# Web Service Integration Based on Abstract Forms in XML for End-user Initiative Development

Takeshi Chusho, Ryousuke Yuasa, Shinpei Nishida and Katsuya Fujiwara *

*Abstract*— **The number of end-users using the Internet has been increasing. End-user initiative development of applications has become important for automation of end-users' own tasks. In particular, Web applications should be supported by domain experts themselves since Web applications must be modified frequently based on domain experts' ideas. This paper describes end-user initiative application development by Web service integration. The abstract forms are considered as interfaces of services based on the simple concept that "one service = one form." Therefore, Web service integration can be defined as form transformation from input forms into output forms. There are two problems for development of Web applications by Web service integration. One is how to communicate with conventional Web applications. The other is how to merge XML-based Web services. This paper proposes the Web page wrapping method by the HTML-to-XML transformation and the XML merging methods. In these methods, application-specific processes are described in XSLT stylesheets with visual tools by end-users, and application-independent processes are generated automatically. In particular, the multistage XML merger is introduced for treating complicated business logic.**

***Keywords: Web service, form transformation, 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. As a result, the number of Web applications which end-users use has also been increasing. Most of these applications are developed by IT professionals. Thus, the work to be automated is limited to particular tasks such as electronic commerce relating to B-to-B and B-to-C, for example, which calculates profit over the development cost. Furthermore, it is difficult to develop and maintain applications quickly. Primarily, Web applications should be supported by domain experts themselves since Web applications must be modified frequently based on the domain experts' ideas.

Therefore, end-user initiative development of applications has become important for the automation of end-users' own tasks. In the near future, the information society will require such new technologies empowering domain experts and enabling them to automate their own work independently without extra training or the help of others.

In the business world, recently, the external specifications of application software are considered as services. Some infrastructure technologies such as SOAP [15], UDDI [12] and WSDL [16] are used for rapid Web service provision [9] [11]. For example, Amazon [1] and Yahoo [18] provide Web service interfaces.

As for UDDI, problems concerning security, trust, quality etc. must be resolved for practical use in e-marketplaces. Therefore, such infrastructures may pervade from local or private areas such as private UDDI to global areas such as e-marketplaces.

As a solution based on CBSE(Component-Based Software Engineering) [2] and SOA(Service-Oriented Architecture), this paper describes a form-based approach to Web service integration [14] for end-user computing, because end-users consider their applications as a level of service, not as a level of software. That is, the service counter is considered as a metaphor to describe the interface between service providers and their clients for Web services, and is designed based on the simple concept that "one service = one form." This concept provides form-based interfaces enabling Web service integration to be defined as the form transformation from input forms into output forms. In addition, our approach will be applied, primarily in local areas such as within a group of trusted organizations, at an experimental stage of the end-user initiative development.

There are some other works related to the end-user initiative development. In the database field, the example-based database query languages [13] such as QBE(Query-By-Example) were studied. User-friendly inquiry lan-

*Manuscript received December 28, 2006. T. Chusho, R. Yuasa and S. Nishida are with the Department of Computer Science, Meiji university, kawasaki, Japan (e-mail:chusho@cs.meiji.ac.jp). K. Fujiwara is with the Department of Computer Science and Engineering, Akita University, Akita, Japan (e-mail:fujiwara@ie.akita-u.ac.jp)

guages were proposed in comparison with SQL. In the programming field, the technologies for programming by example [10] were studied. The non-programming styles for various users including children and for various domains including games, were proposed. Our research target is business experts and business domains.

This paper presents the end-user initiative approach in Section 2, technical issues for automatic integration of these services in Section 3 and our solutions in Section 4, Section 5 and Section 6.

## 2 End-user 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 maintenance.

(2) End-user initiative.

Business based on Internet technology is rapidly changing. For this reason, the application development period from defining a new business model through releasing new services, must be kept short. If it is 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 the lack of timeliness.

Our approach to how to make Web applications supporting Web services is shown in Figure 1 [5]. The three layers of the left zone and the central zone imply the abstract level and the concrete level, 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 using components to be combined.

In this approach, there are two technological gaps, these being, 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 [8] and application frameworks [7] based on CBSE(Component-Based Software Engineering). Conversely, the semantic gap is bridged by form-based technologies.

As for the granularity gap, our previous studies verified the effectiveness of application framework technologies by development of service counter frameworks.

