September 27, 2005

RT0628

Computer Science; Human-Computer Interaction; Service Science 12 pages

Research Report

Complexity Evaluation of Web Service Interfaces for End Users

Takayuki Yamaizumi, Takashi Sakairi, Masaki Wakao, Hideaki Shinomi

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Complexity Evaluation of Web Service Interfaces for End Users

Takayuki Yamaizumi[1], Takashi Sakairi[1], Masaki Wakao[2], Hideaki Shinomi[2]

[1] Tokyo Research Laboratory, IBM Japan Ltd.
1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa 242-8502, Japan
{zumi|sakairi}@jp.ibm.com
[2] Yamato Software Laboratory, IBM Japan Ltd.
1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa 242-8502, Japan
{wakao|shinomi}@jp.ibm.com

Abstract. Providing simple interfaces which can be easily handled by people who have neither programming skills nor deep understanding about Web service is a key factor in releasing Web services to end users as well as developers. This paper describes an algorithm to evaluate the usability of Web services by evaluating the complexity of the structures that are provided as the input of the Web service methods. The proposed algorithm is quite simple, because the outputs of the algorithm can be written as a scalar value which can be calculated by reading and analyzing the WSDL (Web Services Definition Language) file on a Web service provider.
The usability of the methods which are defined in some Web services that can be accessed through the Internet were analyzed with the algorithm proposed in this paper. The evaluation results of the proposed algorithm are discussed from some perspectives such as the number of methods, the average complexity of each Web service, and the complexity comparison between the Web services which has the similar functionality. This algorithm will also encourage end users to adopt the SOA (Service-Oriented Architecture) by combining Web services with simple interfaces in the distributed environment.

1 Introduction

Service-Oriented Architecture (SOA) is an emerging methodology to construct enterprise systems [1], as well as to integrate the existing systems into a larger system. Web services now play an important role in SOA because most SOA systems will be constructed by combining Web services.

Furthermore, many Web services that are open to the public can be accessed via the Internet [4]. They are quite useful for skilled developer who can implement the interfaces for Web services with stub code because these Web services prevent them from "reinventing the wheel". Likewise, the public Web services can be accessed by end users who have neither programming skills nor deep understanding of their specifications through the interfaces generated from WSDL

(Web Services Definition Language) files [5, 8]. However, since some of the Web services require a specifying a complicated data structure for the input data, users must have a deep understanding about the specifications of the Web services if they want to handle complicated data, and this should not be required of end users. In such cases, if there is an other public Web service that offers the same functionality and needs a simpler data structure as an input, the users would be helped if they had a way to evaluate the simplicity of the Web service, so that they could use the Web services with simpler interfaces. For this reason, some sort of evaluation of Web services for end users is needed, although at the present time of writing, there is no system associated with Web services which have the functionalities to evaluate Web services. The similar evaluation functionalities are also needed for expert users, because when the private UDDI (Universal Description, Discovery, and Integration) [22] registry is merged into an other UDDI registry, then an administrator wants to select the one of the Web services if both of the registries have a Web service that can operate similarly to those in the other Web service.

In this paper, we describe an algorithm based on GOMS (goals, operations, methods and selection rules) task model analysis [12] to evaluate the usability of Web services by evaluating the complexity of the data structures which are used as the input of the Web service methods. We have also used the proposed evaluation algorithm on 101 popular Web services that can be accessed through the Internet. The evaluation results of the proposed algorithm are discussed from some perspectives such as the number of methods, the average complexity of each Web service, and the complexity comparison between the Web services that have the similar functionality.

The paper is organized as follows: In Section 2, related work is discussed. The Web service interface evaluation algorithm is proposed in Section 3, with a definition of an end user operation model. The results of the evaluations with the proposed algorithm are described in Section 4. Section 5 discusses the results of the evaluations, and Section 6 gives the overall conclusions of this paper.

2 Related Work

To develop the enterprise system, business process behavior based on Web services can be defined in the Business Process Execution Language for Web Services (BPEL4WS[2], WS-BPEL[3]). BPEL4WS (WS-BPEL) is one of the core technologies to realize SOA [2, 3]. It is possible to use this to define an executable business process which determines the nature and sequence of Web service interactions. Tool support has been provided to make the definition in BPEL4WS easier. However, these tools are not usable by end users, and a developer must have a deep understanding and do the analysis of the business processes themselves and of the interaction of the Web services which the developer wants to define in the BPEL file.

