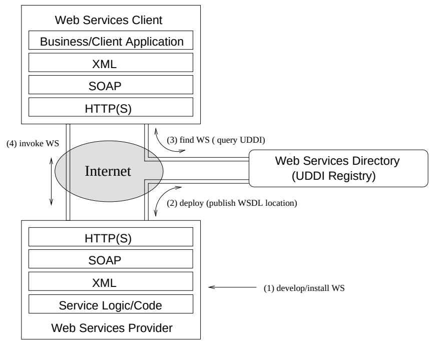


Figure 1. Web services call process

2.1 Simple Object Access Protocol

Simple Object Access Protocol (SOAP)[6],[7],[8] represents a standard for lightweight XML-based messaging protocol for Web Services. It enables the exchange of information between two or more peers in a decentralized, distributed application environment.

A SOAP message consists of an HTTP header and a SOAP envelope (see Figure 2). The envelope surrounds the SOAP header and body. The SOAP body is a regular payload or a SOAP fault. Only a WS response message may carry a SOAP fault instead of a return value.

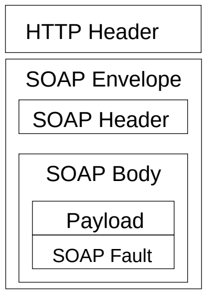


Figure 2. SOAP message structure

Table 1 shows a typical SOAP request message. The SOAP header is an optional part. The underlined are namespace declarations; most of them (SOAP-ENC, SOAP-ENV, xsd and xsi) are essential and present in every SOAP message. The SOAP payload part carries an application message. In Section 3, we use this information to estimate WS message sizes.

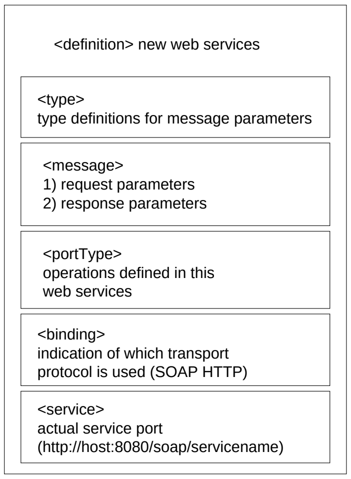
The most commonly used encodings in SOAP messages are the soap and the literal encoding. Encoding represents the action of transforming language specific data format to XML format (serialization) and vice versa (deserialization). When SOAP was first defined, the XML Schema (XSD) standard was not specified yet. As a result, SOAP had to define its own encoding rules as part of its specification.[1] Later, after the introduction of XSD, the literal encoding, based on XSD, became popular. There are significant differences between the soap and the literal encodings. For instance, messages following the soap encoding do not conform to a specific XML schema, so the validation of these messages is not possible.

SOAP messages can be classified into RPC or document style. There is little difference between these two styles, except in the SOAP body. Since the RPC style simulates an RPC invocation and response, an RPC-style request contains a method name to be invoked and input parameters. The response includes a return value and output parameters. In contrast, the document style does not impose any restriction on the message structure. There is no specific convention for how to specify a method name or a parameter. Actually, the biggest difference may be that most RPC-style messages use soap encoding while document-style ones use literal encoding. In Section 3, we use message style to classify WS.

[1] Since the encoding rules were defined in Section 5 of SOAP specification, the soap encoding rules are commonly known as Section 5 encoding.

2.2 Web Services Description Language

The Web Services Description Language (WSDL) [9],[10],[11] is used to describe WS in a common XML grammar. The WSDL document associated with a WS provides enough information to locate and access the methods of the WS. With WSDL-aware tools, clients can automate this process, enabling the integration of WS into existing applications with little effort. In Section 3, we will discuss WSDL files collection and the analysis process, which is essential to the investigation of WS characteristics.



<definition> new web services
<type>
type definitions for message parameters
<message>
1) request parameters
2) response parameters
<portType>
operations defined in this
web services
<binding>
indication of which transport
protocol is used (SOAP HTTP)
<service>
actual service port
(http://host:8080/soap/servicename)

Figure 3. WSDL document structure

Figure 3 shows the structure of a WSDL document. The five types of information in the structure are:

Types: data type definitions used in the message description,

Messages: abstract definitions of the data being transmitted,

PortTypes: collections of abstract operations, where each operation refers to an input and an output message,

Binding: concrete protocol and data format specifications for each portType,

Service: a set of related ports, where each port specifies a binding address, thus defining one communication endpoint.

2.3 Universal Discovery, Description, and Integration

UDDI is a technical specification for describing, discovering, and integrating WS [12]. UDDI is therefore a critical part of the emerging web services protocol stack, enabling companies to both publish and find WS.

A UDDI registry implementation is a WS-based registry that provides a mechanism to advertise and discover WS. The registry contains information about businesses and the services that they offer, and it associates some of those services with the technical specifications of the WS. These technical specifications are usually defined using WSDL and termed as *tModel*s. WS consumers query the UDDI registry to find WSDL descriptions.

The UDDI Project operates a global public registry called the UDDI Business Registry (UBR). All the information in this registry is available to everyone at no charge. All information in any one UBR node is automatically replicated to all other UBR nodes within 24 hours. Information on all the public WS can be collected by querying the *tModel*s of any UBR instance.