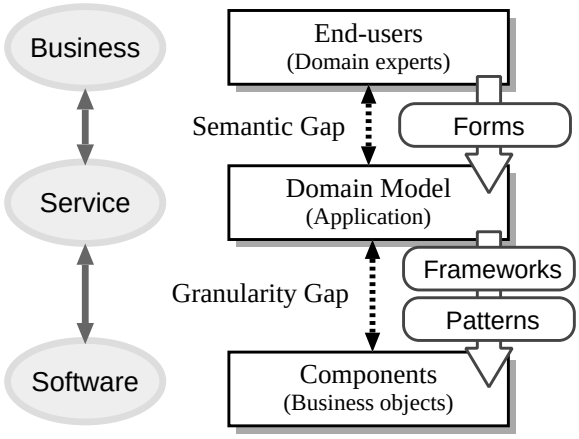


Figure 1: Technologies for bridging gaps between end-users and components.

Concerning the semantic gap, this gap was bridged by agent technologies detailed in our previous studies [4]. That is, an intelligent form with an agent which learns business logic, was introduced. However, in our recent studies on end-user initiative development, it is considered that a simple form is better for end-users.

### 2.2 Metaphors for Web Services

We direct our attention to an application system for a service counter. Such a 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 data exchanges at the virtual service counter.

Of course, many kinds of service counters have already been put to practical use on the Internet and in 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 upgrades, it is difficult to modify software quickly because the domain experts do not maintain the systems themselves and need to rely on the IT professionals instead. Therefore, our goal is the development and maintenance of form-based applications by end-users.

Generally, the service counter receives service requests from clients to 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."

The integration of some individual Web services is considered as transformation from some input forms into some output forms as shown in Figure 3 with the goal being that domain experts make an application by form trans-

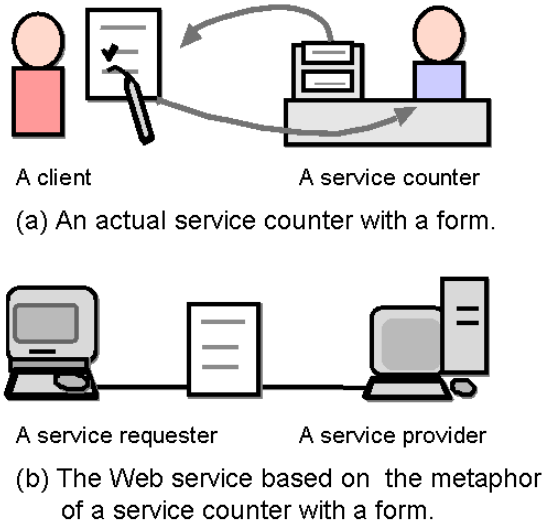


Figure 2: A service counter as a metaphor for Web service.

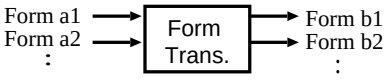


Figure 3: The form transformation for Web service integration.

formation.

In this case, most of these forms are not visual forms, but are abstract forms in XML. Since end-users consider such Web service integration as work flow with visual forms which they are familiar with, IT skills are not required of end-users.

For example, in making reservations for travel and accommodations, when one service provider receives the input form for reservations for both a hotel room and a flight, the provider transforms the input into the first output form for reservation of the hotel room and the second output form for reservation of the flight. Then, the provider sends the first output form to the service provider providing hotel room reservations and the second output form to the service provider providing flight reservations, respectively. When the provider receives both replies to these output forms, that is, the first input form on available hotel rooms and the second input form on available seats on a flight, the provider transforms these two input forms into the output form which is a combination of available hotel rooms and available seats on a flight. Finally, this output form is returned as the reply to the initial request. In Web service integration, forms which are used as the interface between service providers and those who request service, are abstract forms in XML.

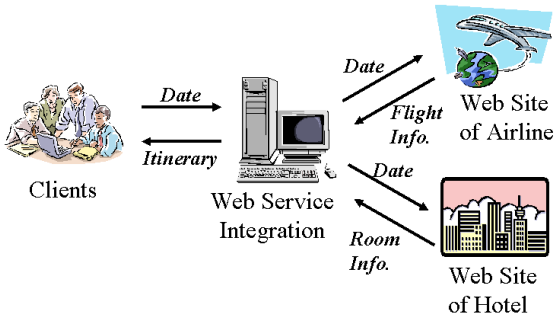


Figure 4: An example of Web service integration.

