A Form-based Approach for Application Development By Web Service Integration

Takeshi Chusho, Ryousuke Yuasa and Shinpei Nishida
Department of Computer Science, Meiji University
Kawasaki, 214-8571, Japan
chusho@cs.meiji.ac.jp

Katsuya Fujiwara
Department of Computer Science and engineering, Akita University
Akita, 010-8502, Japan
fujiwara@ie.akita-u.ac.jp

ABSTRACT

This paper describes enduser-initiative application development by Web service integration with interfaces based on the simple concept that “one service = one form.” The form transformation from input forms into output forms implies service integration. The Web page wrapping method by the HTML-to-XML transformation for communicating with conventional Web applications and the XML merging method for integrating XML-base Web services, are proposed. In these methods, application-specific processes are described in XSLT stylesheets with visual tools, and application-independent processes are generated automatically.

KEYWORDS

Web service integration, object-oriented technology, XSLT stylesheet, end-user computing

1. INTRODUCTION

The number of end-users using the Internet inside and outside of the office has been increasing. Enduser-initiative development of applications has become important for automation of their own tasks. In particular, applications for Web services should be supported by domain experts themselves sinceWeb services must be modified frequently based on the domain experts’ ideas.

As the solution based on CBSE(Component-Based Software Engineering) [1, 3], this paper describes a form-based approach for end-user computing. For the typical distributed information system, we direct our attention to the application system for service counters in banks, city offices, travel agents, mail-order companies, etc. Some of the services provided at these counters have already been implemented in current computer networks including the Internet for practical use such as online shopping. However, the work to be automated may be limited to particular ones such as electronic commerce related to B-to-B and B-to-C, which make a profit over the development cost.

In the business world, recently, the external specifications of application software are considered as services. The various infrastructure technologies such as SOAP [10], UDDI [9] and WSDL [11] are used for rapid Web service provision [6]. For practical use in e-marketplaces, some problems on security, trust, quality etc. must be solved. Therefore, such infrastructures seem to pervade from local or private areas such as private UDDI to global areas such as e-marketplaces.

It becomes necessary, then, for end-users to develop and maintain applications sinceWeb services change constantly. Our approach will be applied primarily into local areas such as a group of trusted organizations also at an experimental stage of the enduser-initiative development. Web service integration [7, 8] must be the solution for these problems because end-users may consider their operations as a level of service, not as a level of software. That is, the service counter is considered as a metaphor of the interface between service providers and their clients for Web services, and is designed based on the simple concept that “one service =

one form." It provides form-based interfaces and then the form transformation from input forms into output forms implies service integration.

This paper presents the enduser-initiative approach in Section 2, technical issues for automatic integration of these services in Section 3 and our solutions in Section 4 and Section 5.

2. ENDUSER-INITIATIVE APPROACH

2.1 Basic concepts for Web services

For Web applications supporting Web services, the following two features are considered to be essential:
(1) Rapid development and continuous variation.
(2) End-user initiative.

Business based on Internet technologies, is rapidly changed. For this reason, the application development period from defining a new business model through releasing new services, must be short. If not, the business chance will be lost. Furthermore, after the release, the application should be maintained continuously as the business world changes. Conventional ways for application development and maintenance by system engineers, are not suitable because of no timeliness.

Our approach [2] to how to make Web applications supporting Web services is shown in Figure 1. The left zone and the central zone imply the abstract levels and the concrete levels respectively. The right zone implies technologies. The business model at the business level is proposed by end-users of domain experts. Then, at the service level, the domain model is constructed and the required services are specified. That is, the requirement specifications of the application for the business model are defined. At the software level, the domain model is implemented by combination of components.

In this approach, there are two technological gaps, that is, the granularity gap between components and the domain model, and the semantic gap between the domain model and end-users. The granularity gap is bridged by business objects, patterns [5] and application frameworks [4] based on CBSE(Component-Based Software Engineering). On the other hand, the semantic gap is bridged by form-base agent technologies.