Table 1. An example SOAP message

HTTP Header	HTTP/1.1 200 OK Date: Thu, 19 Jun 2003 15:12:30 GMT Server: Stronghold/2.4.2 Apache/1.3.6 C2NetEU/2412 (Unix) mod_throttle/3.1.2 mod_fastcgi/2.2.12 Connection: close Content-Type: text/xml
SOAP envelope tag Namespace declaration (underline d italic-face)	<?xml version="1.0" encoding="UTF-8"?> <SOAP-ENV:Envelope *xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"* *SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"* *xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance/"* *xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"* *xmlns:xsd="http://www.w3.org/2001/XMLSchema"* *xmlns:amazon="http://soap.amazon.com"*>
SOAP body tag	<SOAP-ENV:Body>
Payload	<namesp1268:AsinSearchRequestResponse *xmlns:namesp1268="http://soap.amazon.com"*> <return xsi:type="amazon:ProductInfo"> <Details SOAP-ENC:arrayType="amazon:Details[1]" xsi:type="SOAP-ENC:Array"> <Details xsi:type="amazon:Details"> <Url xsi:type="xsd:string">http://www.amazon.com/exec/obidos/…</Url> <Asin xsi:type="xsd:string">0596002246</Asin> <ProductName xsi:type="xsd:string">Web Services Essentials (O'Reilly XML)</ProductName> <Catalog xsi:type="xsd:string">Book</Catalog> </Details> </Details> </return> </namesp1268:AsinSearchRequestResponse>
SOAP body tag	</SOAP-ENV:Body>
SOAP envelope tag	</SOAP-ENV:Envelope>

3. Data Collection and Analysis

The analysis in this section is based on the information we collected from an UBR every week, from August 8th to November 7th 2003. We search the UBR entries for *tModels* which refer to a WS.[2] We retrieve the WSDL file, and record retrieval delays and HTTP headers.

The analysis is divided into four main phases, each one described in a separate section. First, we analyze WS *tModels* and associated WSDL files. Second, we determine the WS characteristics from the WSDL files. Third, we study WS message sizes using realistic estimates for the variable-size fields in the WS description. Fourth, we measure the latencies of the WS endpoints, and compare them with the latencies of the corresponding HTTP endpoints.

3.1 Population and Geographic Distribution

There are more than 5000 *tModels* in a UBR; among them, about 1,000 *tModels* refer to WS. We noticed no significant changes in the number of WS *tModels* and valid WSDL files during the data collection period. Figure 4 summarizes the data collected.

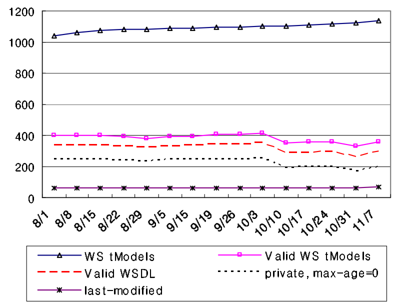


Figure 4. Web Services in UBR

The number of 'valid' WS *tModels* – *tModels* which have a URL where a WSDL file is retrievable - is substantially smaller than the total number of WS *tModels*: approximately 67% of the WS *tModels* are not valid, which is similar to the findings of a previous UDDI integrity study [25]. Furthermore, many of the downloaded WSDL files are invalid. The most common errors are syntax errors and omission of mandatory elements. During the three month interval, the number of valid WS decreases a little, which is contrary to the slight increase in the number of WSDL files published. Note that there is a small but noticeable decrease in the number of valid *tModels* on Oct. 10th due to a server hosting 54 web services becoming unavailable. [3] Finally, we found that very few organizations update their WSDL files after publication.

We measure the retrieval latency of WSDL files, as WS are expected to be discovered, integrated and invoked dynamically. Figure 5, which is based on measurements taken from IBM Watson, shows that 90% of WSDL files are retrieved within 400 msec while it takes almost 2 seconds to retrieve an additional 8%. In addition, we record the caching characteristics of the retrieved WSDL files. The bottom two lines in Figure 4, 'last-modified' and 'private, max-age=0', show the number of WSDL files which have cache control elements in the associated HTTP headers; the two elements are mutually exclusive. Together, they show that 92% of the valid WSDL files are cacheable in private caches.

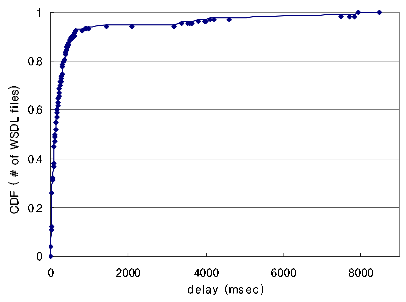


Figure 5. WSDL Retrieval Latency

We use the NetGeo service [13],[14] to map WS to their geographic locations. NetGeo and several other services which provide IP-based host information are using Whois[15] queries of internet repositories. NetGeo outperforms the other services by utilizing other result fields, such as phone number and e-mail, to infer missing fields in the Whois response, such as country and city.

Figure 6 (a) shows the geographic distribution of public WS on November 7th. 63% of the WS are hosted in United States. Figure 6 (b) shows the distribution of WS hosting sites on the same day. From the fact that the portion of sites in US is smaller, we can infer that a larger number of US-resident WS are hosted by the same site.[4]

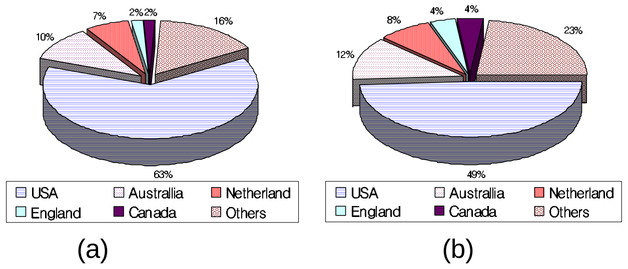


Figure 6. Geographical Distribution (a) WS, (b) Hosting Site

3.2 Styles and Structures

By design, a WSDL file includes a comprehensive description of the associated WS. WSDL file analysis exposes many of the WS characteristics, such as encoding type, message style, number of operations, and number of parameters for each operation. In Section 3.3, we use this information to estimate the size of WS request and response messages.

