

Situated Web Service: Context-Aware Approach to High-Speed Web Service Communication

Ikuo Matsumura¹, Toru Ishida^{1,2}, Yohei Murakami², and Yoshiyuki Fujishiro¹

¹ *Department of Social Informatics, Kyoto University, Kyoto 606-8501 JAPAN*

² *National Institute of Information and Communications Technology, Kyoto 619-0289 JAPAN*

matumura@ai.soc.i.kyoto-u.ac.jp, ishida@i.kyoto-u.ac.jp

yohei@nict.go.jp, fujishiro@kuis.kyoto-u.ac.jp

Abstract

A framework is proposed to improve Web Service performance based on context-aware communication. Two key ideas are introduced to represent a client context; (1) available protocols that the client can handle, and (2) operation usage that shows how the client uses Web Service operations. We call our context aware approach a Situated Web Service (SiWS). We implemented and evaluated the SiWS and found that the overall performance was improved if more than three Web Services were executed between context changes.

1. Introduction

Web Services, which generally utilize XML-based SOAP¹ as a data exchange protocol, can hardly achieve the same high performance as other distributed computing technologies, such as CORBA and DCOM. The key principle of Web Services is that by utilizing the Internet, Web Services enable heterogeneous software components to perform a platform-independent remote procedure call (RPC) or a simple data exchange. However, the verbosity of XML, which is good for interoperability, in turn affects performance of Web Services. This problem becomes serious when Web Services are used in those emerging pervasive devices in which network bandwidth and processing power are limited. So, studies have been undertaken to improve performance of Web Services from various points of view.

Recently, using standard workflow languages, such as BPEL², an increasing number of systems consisting of one or more various Web Services have been developed. In such diverse systems, former performance improvement methodologies targeting uniform Web Services are sometimes ineffective in improving performance.

For example, Language Grid [2], which composes one or more language services deployed as Web Services into composite Web Services according to a user's demand, is one such system. As a result of creating composite services according to a user's demand, diverse operations

would appear from those that exchange short strings (ex. text chat messages) to those that exchange large amounts of data, such as complex data structures (ex. concept structure such as WordNet³ and ontology) or voice data (ex. data for speech recognition and synthesis). When we are to achieve real-time applications, such as multilingual chat on Language Grid, poor performance of Web Services may become a serious problem, and the diverse interfaces required may further compound the problem.

In this paper, we present a context-aware approach called Situated Web Service (SiWS) to improve performance of Web Services with diverse interfaces and various clients. We also present two concepts to represent client context. Finally, we show an implementation of the SiWS and evaluate it.

2. High-Speed Web Service Communication

Various technology and research have been proposed to improve performance of Web Services:

1) Binary encodings considered for text, XML and SOAP

Technology has been proposed to avoid high cost in processing and transmitting XML by expressing XML, which is originally a character-based format, as binary encoding. When we apply these approaches to existing Web Services components, both client and server sides must be changed because a transmitted message contains specific binary data and does not conform to SOAP.

First, it is easy to think of applying famous text compressing technology, such as gzip, into the presentation layer of the OSI model. Although it becomes more effective beneath the transport layer because of byte efficiency, this approach makes performance worse in the presentation layer. It is because the extra processes of encoding XML from text to compressed binary data, and vice versa, will be added to normal process of SOAP.

Second, XBIS⁴ and Fast Infoset⁵ are methods that improve the cost of XML processing and transmitting. They utilize binary encodings specially designed for

¹<http://www.w3.org/TR/soap/>

²<http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>

³<http://wordnet.princeton.edu/>

⁴<http://xbis.sourceforge.net/>

⁵<http://asn1.elibel.tm.fr/xml/finf.htm>

XML. In these methods, the messages before and after processing are not identical in terms of text but are identical in terms of logical XML. For instance, before and after processing, two messages may differ in the number of white spaces or line separators, but the messages themselves are identical as an XML document.

Although these two approaches are effective when the compressed data size and the complexity of computation for compression are balanced, it is difficult to adjust the balance for all kinds of data (text, binary, large, small, etc.)

Third, Fast Web Service⁶ is a method for improving performance of Web Services based on Fast Infoset and a specification (ITU-T X.694) that maps XML Schema and ASN.1⁷. Fast Web service communicates in binary encoding which is designed for a particular message schema and not for a general SOAP message. In this method, the transmitted message contains no XML tags or namespace declarations and redundancy of the transmitted message is smaller than XBIS and Fast Infoset, although the self-describing nature of the message is lost.