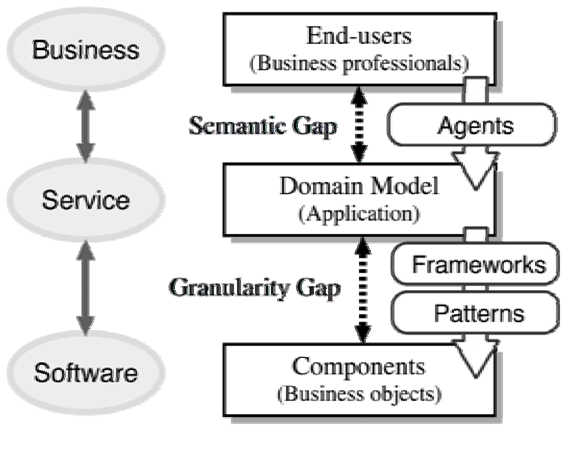
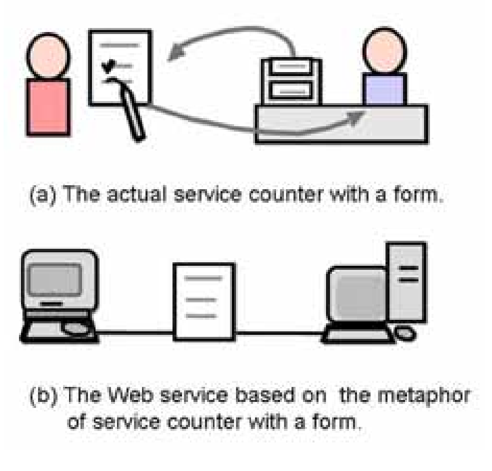


Figure 1. Technologies for bridging two gaps.


(a) The actual service counter with a form.
(b) The Web service based on the metaphor of service counter with a form.
Figure 2. Service counter as a metaphor for Web service.

2.2 Metaphors for Web services

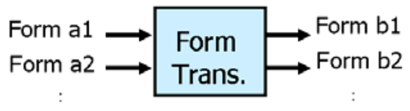
As a typical distributed information system, we direct our attention to application systems for service counter. Such service counter is not limited to the actual service counter in the real world. For example, in a supply chain management system (SCM), exchanges of data among related applications can be considered as the virtual service counter.

Of course many kinds of service counters have already been put to practical use in the Internet and intranets. However, these systems must have been developed by IT professionals, not by end-users and are expensive. Furthermore, although the domain experts require frequent modification of specifications in these systems for service upgrade, it is difficult to modify software timely because the domain experts do not maintain the systems by themselves and need to ask the IT professionals instead. Our goal is development and maintenance of form-based applications by end-users.

Generally, service counter is considered as service requests between clients and service providers as shown in Figure 2. Forms to be filled in are considered as the interface between them. That is, the following concept is essential for our approach:

”One service = One form”

Then the Web service integration of some individual Web services is considered as transformation from some input forms into some output forms as shown in Figure 3. Our goal is that domain experts make an application by the form transformation.



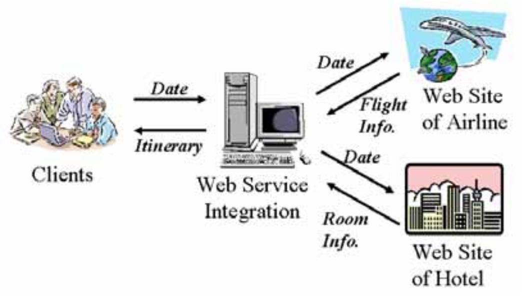


Figure 3. Form transformation for Web service integration. Figure 4. An example of Web service integration.

3. TECHNICAL ISSUES

Let us suppose that we plan an itinerary by using reservation services of a hotel room and a flight via Internet as shown in Figure 4. We usually visit the Web site of a hotel and the Web site of an airline company separately, and then book a hotel room and a flight seat individually.

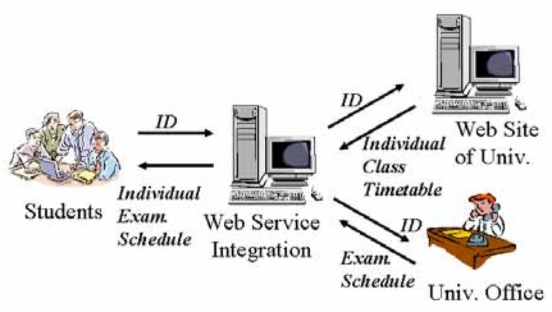
There are two technical issues for integration of these services. One is how to communicate with conventional Web applications which support HTML documents as user interfaces. Two solutions are considered for this problem. One is that a front-end part of the Web application is newly developed in accordance with the XML-base Web service specifications. This solution is, however, not realistic because many companies will not do so until the XML-base Web service pervades the Internet world. The practical solution is the Web page wrapping that the HTML documents are transformed automatically into the XML documents. The other issue is how to merge two inputs in XML. That is, the reasonable combination of an available hotel room and an available flight seat should be found from the hotel room information in XML and the flight seat information in XML.

4. PROTOTYPING

4.1 Target application

For applying our solutions to the practical Web service integration, we select the application which generates an individual examination schedule for each student. In our university, actually, the examination schedule is notified on bulletin boards. On the other hand, the university supports the individual portal site for each student. The student visits the portal site and can display the individual timetable for classes.