In contrast, the most primitive approach to work with Web services is to implement an application to send SOAP messages between Web services [6].

Most of the work, especially the implementation for communication, is eased by a tool which can generate stub code by parsing a WSDL file. However, a user still must have some programming skills to work with Web service because the user must code some user interfaces.

The WSDL file which defines the interface of a Web service includes the XML schema [7] in which the data structure for the Web service are defined. Hence, a Web interface can be generated by reading the data structure definitions by reading a WSDL file on the Web service provider [8] and users can confirm and test the functionality of the methods in the Web services. However, from the end users' point of view, if a method in a Web service requires complicated or very large amounts of data, that method is too difficult to be used by end users, because the end users must have a deep understanding about the data structures of the Web services.

Web Services can be discovered from some UDDI registries, but it is difficult to find which Web service is the best for what user needs. One of the approaches to find the most suitable Web services is to find a similar Web service based on the semantics of the identifiers of the WSDL file and the structure of their operations [9].

For usability evaluation automation, Ivory et al. describe a taxonomy of usability evaluation automation [10]. They surveyed 132 usability evaluation methods. They classified the GOMS family of analysis methods into analytical modeling methods. They note that GOMS analysis can only be applied to an user model for an expert user. However, if there are few differences between an expert user model and a novice user model, then this approach is very useful. John et al. compare members of the GOMS family of analysis methods [13].

Fischer et al. discussed which domain should be selected for end-user development in [14]. They concluded that it is safe to adapt end-user developments to less complex domains. Thus, the end-user development approach can be adapted to Web services which are open to the public, although management and security issues still remain to be solved. End-user programming is recognized as an important issue because an interface should be customized to the need of particular users, although there are likely to be generic structures, e.g., in an email filtering system [11] which could be provided as a Web service, because it can be shared.

3 Web Service Complexity Evaluation

Web service complexity is calculated by evaluating the structures of the input data for a Web service with the algorithm which is described in this section. The evaluation consists of the following:

- The operation model of end users who cannot understand about either the data structure nor the structure of the Web service itself. This operation model is derived from GOMS analysis.
- The evaluation algorithm to calculate the complexity score by applying the operation model to the structure of the input data.

In the reminder of this section, the details of the Web services evaluation method is described.

3.1 The Operation Model for End Users

GOMS analysis [12] facilitates the quantitative evaluation of user interfaces and prediction of a task model. These evaluations and predictions are limited to error-free expert performance in many cases. Thus, these evaluations and predictions cannot be used in a straightforward way to any type of user interfaces because a user has to work with various types of interfaces even if the user sets data values such as name, address, account number, and so on, and the optimization methods may differ between those used by expert users and by end users.

We focus on the interfaces for the input data structures for Web services which can be generated from the definitions which are declared in a WSDL file. The data structures for input are represented as "tree-view like" interfaces such as shown in Fig. 1.

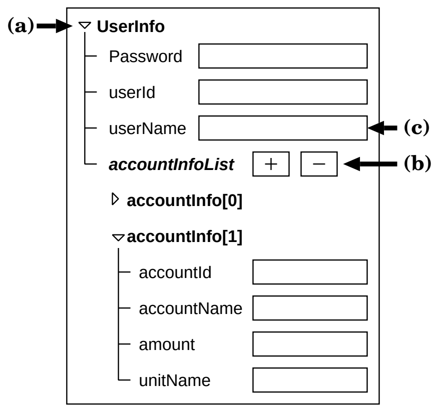

Fig. 1. "Tree-view like" interface sample for a Web service. (a) is a node handler button to expand (or collapse) the node for complex type, (b) is a button to add (or delete) an element of an array, and (c) is text field to set data.

This view only includes three types of interfaces (buttons, text fields, and a node handler) for an input which can intuitively be understood by end users. If an end user is asked to work with this display without any informations about the data structure of the input to the Web service or if an end user lacks the ability to understand their specifications, the following operations are available for a end user:

– Click a node handler button ((a) in Fig. 1) to expand (or collapse) to check the data that are included in the complex type data, which is defined with

a `complexType` tag in a WSDL file [7]. The sample definitions of data structures with a `complexType` tag are shown in Fig. 2. This operation is expressed as a task T_n. An end user may check the type name of complex type data. However, this is omitted from the task T_n because it is difficult to understand the meanings of the type names for an end user.
- Click a button ((b) in Fig. 1) to add (or delete) an element in an array. This operation is expressed as a task T_a.
- Type input data into the text field ((c) in Fig. 1). This operation is expressed as a task T_t. An end user may check the field name. However, this is omitted from the task T_t because it is difficult to understand the meanings of the type names for an end user as well.