Although the third approach is effective when both client and server can handle the same protocol, this approach reduces interoperability. Performance problems cannot be solved on various clients by Fast Web Service alone because such a specialized protocol is not widely used, and we often have to use SOAP with some other services.

2) Utilization of similarity of XML in SOAP

XML messages repeatedly exchanged in SOAP have many common static parts, and the dynamically changing parts tend to be smaller than the static parts (See [5] 2.3). There are methods that utilize SOAP message features to reduce processing costs of XML in which different oriented processing with earlier processed message(s) can be performed [5]. Such methods only require one side of the client or server to be changed because the message transmitted is a raw XML document and conforms to SOAP specifications. This method improves the cost in the presentation layer of the OSI model, even though it does not improve the cost beneath the transport layer.

Although these approaches are effective when there is sufficient similarity in XML, these approaches are ineffective when the messages are quite different, such as with large data parameters.

3) Switching multiple protocols

WSIF [1] is a framework that enables more effective communication in more than one protocol, in addition to SOAP, by specifying and describing protocol specific WSDL binding and preparing each implementation, called a provider.

⁶<http://java.sun.com/developer/technicalArticles/WebServices/fastWS/>

⁷<http://asn1.elibel.tm.fi/>

The WSIF approach is powerful because any protocol can be used without requiring protocol specific coding as long as the client and the server are satisfied. However, invoking a very large operation interface, which may be created by composite Web Services based on user demands, and forcing every client to send all parameters to such a large interface causes performance problem.

So, to cope with performance problems on such potentially diverse Web Services, we have to develop more comprehensive methodologies.

3. Context-Aware Approach

3.1. Situated Web Service (SiWS)

To comprehensively deal with this problem, we utilize *context* that encapsulates the diverse situation of clients. We use available protocols and static parameters as context and develop an approach where the servers adapt to this client's context. We call this a Situated Web Service (SiWS) approach. In the SiWS approach, servers prepare a customized communication environment for each definite client.

The model of request and response in this approach should be changed from the standard model. The standard invocation model is a simple one consisting of sending parameters and response, as shown in Figure 1.

The invocation model for this approach should be as shown in Figure 2.

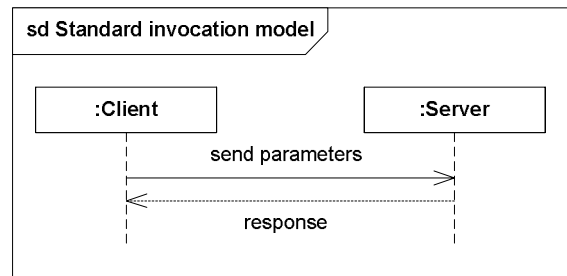


Figure 1: Standard invocation model

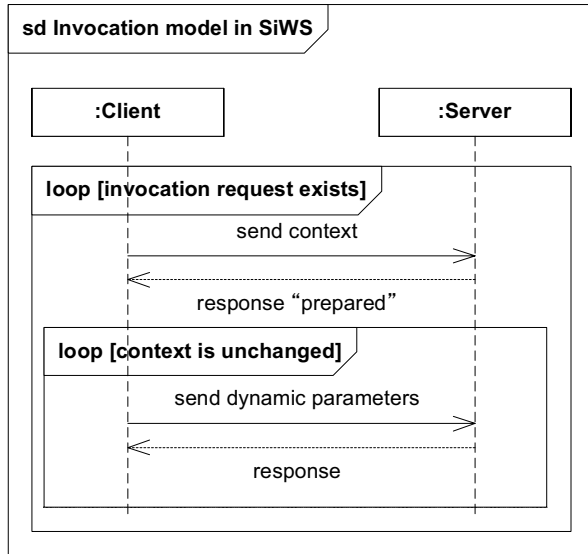


Figure 2: Invocation model in SiWS

First, a client sends a context to a server, and, second, the server prepares a customized communication environment for the received context. Next, the client only sends the dynamic parameters on the basis that the client context has not changed. At this time, when the dynamic parameters are sent, the fastest protocols available in both the client and server sides are used in the communication.