By inspecting the collected WSDL files, we found that there are many more document-style WS than RPC-style WS; the argument about which style is better is still an ongoing debate. Among the 294 valid WSDL files collected on November 7th, 70% define document-style WS and 30% define RPC-style WS. All of the document-style WS adopt the literal encoding and all of the RPC-style WS adopt the soap encoding. HTTPS is used by only 4% of these services, while the others use HTTP. Lastly, more than 74% of the WSDL files were generated with the Microsoft toolkit.

[2] UDDI is a complex specification which is beyond the scope of this paper. See http://www.uddi.org for more information on *tModels*.

[3] Microsoft's .Net WS contest server (http://www.contest.eraserver.net) hosts Web Services which receive Microsoft's Best of the .NET Awards.

[4] The largest hosting site is Visual Basic .NET XML Web Services Examples site (http://www.oakleaf.ws/)

To simplify the analysis, we translate the WSDL files into Java files using the WSDL2Java tool of Axis. WSDL2Java generates Java Bean files for each newly defined compound type, an interface file, and related implementation files.[5] We analyze these Java files instead of the original WSDL files.

Information may be lost during the translation. For instance, fixed-length arrays in WSDL are converted into Java arrays of unspecified length. Also, both base64Binary and hexBinary types of WS are mapped into the same type - Java byte array of unspecified size. Although these may cause inconsistency, this kind of information loss is rare and we believe that it can be ignored in our analysis. For instance, we found no WSDL files which use fixed-size arrays or hexBinary variables.

In the client Java application, WS operations are invoked as local functions, as the generated Java interface implements an RPC-style programming model. Note that the programming model is different from the binding style, as it is possible to provide an RPC-style programming model for a document-style WS.

To determine parameter complexity, we count the occurrence of array and compound types. In order to accurately count array types, we resolve all compound types into elementary or array types. Thus, we can count array types according to their dimension. For compound types, we record the complexity of member types. We classify compound type complexity according to the number of iterations needed to resolve them into elementary types.

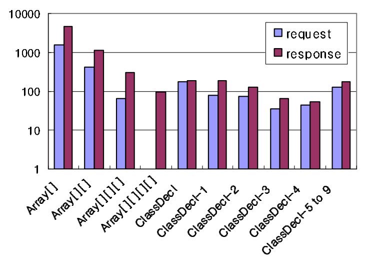
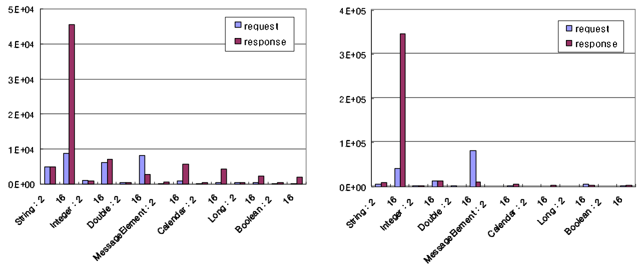


Figure 7. Frequency of Array and Compound Types

Figure 7 shows the frequency of array and compound types in the public WSDL files. The number following compound types represents the complexity of that type. For instance, ClassDecl means a class of elementary types, ClassDecl-1 means a class of elementary types or ClassDecl types, etc. The highest complexity class is ClassDecl-9; we found 4 and 10 ClassDecl-9 parameters in the request and response messages, respectively. Response messages utilize compound types (ClassDecl) and array types (ArrayDecl) more frequently than request message.

To estimate the usage frequency of the elementary types, we convert every class into the set of its member types and resolve arrays into elementary types. As most WS definitions do not specify array lengths, we assume three values for the length of all the arrays: 2, 16, and 32. For instance, length 2 means that each string [] type is converted into two strings, each Integer [][] type is converted into four integers, etc.

While examining the usage frequency of elementary types as a function of the selected array length, we observe that a few WSDL files dominate the results. Only five WSDL files use 3-dimensional arrays, out of which, one has all of the four-dimensional arrays; no five or higher -dimensional arrays were found. The frequency of elementary type usages is highly dominated by the 4-dimensional arrays. Thus, to screen out these biases, we exclude the WSDL file which uses 4-dimensional array from this analysis. Also, we run the analysis on the subsets of WSDL files with and without the remaining four WSDL files with 3-dimensional arrays.



(a) (b)

Figure 8. Frequency of Elementary Types: (a) excluding (b) including WSDL files using 3-dimension

Figure 8 (a) and (b) show the results for array lengths of 2 and 16 when 3-dimension arrays are excluded and included, respectively. The number following each type name denotes an array length. The figure shows that responses use more arrays than requests do, as the number of types in the response grows faster with array length than in the request. It also shows that the string type is the most frequently used type. Even when array length is 32, the proportion of string type is similar to when array length is 16 – 53% when WSDL files which declare 3-dimensional arrays are included and 71% when those files are excluded.