Therefore, we suppose that the XML-base Web service for the examination schedule will be realized in the near future. In our experiment, an actual examination schedule is transformed into an XML document manually. As for the individual timetable for classes, an actual HTML document is extracted from the individual portal site for each student. The target application generates an individual examination schedule for each student from the individual timetable for classes and the examination schedule as shown in Figure 5.



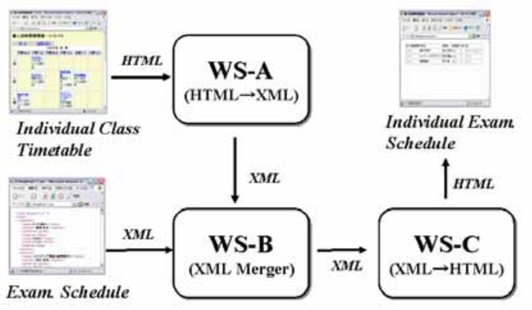


Figure 5. Web service for an individual exam. schedule. Figure 6. System configuration for Web service integration.

4.2 System architecture

The system architecture is shown in Figure 6. The input of the subsystem, WS-A, is the individual timetable for classes. This is an actual HTML document which is extracted from the individual portal site for each student. This document includes information about classes for each student, that is, class names, instructor names and class room numbers. The WS-A transforms this HTML document into the XML document by using the wrapping technology.

The inputs of the subsystem, WS-B, are two XML documents. One is the individual timetable for classes for each student in XML. The other is the examination schedule in XML, which includes information about the examinations in classes, that is, class names, dates and periods, and room numbers. The WS-B merges these two XML documents into the individual examination schedule in XML for each student. The input of the subsystem, WS-C, is the individual examination schedule in XML. The WS-C transforms this XML document into the HTML document which can be displayed in the Web browser for each student.

5. SCRIPTING BUSINESS LOGIC

5.1 End-user support

The system administrator of this application is not an IT professional but a clerk of the university office. Such an end-user does not have the programming ability, but need to modify the system when the inputs change. For the solution of this problem, basically, the procedure of this application is described in a script language. Furthermore, a visual tool supports the end-user.

5.2 HTML-to-XML transformation

The WS-A subsystem transforms the HTML document into the XML document. The input is the individual timetable for classes in HTML. A part of this document is shown as follows:

```
<TR>
  <TH align=middle bgColor=#fff0f5 height=60>2<BR>period</TH>
  <TD bgColor=#f0fff0><A href="http://oh-o.meiji.ac.jp/zy_page.php?lcode=15573400&amp;
tid=890085&amp;ttype=1&amp;tconf=1" target=_blank>
  <B>Software Engineering</B></A><BR><FONT color=#008000>Tak CHU<BR>Room 0311<BR></FONT>
<!-- JIWARNO = 7508--><!-- CLASSCD = 15573400-->
</TD>
```

The following XML document is generated from the information about class names and instructor names which are extracted from this HTML document.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<personaltimetable>
  <subject>
    <name>Software Engineering</name>
    <instructor>Tak CHU</instructor>
  </subject>
</personaltimetable>
```

This procedure is composed of two steps for use of XSLT(XSL Transformations) [12]. At the first step, the HTML document is transformed into the XHTML document because XSLT cannot accept HTML documents but can accept XHTML documents. At the next step, the XSLT stylesheet is used to transform this XHTML document into the XML document.

In this procedure, only the XSLT stylesheet is dependent on the individual application, and must be described by the end-users. The other parts are automated. It may be difficult, however, for the end-users to describe the XSLT stylesheet yet although this scripting is rather easer than programming. This is because the end-user must understand the concepts of Xpath and the template. We developed a visual tool which generates the XSLT stylesheet. This tool is used in following four steps:

(1) Specify the input file name of the HTML document in the text input area.
(2) Define the structure of the XML output document under the guidance with examples.
(3) Select data to be extracted from the HTML document.
(4) Save the XSLT stylesheet generated by entering the output file name.

At the third step, a class timetable is displayed. The checkbox is added at the front of each item of a class name, an instructor name or a room number in this timetable although the original timetable does not have such checkboxes. The user selects necessary data by checking the checkboxes.

5.3 XML documents merging

The WS-B subsystem merges two XML documents, that is, the individual timetable for classes for each student and the examination schedule, into the individual examination schedule in XML for each student. The following XML document is a part of the examination schedule:

```
<?xml version="1.0">
<examine>
  <subject>
    <name>Software Engineering</name>
    <instructor>Tak CHU</instructor>
    <grade>3</grade>
    <class>14・15</class>
    <room>0405</room>
    <date>7/24</date>
    <time>3</time>
  </subject>
```

This subsystem generates the XML document as the output by extracting classes which are included in the both input files. This procedure is composed of the following steps, and the XSLT stylesheet is used.

(1) Assign the nodes of two input XML files into variables.
(2) Extract the element for comparison from each input file by using the variables for counting.
(3) Compare two elements.
(4) Output the element in the specified format if these two elements are the same.

XSLT is a language for transforming XML documents into other XML documents. Furthermore, XSLT makes use of the expression language defined by XPath for selecting elements to be processed, for conditional processing and for generating text. In the XSLT stylesheet, a document function and a position function are used. The document function allows access to XML documents other than the main source document. The input files, xml-1.xml and xml-2.xml, are assigned to the variables, “first” and “second” respectively.

The position function returns the position of one of child elements which have the same parent node. The current position of one of subject elements which are included in the xml-1.xml file, is assigned to the variables, “firstposition.” The one included in the xml-2.xml file, is assigned to the variables, “secondposition.” Then the class names and the instructor names of two input files, are compared respectively.

In this procedure, the XSLT stylesheet is dependent on the individual application, and must be described by the end-users. The other parts are automated. It may be difficult, however, for the end-users to describe the XSLT stylesheet yet because they must understand the concepts of iterative processes with variables.

We developed a visual tool which generates the XSLT stylesheet. This tool is used in the following steps:
(1) Specify the input file names.
(2) Select elements to be compared.
(3) Define conditions for the comparison.
(4) Define the structure of the XML document to be output.
(5) Select data to be embedded in the XML document.
(6) Save the XSLT stylesheet generated.

At the first step, the file names are input in the text input areas. At the second step, two input XML documents are transformed into HTML documents in which the checkbox is added at the front of each node, and are displayed. The user selects the parent node of the element to be compared. Next, the user selects the elements to be compared. That is, in this application, first the subject is selected, and then the class name and the instructor name are selected. At the third step, the selected elements are displayed as follows:

```
 F0 /examine[1]/subject[n]/name[1]
 F1 /examine[1]/subject[n]/instructor[1]
 S0 /personaltimetable[1]/subject[n]/name[1]
 S1 /personaltimetable[1]/subject[n]/instructor[1]
```

F0, F1, S0 and S1 are symbols for the corresponding XPaths. The user can define conditions for the comparison by using these symbols as follows:

```
F0 = S0
F1 = S1
```

For example, the first condition is transformed into the following expression with XPath:

```
$first/examine[1]/subject[$firstposition]/name[1]= $second/personaltimetable[1]/
subject[$secondposition]/name[1]
```

The remainder of the steps are similar to the second, third and fourth steps in the WS-A subsystem.

5.4 XML-to-HTML transformation

The WS-C subsystem transforms the XML document into the HTML document. The XSLT stylesheet for this application is generated by using one of conventional tools.

6. CONCLUSIONS

The Web page wrapping method by the HTML-to-XML transformation was developed for communicating with conventional Web applications. The XML merging method was developed for merging XML-base Web services. In these methods, application-specific processes are described in XSLT stylesheets with visual tools, and application-independent processes are generated automatically.

REFERENCES

[1] Brown(Ed.), A. W., 1996. *Component-based software engineering*. IEEE CS Press, USA.
[2] Chusho,T. et al, 2002. A Form-based Approach for Web Services by Enduser-Initiative Application Development. *Proc. Of SAINT2002 Workshop (Web Service Engineering)*, IEEE Computer Society, USA, pp.196-203.
[3] Egyed, A. et al. 2005. Integrating COTS into the Development Process, *IEEE Software*, Vol. 22, No. 4, pp. 16-18.
[4] Fayad, M. and Schmidt, D. C. (Ed.), 1997. Object-Oriented Application Frameworks. *Commun. ACM*, Vol. 39, No. 10, pp. 32-87.
[5] Gamma, E. et al, 1995. *Design Patterns*. Addison-Wesley, USA.
[6] Gold, N. et al,2004. Understanding Service-Oriented Software. *IEEE Software*, Vol. 21, No. 2, pp.71-77.
[7] Peltz, C., 2003. Web Services Orchestration and Choreography. *IEEE Computer*, Vol. 36, No. 10, pp.46-52.
[8] Tsai, T. et al, 2003. Ontology-Mediated Integration of Intranet Web Service. *IEEE Computer*, Vol. 36, No. 10, pp.63-71.
[9] UDDI. White Papers. http://www.uddi.org/.
[10] W3C. Latest SOAP versions. http://www.w3.org/TR/soap/.
[11] W3C. Web Services Activity. http://www.w3.org/2002/ws/.
[12] W3C. The Extensible Stylesheet Language Family (XSL). http://www.w3.org/Style/XSL/.