Thus, an operation model of an interface for end users is defined as the sequence of T_is which are one of T_a, T_n, and T_t:

$$M = \{T_1, T_2, T_3, \cdots, T_m\} \tag{1}$$

where:

- $T_i \in \{T_a, T_n, T_t\}$,
- $1 \leq i \leq m$,
- and M represents the sequence of operations.

In the initial setting for the most of the toolkits [15, 16], all nodes in a tree view are collapsed. In addition, the elements of arrays are represented by child nodes of an array node. Thus, all nodes have to be expanded and at least one element has to be added to each array if a user wants to check the entire data structure for the inputs.

Let a, n, and t be a number of T_a, T_n, and T_t operations that are included in M. If a user wants to check the entire data structure for inputs, a and t can be regarded as constant values. Furthermore, if the user does not collapse nodes which have been expanded by the user, then n has a minimum value n_{min} because when the data structure is represented by a tree, all nodes and leaves can be reached by following the connection between the nodes [17]. Thus, m has an absolute minimum value which is m_{min}, represented by:

$$m_{min} = a + n_{min} + t \tag{2}$$

3.2 Web Service Complexity Evaluation Algorithm

Based on the discussion in the previous section and Equation (2), We propose to define the "complexity index" c as follows, and then determine the coefficients x_n, x_a, and x_t:

$$c = \sum_{j=1}^{l} ((x_a)_j a_j + (x_n)_j (n_j)_{min} + (x_t)_j t_j) \tag{3}$$

```
<schema targetNamespace="http://ws.bankdemo.act.ibm.com"
        xmlns="http://www.w3.org/2001/XMLSchema">
 <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
 <complexType name="AccountInfo">
  <sequence>
   <element name="accountId" type="xsd:int"/>
   <element name="accountName" nillable="true" type="xsd:string"/>
   <element name="amount" type="xsd:double"/>
   <element name="unitName" nillable="true" type="xsd:string"/>
  </sequence>
 </complexType>
 <element name="AccountInfo" nillable="true" type="impl:AccountInfo"/>
 <complexType name="UserInfo">
  <sequence>
   <element name="accountInfoList" nillable="true" type="tns1:ArrayOfAccountInfo"/>
   <element name="password" nillable="true" type="xsd:string"/>
   <element name="userId" nillable="true" type="xsd:string"/>
   <element name="userName" nillable="true" type="xsd:string"/>
  </sequence>
 </complexType>
 <element name="UserInfo" nillable="true" type="impl:UserInfo"/>
</schema>
<schema targetNamespace="http://ws.bankdemo.act.ibm.com/UserInfo"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
 <complexType name="ArrayOfAccountInfo">
  <complexContent>
   <restriction base="soapenc:Array">
    <attribute ref="soapenc:arrayType" wsdl:arrayType="impl:AccountInfo[]"/>
   </restriction>
  </complexContent>
 </complexType>
</schema>
<schema targetNamespace="http://ws.bankdemo.act.ibm.com/BankInfo"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
 <element name="UserInfo" nillable="true" type="impl:UserInfo"/>
 <complexType name="ArrayOfUserInfo">
  <complexContent>
   <restriction base="soapenc:Array">
    <attribute ref="soapenc:arrayType" wsdl:arrayType="impl:UserInfo[]"/>
   </restriction>
  </complexContent>
 </complexType>
 <element name="ArrayOfUserInfo" nillable="true" type="tns2:ArrayOfUserInfo"/>
</schema>
```

Fig. 2. Data structures as defined in a WSDL file

where the input of the Web service method consists of the data structures $d_1, d_2, \cdots d_l$.