We also examine how many operations each WS provides. Figure 9 shows that 89% of WS have less than 10 operations. The names of more than 46% operations start with *get*. Other frequently used name prefixes are *add*, *delete*, *send*, etc. Owing to these simple functionalities, most WS operations use a small number of parameters.

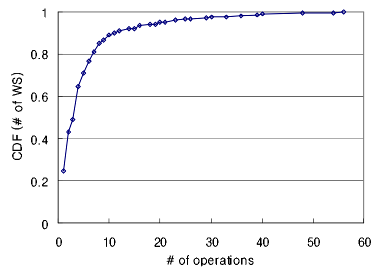
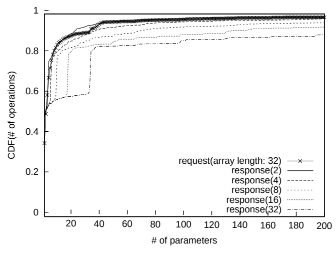


Figure 9. Distribution of Operations

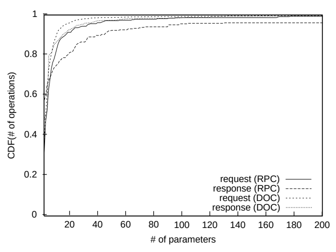
Finally, Figure 10 (a) shows how many elementary-type parameters each operation uses; we assume array lengths of 2, 4, 8, 16, and 32. Note that all but one line correspond to responses, as response messages are more sensitive to the selected array length than request messages (see Figure 8). First, most operations have simple functionalities, as 80% of them use no more than 10 parameters even when array length is assumed to be 8. Second, responses always use more parameters than requests.

[5] For details on WSDL2Java and its conversion processes, refer to the Axis User's Guide [16].

Figure 10 (b) shows these results separated into RPC-style and document-style WS when the length of the arrays is 2. In both cases, response messages have more parameters than request messages. Document-style responses have fewer parameters than RPC-style requests. This suggests that document-style messages are simpler than RPC-style ones.



(a)



(b)

Figure 10. Usage of Elementary-type Parameters: (a) Array Length - 2, 4, 8, 16, and 32, (b) RPC- vs. document-style when Array Length is 2

3.3 SOAP Message Size

Characterizing the size of SOAP messages is important since WS traffic is expected to become a prevailing traffic on the Internet. In this section, we first describe how we estimate SOAP message sizes using the information in the WSDL files. Then, we explain some meaningful characteristics of the SOAP message sizes.

As shown in Section 2.1, a SOAP message can be divided into four parts – HTTP header, SOAP envelope tag, SOAP body tag, and payload. Below is the equation used to infer the size of a SOAP message:

*SOAP message size = HTTP header + essential tag (SOAP envelope tag + SOAP body tag) + namespaces + payload (message tag + number of elementary type field in parameters * (type tag + value size))*

In this equation, ‘essential tag’ represents the SOAP envelope and body tags and ‘namespaces’ represents the aggregation of all occurrences of namespace attributes in a message. The namespace attribute can occur in the SOAP envelope tag, SOAP body tag, parameter tag, etc. The payload is composed of two parts: parameters and message tag.

We determine the size of each message component by examining real SOAP messages. We investigate several messages, including those of Amazon and Google WS (see Section 4). We observe that there are small variations in HTTP header and essential tag and that most messages use 5 ~ 7 namespaces. Four of these namespaces - SOAP envelope, XML schema, XML instance and encoding style - are essential for most SOAP messages.

In order to determine the payload size, we use the following methodology. First, we determine the size of the message and type tags. The message tag is used to wrap up the payload, which is a list of parameters (RPC-style) or a XML tree (document-style) and its size has a small variation. The type tag is used to declare the parameter names and types, and its size has a small variation, as well. According to our examination of real SOAP traffic, for RPC-style messages, the average size of message tag is 47 characters and that of type tag is 40 characters. For document-style messages, the average size of message tag is 57 characters and the average size of a type tag is 24 characters.

Second, we estimate the number of elementary type fields using the methodology developed in Section 3.2. Lastly, we determine the average size of the XML representations for the fields of each type. Table 2 shows average sizes and descriptions. For most numeric types, we assume their average sizes as small as possible. For instance, we use 5 characters as the average size of the long type, although it can be up to 20 characters. Thus, the resulting message size estimate is a practical lower-bound but not a theoretical lower-bound: the message size might be smaller than estimated. We assume that a larger data type (e.g., integer) is used when the parameter is expected to have a larger value than the maximum value of a smaller type (e.g., short).

Table 2. Average Size in XML Representation and Description of Types

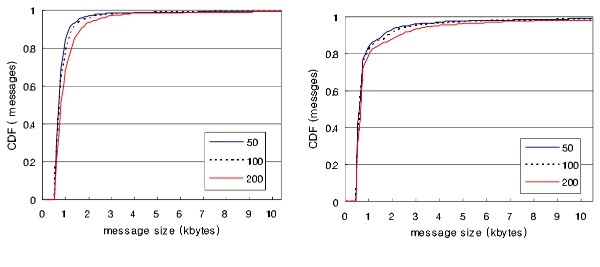
type	estimated size	description
Boolean	4	true/false
Short	2	-32768 to 32767
UnsignedShort	2	0 to 65535
Integer	3	-1,0,126789675
BigInteger	5	-1,0,12678967543233
UnsignedInt	3	1230 to 4294967296
PositiveInteger	3	1,12678967543233
Byte	2	-127 to 128
UnsignedByte	2	0 to 255
Long	5	-9223372036854775808 to 9223372036854775807
UnsignedLong	5	0 to 18446744073709551615
Double	5	64-bit floating type, -1E,12.78e-2
Float	5	32-bit floating type
BigDecimal	5	Arbitrary precision decimal number
Date	10	yyyy-mm-dd
Calendar	16	yyyy-mm-dd-hh-mm
MessageElement	10	xsd:any
Object	10	xsd:any
String	variable	* randomly distributed from min and max.
Qname	12	amazon:searchResult
Entity	10	color="yellow"
ClassDecl	40	compound type declaration tag
ArrayDecl	60	Array type declaration tag