3.2. Definition of Context

A context is information that is unchanged during a number of requests. Although a context may include a wide range of concepts, we have limited context to those that concern performance improvement. Here we define a context, C , as follows:

$$C = (P, S)$$

where P are the available protocols and S is the property we call operation usage.

Since available protocols do not frequently change during a number of requests, available protocols can be regarded as a context. There are two cases when available protocols change:

1. The network environment between the client and the server is changed.
2. A certain protocol module is added or disabled.

When the environment is changed, for example, a client located outside a server's fire-wall is moved inside the firewall – after the movement of the client, they would become able to communicate in another non-HTTP based protocol in addition to the HTTP based protocol. This situation rarely occurs as compared to the intervals for a number of requests in both cases. So, available protocols can be regarded as a context.

We call parameters that are known to be unchanged during the sequence of a number of requests *static parameters*. In contrast, we call parameters that change frequently *dynamic parameters*. Whether a certain parameter is to be a static or a dynamic parameter depends on information regarding the way in which the client uses the operation of the server. We call this information operation usage. Operation usage can be regarded as a context because it does not change frequently.

3.3. Acquisition of Context

A client application uses a stub when it invokes a Web Service. To adapt to a client context at the server side, the stub at the client side should acquire the context beforehand. Available protocols are easily acquired by a stub. However, there are two ways for a stub to acquire an operation usage:

Static acquisition would be performed when a client application gives its operation usage to the stub. Since a stub can obtain information from the client application, and no computation is required in the stub, static acquisition achieves high performance but requires extra coding in a client application.

Dynamic acquisition would be performed when a stub itself detects an operation usage from the parameter sequence of a number of requests. Since a stub has the same interface as the standard Web Services, we can reuse existing client applications. However, a stub has to perform more calculations to detect operation usage at run time.

Although both acquisitions have advantages and disadvantages, for the remainder of this paper we will only consider static acquisition because it can achieve higher performance.

3.4. Usage of Context

We described the way to acquire a context in 3.3. Once the client context for a server is defined, how should a context be used for performance improvement? We propose a method in which a context-specific bypass is created, as shown in Figure 3.

After notifying operation usage to a stub, the client application's first time invocation reaches the server application through the stub and SOAP modules. The stub sends a client context with dynamic parameters, and the server generates a bypass (a context-dependent service handler) at the server side in parallel. After that, the client application can make an invocation through the generated bypass optimized for the context.