These three coefficients can be determined from the numbers of operations. As discussed in the previous section, they can be expressed as the numbers of operations which can be easily understood by an end user. Hence, x_a, x_n, and x_t can be determined for all j which satisfies $1 \leq j \leq l$ as follows:

$$((x_a)_j, (x_n)_j, (x_t)_j) = (1, 1, 1) \tag{4}$$

The value in the left-hand side of Equation (4) can be assigned to the right-hand side of Equation (3), yielding Equation (5) as follows:

$$c = \sum_{j=1}^{l} (a_j + (n_j)_{min} + t_j) \tag{5}$$

The value of c can be calculated with the following algorithm:

1. Read the data structures in a WSDL file as described in Fig. 2 from a Web service provider to construct the data tree (V_j) for each argument as shown in Fig. 3.
2. Count the number of nodes which are represented by circle in Fig 3.
3. Repeat from the first step for all the other arguments, if any.

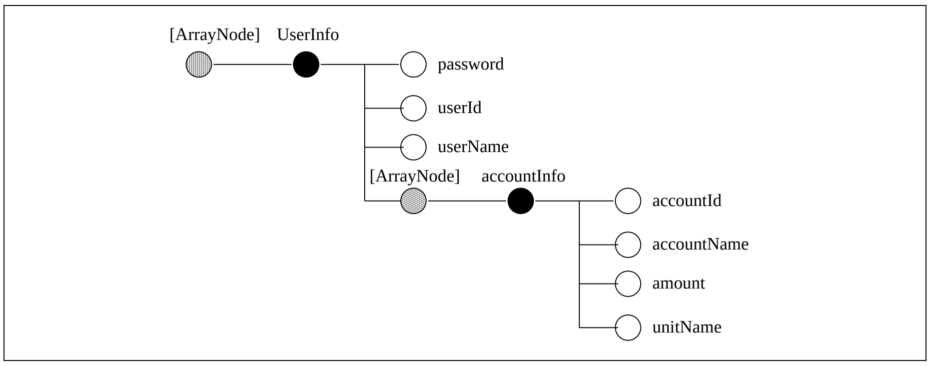

Fig. 3. Data tree V_j generated from the definitions in WSDL file in Fig. 2.

If a data structure is defined recursively as shown in Fig. 4, it is evaluated by one of the following rules, when the evaluation algorithm reaches to count the node (a):

– Children of the node (a) are not counted.
– The complexity value of this method is define as infinity (∞).

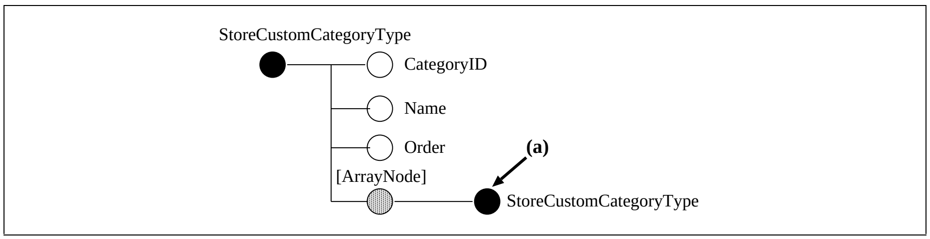

Fig. 4. Data tree which is defined recursively [20].

4 Evaluations of Web Services with the Proposed Algorithm

To evaluate the algorithm proposed in Section 3.2, we implemented the evaluation tool using SWT [15], in order to read and to parse the WSDL files which describe the specifications of Amazon E-Commerce Services [18], Google [19], eBay [20] and those of the Web services which are collected by the Web Service Club [23] as well as those collected by XMethod [4]. We have evaluated 101 Web services, Most of the Web services are collected by XMethod.

4.1 Distribution of the Complexities

First of all, we focus on a minimum value of complexity for each Web service. The distribution of the minimum values c_{min}, average values c_{avg}, and maximum values c_{max} are shown in Table 1, 2, and 3, respectively. In Table 2, "0-1" means "0 or greater than 0 and less than 1".

c_{min}	0	1	2	3	4	5+
Web Services	28	29	30	5	2	7

Table 1. The distribution of minimum values c_{min}

c_{avg}	0-1	1-2	2-3	3-4	4-5	5-10	10-20	20+
Web Services	9	28	34	9	5	9	2	5

Table 2. The distribution of average values c_{avg}

c_{max}	0	1	2	3	4	5	6-10	11-50	51+
Web Services	5	16	29	12	10	6	11	6	6

Table 3. The distribution of maximum values c_{max}

4.2 The Complexity Evaluation of the Popular Web Service

Next, The evaluation results of Amazon E-Commerce Services [18], Google [19] and eBay [20] are shown in Table 4. Amazon and eBay Web services only have methods with relatively complex interfaces, although Google Web services only have simpler methods than the other two Web services.