String is the most frequently used type as shown in Section 3.2, and its size is the most dynamic. The string size may vary a lot according to the context of its message. We select a range for the string size, between a minimum and a maximum size, and assume that actual values are distributed uniformly within this range. In the rest of this section, the minimum size is always 5 characters, and the maximum size is 50, 100, or 200 characters.

ClassDecl and ArrayDecl in Table 2 have a different meaning than the other entries: ArrayDecl represents the size of the XML tag used to describe an array and ClassDecl represents the size of the tag used to declare a compound type. These two are not parameter values, but another kind of type tag.

Array length has a significant impact on SOAP message size since any type could be a base type for an array. In the following, we assume that all the arrays have the same length. In the beginning, we assume that arrays have only two elements. Later, we use 16 and 32 for the array length.

In the rest of this section, we first examine the differences derived from changes in string size distribution or array lengths. Second, we compare our results on SOAP message sizes with existing Web object sizes. Lastly, we examine the difference between RPC-style and document-style SOAP messages.



(a) (b)

Figure 11. SOAP Message Size when Array Length is 2 - (a) request (b) response

Figure 11 (a) shows the CDFs of WS request messages when the maximum string size is 50, 100, 200; array length is assumed to be 2. It shows that 93% of request messages are smaller than 2KB even when maximum string size is 200. In contrast, most HTTP requests are smaller than 500bytes. The HTTP request sizes usually show bimodal distribution, with one large peak occurring around 250 bytes and another, smaller, around 1KB [22]. Figure 11 (a) shows that the size of WS requests has different characteristics from that of HTTP requests.

Figure 11 (b) shows the CDFs of WS response messages. Similar to Figure 11 (a), most messages are small: 88% of response messages are smaller than 2KB, even when the maximum size of a string is assumed to be 200 characters. The figure also suggests that string size has little impact on small messages, as these messages use few parameters.

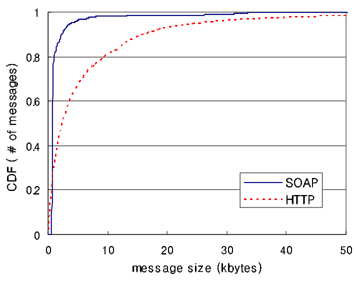
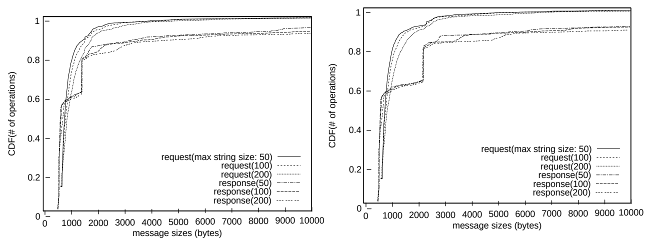


Figure 12. SOAP vs. HTTP : Array Length is 2 and Maximum String size is 200

Next, we compare the distributions of SOAP messages to that of existing Web content (see Figure 12). For Web content, we use the model presented in [27]. In contrast to other recently developed models such as that in [28], this model screens out the population factor of unique files; this approach is compatible with our analysis of WS message sizes.

Contrary to the common expectation that SOAP messages are larger than current HTTP messages due to XML formatting, most SOAP messages are smaller than existing Web objects. For instance, while about 92% of SOAP messages are smaller than 2KB, only 45% of the existing Web objects are smaller than 2KB.

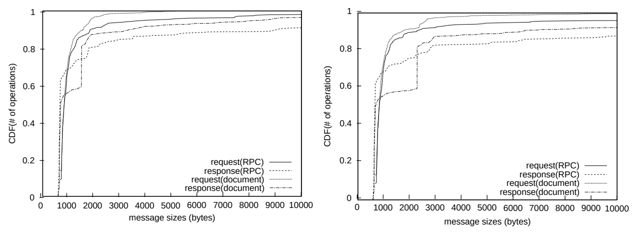
Next, we investigate the message size distribution when we assume array length is 16 and 32 (see Figure 13 (a) and (b), respectively); we assume maximum string length of 50,100, and 200. In both figures, the variations between lines are small. Note that there is little difference between the two figures when the message size is smaller than 500bytes, which suggests that a small number of operations use a large number of arrays.



(a) (b)

Figure 13. SOAP Message Size when Array Length is (a) 16, (b) 32

The same analysis is performed separately for RPC-style and document-style WS. Figure 14 (a) shows the message sizes when the array length is 16, maximum string size is 50. Figure 14 (b) shows the same where array length is 32. Both figures show that the response size of RPC-style message grows much faster than the response size of document-style messages. This suggests that existing RPC-style WS use more complex response messages than document-style WS.