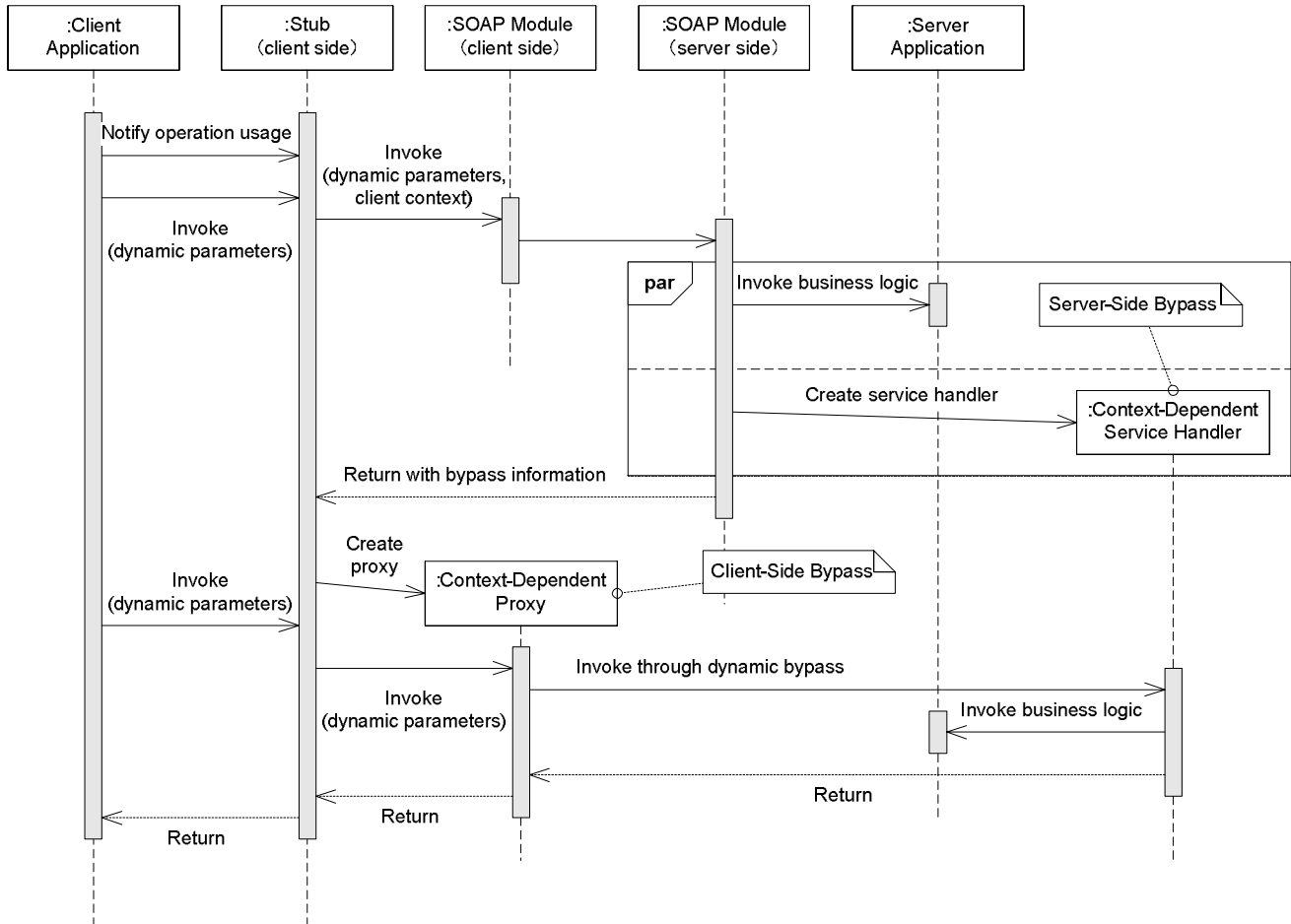


Figure 3: Sequence diagram of invoking service and creating a bypass

4. Implementation of SiWS

We described the definition of a context and how to deal with a context in section 3. In this section, we describe implementation for SiWS.

4.1. Process Flow

A block diagram of the SiWS implementation is shown in Figure 4. There are two methods for communication. One method is communication by a SOAP protocol, and the other method is communication by a dynamically generated bypass adapted to a client context. The client and server have alternative faster protocol modules, such as CORBA or Java RMI, that are usable in a certain network environments.

Service invocation flow from a state of no bypass is as follows:

1. The client application notifies its operation usage S to the stub; (a).
2. The stub detects available protocols P and obtains a context C from S and P .
3. The client sends its context to the server through the SOAP channel in the first invocation; (b) and (c).
4. The server selects one protocol p after receiving context C ; (d), which is to be used in a bypass from those protocols the server can handle. When selecting protocols, the server utilizes description of relative protocol speeds given by a system designer.
5. The Server-side controller generates a context-dependent service handler based on C and p ; (e).
6. As a response of the SOAP channel, the server sends back information of the generated bypass to the client; (f). Bypass information includes information related to p .
7. The client receives bypass information; (g).
8. The client generates a context-dependent proxy based on C and p ; (h).
9. The client application makes a request using the generated proxy through the stub; (i) and (j).
10. The server application is to be executed through the generated service handler; (k).