Web Services	Amazon [18]	Google [19]	eBay [20]
c_{min}	15	2	9
c_{avg}	50.684	4.667	75.000
c_{max}	482	10	632

Table 4. The evaluation results of the popular Web services

Note that the values of eBay Web services are the tentative results, because one of the methods uses the input data which includes a recursive data structure shown in Fig. 4. In this evaluation, children of a node which is proved to be defined recursively are not counted.

4.3 Relationship of the Number of Methods to the Average of Complexities

We also focus on the relationship of the number of methods to the average of complexities to confirm whether Web services can provide many methods which are relatively easy for end users to work with. The result is shown in Fig. 5

4.4 The Comparison between the Similar Web Services

There are more than one Web service which provide the similar service to users. Table 5 shows the evaluation results of the Web services which provide stock prices. Notes that the QuoteOfTheDay (qotd) Web service simply has one method which can be invoked without an input data.

5 Discussion

As shown in Section 4, it can be considered that most of the Web services have at least one method whose input data can easily be set by end users, because

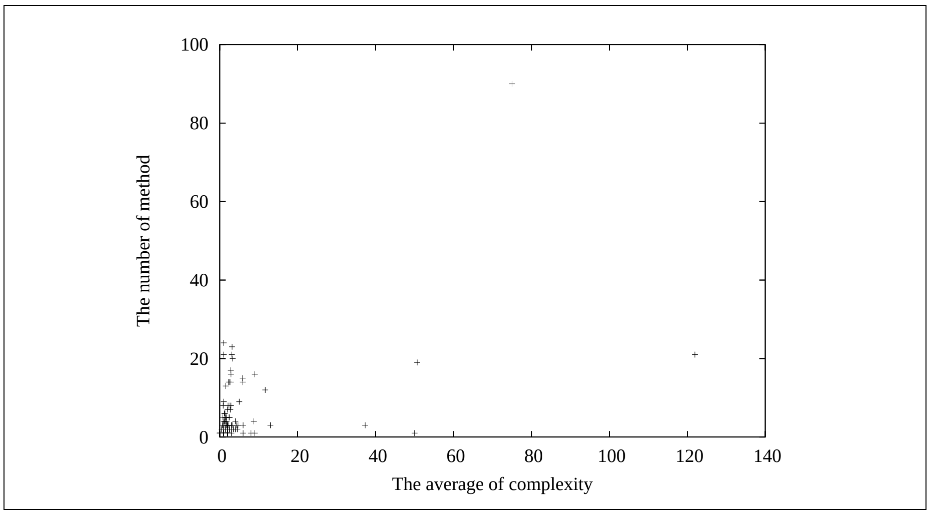

Fig. 5. The relationship of the number of methods to the average of complexities

Web Services	EODData	PowerQuote	XigniteFutures	RealTimeMarketData
c_{min}	1	0	0	1
c_{avg}	2.429	1.500	2.824	2.875
c_{max}	4	2	8	4

Web Services	qotd	RandomQuote	XigniteQuotes
c_{min}	0	1	0
c_{avg}	0.000	1.000	3.300
c_{max}	0	1	25

Table 5. The evaluation results of the Web services which provide stock prices

more then 90% of Web services which we evaluated have the methods whose complexities are less than 5.

The relationship of the number of methods to the average of complexity can be overviewed by Fig. 5. It is revealed from Fig. 5 that end users can work with relatively large web services through simple methods, because the numbers of methods in web services vary up to 20, although most of the web services have less methods and less complexity.

The results in Table 4 indicates that the evaluated complexity values of the Google Web service are much lower than the other two Web services. in other words, the Google Web service is more useful than Amazon and eBay Web services. In fact, another Web services which use some methods defined in Google Web service have been registered on the XMethod.

Table 5 reveals that the similar Web services can be easily compared from the end users' point of view with the proposed algorithm. In this example, the end users should use QuoteOfTheDay (qotd) service first. They can use PowerQuote or RandomQuote after they have tried qotd.