(a) (b)

Figure 14. RPC vs. document SOAP Message Size when Array Length is (a) 16, (b) 32

The results presented in this section show that SOAP message characteristics are different from those of existing Web traffic. The WS request and response sizes follow a similar distribution. Contrary to common expectation, current SOAP messages are not necessarily larger than existing Web objects.

While this methodology provides an estimated rather than exact SOAP message size, we believe that this is a useful way to get an understanding of the size and complexity of SOAP messages. WS researches may use this early result as a starting point for a more detailed model. Network service providers may use this methodology for traffic estimation.

3.4 Liveness and Invocation Delay

The server providing the WSDL file is typically unrelated to the server hosting the WS. Therefore, it is required to verify the liveness of the WS directly.

To validate the liveness of public WS, we wrote a small program called Web Services Ping (WSPing). The current version of WSPing only supports http/https. WSPing accesses the endpoints specified in the WSDL files. It sends a SOAP message to the WS endpoint and waits for a response. Note that ICMP ping utility or Web robots cannot examine the presence of WS engines.

WSPing sends a simple SOAP message which has a valid HTTP header and SOAP envelope. The message is shown in Figure 15. It has only one field which is a message to indicate that it is not a malicious attack along with our e-mail address. Since the message does not conform to the required message format, the response is a SOAP fault: the server cannot understand our request message. If the response conforms to a valid SOAP fault message format, the WS is considered alive.

Our weekly experiments show that: approximately 16% of the valid WS are down and that 96% of the live WS respond in two seconds or less. Figure 16 shows the CDF of response times for WS as well as Web servers, as measured on November 13th; measurements performed on other dates show similar results. When probed from two locations, IBM Watson and KAIST, about 85% of WS servers are alive, and about 2~3% more Web servers are alive. Our attempts to measure ping delays do not show any meaningful results, as most sites block ICMP ping messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <Request
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <dummy xsi:type="xsd:string"> These requests are sent for an
academic research purpose. Please send an e-mail to
smkim@nclab.kaist.ac.kr if any problem. Thanks </dummy>
  <Request>
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 15. WS probing message

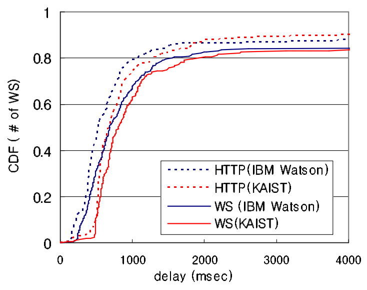


Figure 16. WS and HTTP response delay

4. Case Studies

In this section, we analyze the Amazon and Google WS. We analyze both WS according to our methodology: start by analyzing WSDL files and continue with estimating SOAP message sizes and access delays. We then compare the estimated message sizes with the actual message sizes.

Amazon and Google provide a WS API for their original services; Amazon provides item browsing and purchasing, and Google provides Web searching functionalities. Both WS are open to the public and require only a simple pre-registration. They both provide WS development toolkits. The toolkit helps clients integrate the functionality into their programs and web sites. Figure 17 shows the relative position of the toolkit in the client-side protocol stack.

For both sites, we analyze real messages by recording incoming and outgoing traffic for each operation, including the HTTP header. On the client side, we use Apache Axis. To capture messages, we use tcpmon, a message capture utility that comes with Axis. We determine the average size of each SOAP message segment based on the captured messages. Table 3 shows the results.

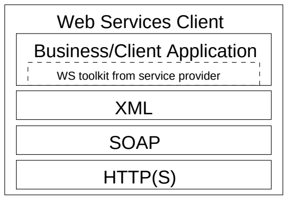


Figure 17. Client-side WS protocol stack

Table 3. Average Size of SOAP Message Components

	Amazon		Google	
	Request	Response	Request	Response
HTTP header	281	189	164	189
Namespace (x frequency)	47 (x 6)	48 (x6)	46 (x5)	48 (x6)
essential tags XML & Envelope	104	106	108	108
message tags	77	77	41	57
type tag	40	45	36	52

4.1 Amazon

Amazon provides their WS for associates, suppliers, or developers. The 'associates' program is a business model enabling 3rd party web site operators to link their web sites to Amazon and earn referral fees for the sales made through their links. Amazon actively supports their WS: version 1.0 was released in July 2002 with basic shopping capabilities; version 2.0 was released in October 2002; lastly, version 3.0 was released in April 2003 with an expanded API for 3rd party suppliers and shopping cart handling. In addition to the main US Amazon site, the WS API is supported for the Amazon sites in UK, Japan, and Germany. The WS Toolkit, including examples, can be downloaded from the Amazon WS home [23].

The main Amazon WS site is located in US and it is operated by Amazon itself, i.e., not outsourced. Their WS operations use only string types: 279 elementary strings, 778 one-dimensional, 702 two-dimensional, and 40 three-dimensional string arrays. Most of these strings and string arrays are used in response messages, as only 179 elementary strings and 9 string arrays are used in request messages. For the complexity of messages, they use 7 ClassDecl, 39 ClassDecl-1, 32 ClassDecl-2, and 2 ClassDecl-3 compound parameters. Among these 80 compound parameters, only 2 ClassDecl are used in request messages.

Amazon WS v3.0 API has 20 operations, shown in Figure 18. We classify the operations according to their functionalities into - Product Browse operations and Shopping Cart operations. Then, first-level operations are classified according to their response message type. These types are shown as ovals. Lastly, the second-level operations are classified according to request message type. As a result, operations in the same leaf node have the same request and response message types. HTTP and WS response delays are 327 and 502msec when measured from IBM Watson, and 501 and 510msec from KAIST.