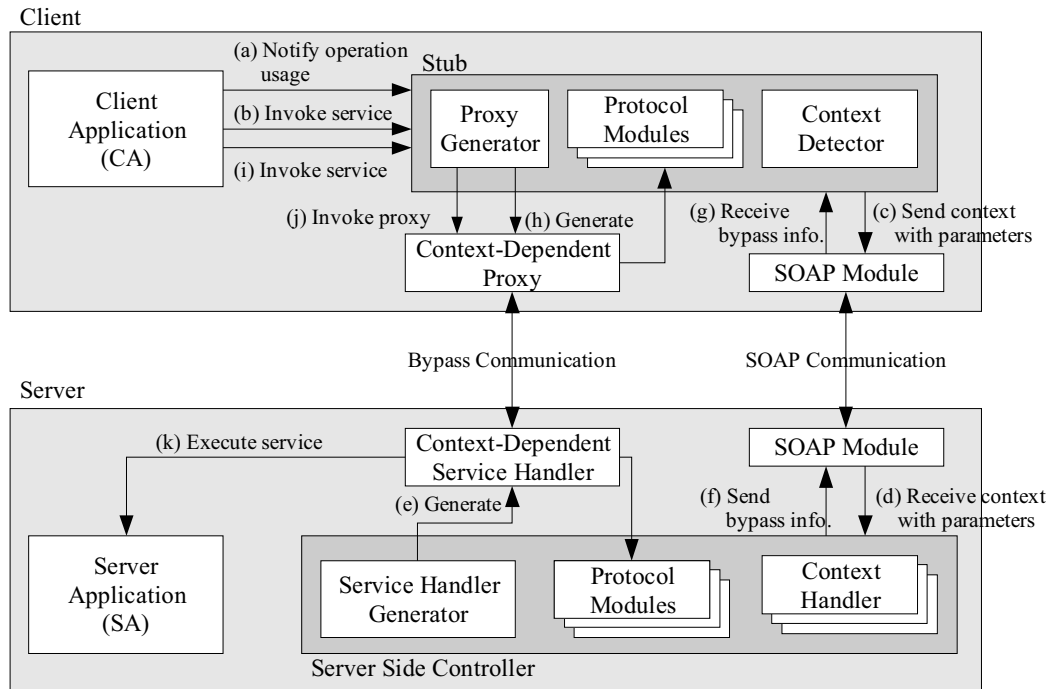


Figure 4: Block diagram of SiWS implementation

After a bypass is first established using the previously mentioned steps, the communication is performed on (i)-(j)-(k) route as long as the client context remains unchanged. When the client context changes, the steps we previously introduced must be performed and a new bypass will be established.

4.2. Schemas for Context/Bypass Information

We described that client context and bypass information must be transmitted from the client to the server and vice versa. Such information transmission is performed utilizing the header of the SOAP message. For this purpose, we developed schemas for context and bypass information in the XML Schema.

1) Schema for Context Information

A schema for context information is shown in Figure 6. The element `clientContext` represents the root of context information. The `clientContext` has two child elements: `operationUsage` and `availableProviders`. The former represents an operation usage, and the latter represents available protocols.

The `operationUsage` element describes static parameters as an operation usage. In Web Services of the RPC model, operation usage is described in the `staticIndex` element where the numbers of input parameter index are static parameters. This designation of index starts with 0 and not with 1. In Web Services of a

document-oriented model, as there is no concept of input parameters, the `staticRange` element instead of `staticIndex` element describes the static parameters. The designation of static parameters in the document-oriented model is to be performed using a language that appoints a range of XML document, such as XPointer⁸.

The `availableProviders` element describes a list of available protocol identifiers as a string.

An example of context information described in the SOAP header is shown in Figure 5. In this example, by describing 1 and 0 in the `staticIndex` element, which is enclosed by the `operationUsage` element, indicates that the first two input parameters are static parameters. Still, by describing identifiers `rmi` and `fastinfoset`, indicates that the corresponding two protocols are available at the client side.

```
<clientContext xmlns=
"http://i.kyoto-u.ac.jp/siws/soap/">
  <operationUsage>
    <staticIndex>1</staticIndex>
    <staticIndex>0</staticIndex>
  </operationUsage>
  <availableProviders>
    <provider>rmi</provider>
    <provider>fastinfoset</provider>
  </availableProviders>
</clientContext>
```

Figure 5: Example of context information

⁸<http://www.w3.org/TR/xptr/>

```

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace=
    "http://i.kyoto-u.ac.jp/siws/soap/"
  xmlns:tns="http://i.kyoto-u.ac.jp/siws/soap/"
  xmlns:Q1="xs">

  <element name="clientContext">
    <complexType>
      <all minOccurs="1" maxOccurs="1">
        <element name="operationUsage"
          type="tns:OperUsage"
          maxOccurs="1" minOccurs="1">
        </element>
        <element name="availableProviders"
          type="tns:AvailableProviders"
          maxOccurs="1" minOccurs="1">
        </element>
      </all>
    </complexType>
  </element>

  <complexType name="OperUsage">
    <sequence>
      <choice>
        <element name="staticIndex"
          type="nonNegativeInteger"
          maxOccurs="unbounded" minOccurs="0">
        </element>
        <element name="staticRange"
          type="string"
          maxOccurs="unbounded" minOccurs="0">
        </element>
      </choice>
    </sequence>
  </complexType>

  <complexType name="AvailableProviders">
    <sequence>
      <element name="provider" type="string"
        maxOccurs="unbounded"
        minOccurs="1">
      </element>
    </sequence>
  </complexType>
</schema>