We proposed two rules to evaluate a data node which is defined recursively in Section 3.2. In the evaluation of eBay web service, children of a data node which is proved to be recursive are not counted, because a user could find immediately that a node is defined recursively when a node which has the same name as the original node appears. However, if one of grand children (or descendants) of a node has the same type as the node itself, it may be difficult for a user to know their recursiveness, so it is better to regard the complexity of this node as infinity. Thus, some additional rules may be needed to solve this issue.

6 Conclusion and future work

In this paper, we propose a metric for the complexity evaluation algorithm of Web services for end users by evaluating the data structures of the input arguments for each method. The proposed evaluation algorithm facilitates intuitive complexity evaluation by end users, because the evaluation results are represented by scalar values (scores).

The evaluation results revealed that most of the Web services still have simple interfaces. However, public Web services which have more complex data structures than Amazon has already appeared in the Internet. In addition, the Web services that have almost the same functionalities has also appeared. Thus, it is important to provide this type of evaluation to people who understand their businesses deeply, because it is better for them to know which method in a Web service should be used to integrate them into their own services.

As a result, the proposed evaluation algorithm will also encourage end users, as well as skilled people, to adopt SOA (Service Oriented Architecture) by combining the Web services with simple interfaces in the distributed environment.

In this paper, we only considered the data structures used for input. End users may be interested in Web services which send lots of relevant information to the user as a result of the input, so it may be worth working with the relationship between input, output, and end users. We are currently working with ad-hoc Web service orchestration for end users [21]. This will be helpful for users who has deep understandings their businesses, but does not have programming skills.

References

1. Service-Oriented Architecture, `http://www.ibm.com/software/info/openenvironment/soa`
2. Business Process Execution Language for Web Services version 1.1, `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`
3. OASIS Web Services Business Process Execution Language (WSBPEL) TC, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`
4. XMethods web site, `http://www.xmethods.net/`
5. W3C, Web Services Description Language (WSDL) 1.1, W3C Note, 2001.
6. W3C, SOAP - Simple Object Access Protocol, `http://www.w3.org/TR/SOAP`
7. W3C, XML Schema, `http://www.w3.org/XML/Schema`
8. Mindreef: Comprehensive Web services diagnostics and testing, `http://www.mindreef.com/`
9. Wang, Y. and Stroulia, E., Semantic Structure Matching for Assessing Web-Service Similarity, *ICSOC 2003, LNCS 2910*, pp 194-207, 2003, Springer.
10. Ivory, M. Y. and Hearst, M. A., The State of the Art in Automating Usability Evaluation of User Interfaces, *ACM Computing Surveys*, 33(4):470-516, December 2001.
11. Myers, B., Hudson, S. E. and Pausch, B., Past, Present, and Future of User Interface Software Tools, *ACM Transactions on Computer-Human Interfaces*, 7(1):3-28, March 2000.
12. Card, S. K., Morran, T. P. and Newell A., *The Psychology of Human-Computer Interaction*, Lawrence Elrlbaum Associates, 1983.
13. John, B. E. and Kieras, D. E., The GOMS family of user interface analysis techniques: Comparison and contrast, *ACM Transactions on Computer-Human Interaction*, 3(4):320-351, 1996.
14. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A. G. and Mehandiev, N. META-DESIGN: MANIFESTO FOR END-USER DEVELOPMENT, *Commnunication of ACM* 47(9):33-37, September 2004.
15. Eclipse Projects, SWT: The Standard Widget Toolkit, `http://www.eclipse.org/swt/`
16. The GIMP Toolkit, `http://www.gtk.org/`
17. Diestel, R., *Graph Theory*, Springer-Verlag New York, 1997.
18. Amazon Web Services, `http://www.amazon.com/gp/browse.html/002-0381178-1342459?%5Fencoding=UTF8&node=3435361`
19. Google Web APIs, `http://www.google.com/apis/index.html`
20. eBay Developers Program, `http://developer.ebay.com/soap/`
21. IBM alphaWorks, Ad Hoc Development and Integration Tool for End Users `http://www.alphaworks.ibm.com/tech/adieu/`, IBM, 2005.
22. OASIS, uddi.org, `http://www.uddi.org/`.
23. The Web Service Club, `http://objectclub.esm.co.jp/webservice/home.html` (contents are described in Japanese)