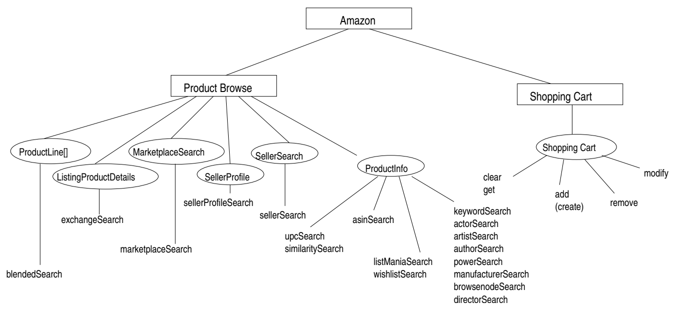
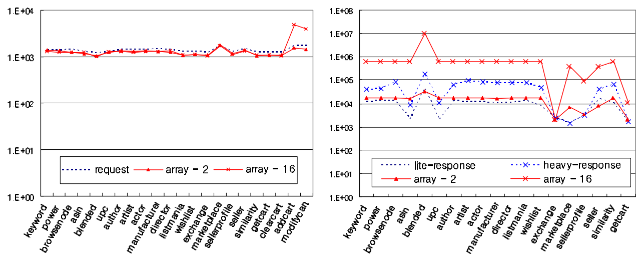


Figure 18. Operation Tree of Amazon Web Services

Figure 19 (a) and (b) show the message sizes, both real and estimated, for requests and responses, respectively. We assume maximum string size of 50 characters and array length of 2 or 16. Note that the browse operations have two kinds of responses – lite and heavy. A lite response delivers the summary of the selected items, while a heavy response delivers all the available information. The fixed size components of both lite and heavy are identical but the payload varies widely.



(a) request (b) response

Figure 19. Amazon WS Message Size

Figure 19 (a) shows that our estimation of request sizes are accurate. When the array length is 2, there is little difference between real message sizes and estimated ones. The only large gaps, when array size is assumed to be 16, are due to the fact that the addcart and modifycart operations use string arrays.

Figure 19 (b) shows the response message sizes. It shows that our estimations are inaccurate. However, it should be noted that the pattern of lines are almost identical and the line for heavy-response is between estimated lines. To improve accuracy, application specific information is needed.

4.2 Google

Google provides a WS API to their Web search engine in order for developers to embed Google search functions into their programs. The Google WS API was launched in April 2002 and is still at the beta version [24].

The Google API has only three operations: *doGetCachedPage*, *doSpellingSuggestion*, and *doGoogleSearch*. These operations use 14 elementary strings, 11 one-dimensional string arrays, 5 Booleans, and 5 Integers. Both a ClassDecl and a ClassDecl-1 are used in response messages of the *doGoogleSerch* operation. Figure 20 shows the three operations. HTTP and WS response delays are 292 and 329msec when measured from IBM Watson, and 841 and 1046msec from KAIST.

Figure 21 shows the message sizes of Google WS. We assume maximum string size of 50 characters and array length of 2 or 16. The figure shows that our estimation of message sizes is accurate except for the response message size of *doGetCachedPage*. In this case, as Google returns a cached Web page as a single parameter of byte[] type, array length should be much larger than 16.

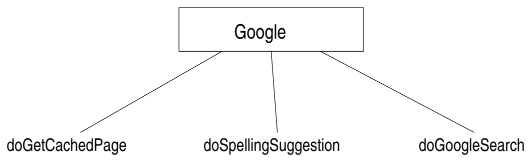


Figure 20. Google Web Services

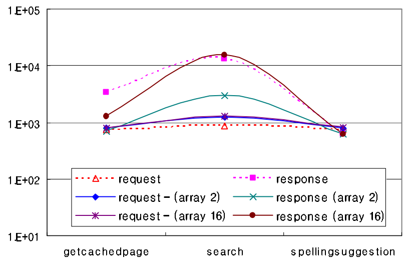


Figure 21. Google WS message size

5. Related Work

To the best of our knowledge, this is the first survey of public WS. Our work is related to Web evolution studies [35], [36] and to existing activities in Internet geography drawing, WSDL file analysis, and WS portals.

Mapping and analysis of the Internet geography are studied by many researchers [29],[30]. They studied the geographic location of Internet components such as end nodes and routers. Our investigation considers a geographic distribution of WS endpoints, not just IP-level internet nodes.

There are many WSDL analysis tools: xmethod.net's WSDL analyzer [17], WSDL Explorer of IBM alphaworks[18], Mindreef's SOAPscope [19], IBM's Web Services quick tester [20], and Bindingpoint QuickTry [21]. However, all these tools except SOAPscope analyze WSDL files to invoke Web Services automatically. SOAPscope checks if the WSDL file is well-formed, as defined in core WS specifications. Our WSDL file analysis provides more sophisticated results: the structure, style, location, and expected SOAP message sizes.

WS portals [31],[32],[33],[34] provide information about their WS, including category, rate, price, and service explanation. Most of this information targets WS consumers. We investigate the evolution, internal structures, and message characteristics to improve the understanding of WS technology.