```

Figure 6: XML Schema for context information

```

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace=
    "http://i.kyoto-u.ac.jp/siws/soap/"
  xmlns:tns="http://i.kyoto-u.ac.jp/siws/soap/"
  xmlns:Q1="xs">

  <element name="bypassInfo">
    <complexType>
      <sequence>
        <element name="provider"
          type="string">
        </element>
        <element name="param" type="tns:Param"
          maxOccurs="unbounded" minOccurs="0">
        </element>
      </sequence>
    </complexType>
  </element>

  <complexType name="Param">
    <simpleContent>
      <extension base="string">
        <attribute name="key" type="string"
          form="unqualified">
        </attribute>
      </extension>
    </simpleContent>
  </complexType>
</schema>

```

Figure 7: XML Schema for bypass information

2) Schema for Bypass Information

A schema for bypass information is shown in Figure 7. The `bypassInfo` element represents the root of the bypass information and has two child elements: `provider` and `param`. The `provider` element appoints a string identifier of protocol, and this is used for a bypass. The `param` element describes information needed to access the generated bypass, and they are described by a pair consisting of a key and a value. The `key` attribute describes the name of a key, and the text node of the `param` element describes the string value of a key.

An example of bypass information described in the SOAP header is shown in Figure 8.

In this example, by describing `rmi` in the `provider` element, which is enclosed by the `bypassInfo` element, indicates that a bypass based on a corresponding protocol (Java RMI in this case) has been prepared. Also, information in the `param` element describes a key whose name is `rmi_name` and whose value is `rmi://localhost:11099/siwsObj0`.

```

<bypassInfo xmlns=
"http://i.kyoto-u.ac.jp/siws/soap/">
  <provider>rmi</provider>
  <param key="rmi_name">
    rmi://localhost:11099/siwsObj0
  </param>
</bypassInfo>

```

Figure 8: Example of bypass information

Semantic interpretation of param elements depends on protocol modules. In this example, the content of rmi_name key indicates an address of a remote Java RMI object.

5 Experiments

In the SiWS implementation we described in section 4, it is obvious that performance improvement is achieved in the bypass communication because bypass generally utilizes faster protocols than SOAP. However, performance is reduced when preparing a context-dependent bypass, and we cannot improve performance on the whole when the client context changes frequently. In this section, we describe experiment and an evaluation to understand how frequently changing context affects performance.

5.1. Language Service Domain

We experimented with a Web Service: “SimpleTranslation” for language translation that contained port type of WSDL, as shown in Figure 9. The port type SimpleTranslation has one operation: to translate. The translate operation has three input parameters: sourceLang, targetLang and text. Each input parameters indicates source language, target language and the target translation text. To measure only the time for communication, SimpleTranslation was implemented to perform no other function, it just returns fixed text.

Targeting SimpleTranslation, we examined time in the following three cases:

- (a) Invocation by former SOAP
- (b) Invocation by SOAP in the SiWS implementation
- (c) Invocation by bypass in the SiWS implementation

Case (b) is the case in which the SiWS framework prepares a bypass.

The client context (an operation usage and available protocols) must be specified in cases (b) and (c). We set sourceLang and targetLang from the three input parameters as static parameters in operation usage and their static values were set as follows:

(sourceLang, targetLang) = (“en_US”, “ja_JP”).

We only set Java RMI as the available protocols.

```

<wsdl:message name="translateResponse">
  <wsdl:part name="translateReturn"
    type="xsd:string"/>
</wsdl:message>

<wsdl:message name="translateRequest">
  <wsdl:part name="sourceLang"
    type="xsd:string"/>
  <wsdl:part name="targetLang"
    type="xsd:string"/>
  <wsdl:part name="text"
    type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="SimpleTranslation">
  <wsdl:operation name="translate"
    parameterOrder=
      "sourceLang targetLang text">
    <wsdl:input
      message="impl:translateRequest"
      name="translateRequest"/>
    <wsdl:output
      message="impl:translateResponse"
      name="translateResponse"/>
  </wsdl:operation>
</wsdl:portType>