6. Conclusion

Enterprise IT infrastructures and their interfaces to private partners and general public are currently migrating toward a service-oriented architecture, using WS as a de-facto implementation protocol. In this survey, we analyze publicly-accessible WS using the information that we collected over the past 3 months. Public WS are services made available over the Internet to any other organization. Our analysis uses information collected weekly from an UDDI Business Registry.

We study several aspects of public WS. First, we determine the population and geographic distribution of WS. Second, we determine the characteristics and preferred message styles. Third, we develop a methodology for estimating WS message sizes. Fourth, we examine the liveness and response time of each WS, by probing their service ports. Lastly, using our methodology, we analyze the WS of two popular sites - Amazon and Google – and compare the message sizes predicted by our methodology with the message sizes observed during interactions with the two sites.

Our initial results show that the number of public WS does not increase dramatically and that about three fifths of the current WS population is based in USA. In addition, our results indicate that there are substantial differences between WS traffic and the existing Web traffic.

We plan to extend our survey by collecting more WSDL information from other sources. We also plan to refine our methodology for WSDL analysis as well as message size estimation. For instance, we plan to use WS operations semantics to estimate string and arrays lengths. This survey on public WS is part of an ongoing project and upcoming analysis results will be published on our web site [2].

7. REFERENCES

[1] Peter Fletcher and Mark Waterhouse, Web Services Business Strategies and Architectures, Expert press, Birmingham, UK., 2002

[2] http://nclab.kaist.ac.kr/~smkim/ws_survey/index.html

[3] Ethan Cerami, Web Services Essentials, O'reilly, CA, USA, 2002

[4] Eric Rescorla, SSL and TLS - Designing and Building Secure Systems, Addison-Wesley, IN, USA, 2001

[5] Ramesh Nagappan, Robert Skoczylas, Rima Patel Sriganesh, Developing Java Web Services - Architecting and developing Secure Web Services Using Java, Wiley Publishing, Inc., IN, USA., 2003

[6] http://www.w3.org/TR/soap12-part0/, SOAP Version 1.2 Part 0: Primer, W3C Recommendation

[7] http://www.w3.org/TR/soap12-part1/, SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation

[8] http://www.w3.org/TR/soap12-part2/, SOAP Version 1.2 Part 2: Adjuncts, W3C Recommendation

[9] http://www.w3.org/TR/wsdl12-bindings, Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language, W3C Working Draft

[10] http://www.w3.org/TR/wsdl12-bindings, Web Services Description Language (WSDL) Version 1.2 Part 2: Message Patterns, W3C Working Draft

[11] http://www.w3.org/TR/wsdl12-bindings, Web Services Description Language (WSDL) Version 1.2 Part 3: Binding, W3C Working Draft

[12] http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3, UDDI Specification

[13] http://www.caida.org/tools/utilities/netgeo/, NetGeo web site

[14] http://www.netgeo.com, Internet Geography Guide, NetGeo Infoscope White Paper

[15] Whois database, http://www-whois.internic.net/cgi/whois

[16] http://ws.apache.org/axis, Apache Axis home page

[17] http://www.xmethods.net, WSDL Analyzer

[18] http://www.alphaworks.ibm.com/tech/wsdlexplorer, WSDL Explorer

[19] http://www.mindreef.com, SOAPscope

[20] http://www-106.ibm.com/developerworks/webservices/demos/quicktest/index.html, IBM VisualAge Smalltalk Web services Quick Tester

[21] http://www.bindingpoint.com/QuickTryv2.aspx, Bindingpoint QuickTry

[22] Bruce A. Mah, An Empirical Model of HTTP Network Traffic, INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE , Volume: 2 , 7-11 April 1997, Page(s): 592 -600 vol.2

[23] http://www.amazon.com/webservices, Amazon web services home

[24] http://www.google.com/apis/, Google web APIs home

[25] Mike Clark, UDDI – The Weather Report (The outlook is missed),http://www.webservicesarchitect.com/content/articles/clark04.asp

[26] http://www.contest.eraserver.net, the Contest site for Microsoft's Best of the .NET Awards Academic competition

[27] Paul Barford and Azer Bestavros and Adam Bradley and Mark Crovella", Changes in Web Client Access Patterns: Characteristics and Caching Implications, special Issue on Characterization and Performance Evaluation, 1999

[28] Maurizio Molina, Paolo Castelli, and Gianluca Foddis, “Web Traffic Modeling Exploiting TCP Connections’ Temporal Clustering through HTML-REDUCE”, IEEE Network Magazine, vol. 14, no. 3, pp. 46-55,2000

[29] http://www.cybergeography.org/atlas/geographic.html, An atlas of cyberspaces

[30] A. Lakhina and J. Byers and M. Crovella and I. Matta, *On the Geographic Location of Internet Resource*, Technical Report BUCS-TR-2002-015, Boston University, 2002.

[31] http://www.xmethods.net, Xmethods

[32] http://www.salcentral.com/salnet/webserviceswsdl.asp, salcentral – web services brokerage

[33] http://www.remotemethods.com/, remotemethods

[34] http://www.bindingpoint.com/, BindingPoint

[35] Junghoo Cho and Hector Garcia-Molina, “The evolution of the web and implications for an incremental crawler”, Porc. Of the Twenty-sixth Intl. Conf. on VLDB, Cairo, Egypt, 2000

[36] Dennis Fetterly, Mark Manasse, Mark Najork, and Janet Wiener, “A Large-Scale Study of the Evolution of Web Pages”, WWW2003, May 20-24, 2003, Budapest, Hungary