```

Figure 9: Port Type of SimpleTranslation

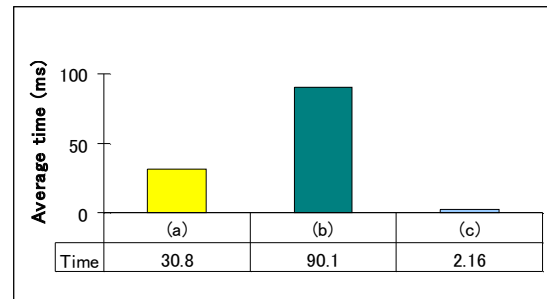


Figure 10: Average invocation time

So information exchanged in (c) as a dynamic parameter was data corresponding to the text input parameter, and the protocol used in (c) was Java RMI.

Results showing the average of these three cases and 100 run times are shown in Figure 10.

5.2. Results

From the above result, we evaluated how frequently changing context affects performance.

Let average invocation time (a), (b) and (c) be s , r' , and r . Assuming k is the number of context changes, n is the number of requests, we find that f is the ratio of the sum of request time to that of the former SOAP and is given as:

$$f = \frac{kr' + (n - k)r}{ns}$$

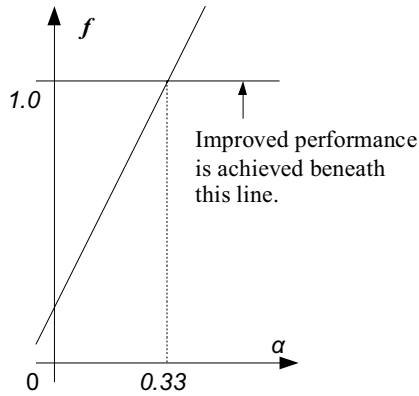


Figure 11: $f - \alpha$ graph

Applying the results (s , r' , and r) = (30.8, 90.1, and 2.16) to the equation above, we can obtain:

$$f = 2.86\alpha + 0.07 \quad (\alpha = k/n).$$

Figure 11 illustrates the above equation.

Where the bound in which performance improvement is to be achieved ($f = 1$), α is:

$$\alpha = 0.33.$$

From this result, we find that when about more than three requests per context change are performed, improved performance is achieved on the whole.

6. Conclusion

We indicated that there are still performance problem with Web Services that contain diverse interface and various clients, especially those created by general user as composite Web Services.

To solve this problem, we presented a context-aware approach that utilize context. Context allows clients to have diverse situation and includes two key ideas: (1) available protocols that the client can handle, and (2) operation usage that shows how a client uses Web Service operations.

Utilizing available protocols as context, we can handle the problem of higher cost in processing and transmitting XML and the problem of opportunity loss of using more effective protocols. We can also handle the problem of

redundancy in sending static parameters by utilizing operation usage as a context.

We proposed an architecture in which the system statically acquires a context and dynamically creates a bypass. Our architecture improves the performance of Web Services while preserving its wide-use nature.

We implemented the architecture and evaluated how frequently changing context change affects performance. We found that the overall performance was improved if more than three requests were executed between context changes in a conceivable client context

Although limitations to frequently changing context exist by SiWS, SiWS would solve performance problem when dealing with highly diverse Web Services.

Acknowledgments

This work was supported by a JSPS Grant-in-Aid for Scientific Research (18200009) and the Language Grid Project of National Institute of Information and Communications Technology.

References

- [1] Matthew J. Duftler et al., Web Services Invocation Framework (WSIF), *Proc. of the OOPSLA Workshop on Object-Oriented Web Services*, 2001
- [2] Toru Ishida, Language Grid: An Infrastructure for Intercultural Collaboration, *IEEE/IPSJ Symposium on Applications and the Internet (SAINT-06)*, 2006, pp. 96-100.
- [3] Markus Keidl, and Alfons Kemper, Towards Context-Aware Adaptable Web Services, *Proc. of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004, pp. 55-65
- [4] Lalitha Suryanarayana et al., Profiles for the Situated Web, *Proc. of the 11th international conference on World Wide Web*, 2002, pp. 200-209.
- [5] Toshiro Takase et al., An Adaptive, Fast and Safe XML Parser Based on Byte Sequences Memorization, *Proc. of the 14th international conference on World Wide Web*, 2005, pp. 692-701.