### 2.3 A Shift from Message Flow to Form Flow

Our previous model-driven solution for end-user initiative application development was based on an object-oriented model that one task in a domain model of cooperative work corresponds to one object in a computation model. This solution requires, out of necessity, fixed architecture and ready-made components such as business objects for component-based development. The application development environment, M-base, was developed for supporting this solution [3] . At the first stage in the process, the system behavior is expressed as a message-driven model by using a visual modeling tool while focusing on message flow and components.

However, since end-users are not familiar with object-oriented technologies, practical development processes must be provided based on metaphors of an office. That is, cooperative work at an office is expressed by using a form flow model. Thus, our approach to end-user initiative development has shifted from a message flow base to a form flow base.

## 3 Technical Issues

Let us suppose that we are planning an itinerary by using reservation services for both hotel accommodations and flight booking via the 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 independent of each other.

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 could be 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 in the Internet world. The other, and more practical solution, is Web page wrapping in which the HTML documents

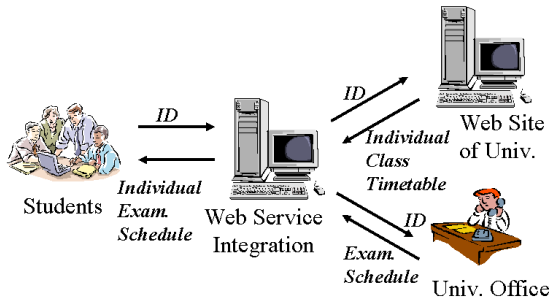


Figure 5: Web service integration for an individual exam. schedule.

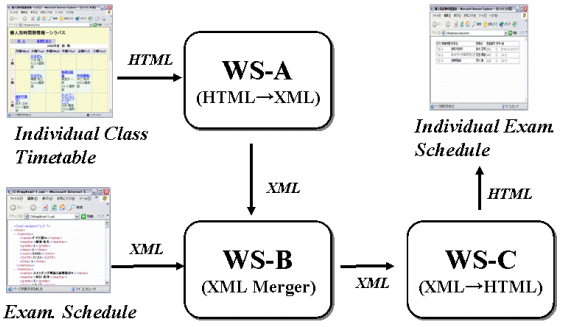


Figure 6: System configuration for Web service integration.

are transformed automatically into XML documents.

The other issue is how to merge two outputs in XML. That being, the reasonable combination of information pertaining to an available hotel room and an available seat on a flight should be derived from both the hotel room information in XML and the flight information in XML.

## 4 The basic XML merger

### 4.1 Target Application

For applying our solutions to practical Web service integration, we select an application which generates individual examination schedules for each student [6]. In our university, actually, the examination schedule is notified on bulletin boards. Conversely, the university supports the individual portal sites for each student. The student visits the portal site which displays the individual timetable for classes.

Therefore, we foresee 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.

### 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 subjects for each student, that is, subject names, instructor names and class numbers. The WS-A transforms this HTML document into an XML document by using the wrapping technology mentioned previously.

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 subject names, dates and periods, and room numbers. The WS-B merges these two XML documents into the individual examination schedule in XML format 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 an HTML document which can be displayed on 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 in the university office. Such an end-user does not have the ability to perform programming, but needs only 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 an XML document. The input is the individual timetable for classes in HTML. The following XML document is generated from the information about subject 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>Txx Cxx</instructor>
 </subject>
</personaltimetable>
```

This procedure is composed of two steps for use of XSLT(XSL Transformations) [17]. 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, XSLT is used to transform this XHTML document into the XML document by using the following XSLT stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
 <xsl:output encoding="UTF-8" method="xml"/>
 <xsl:template match="/">
  <personaltimetable>
   <xsl:apply-templates
    select="/html[1]/body[1]/table[3]/
tbody[1]/tr/td/a[1]/b[1]"/>
  </personaltimetable>
 </xsl:template>
 <xsl:template match="/html[1]/body[1]/
table[3]/tbody[1]/tr/td/a[1]/b[1]">
  <subject>
   <name>
    <xsl:value-of select="/text()"/>
   </name>
   <instructor>
    <xsl:value-of select="../../font[1]/
text()"/>
   </instructor>
  </subject>
 </xsl:template>
</xsl:stylesheet>
```

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 comparatively 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 four steps. At the first step, the file name of the HTML document is input into the text input area which is displayed by the visual tool. At the second step, the structure of the XML document to be output is defined in the text input area under the guidance with examples which are displayed by the visual tool. Alternatively, the user can include the pre-defined file. At the third step, a class timetable is displayed. A checkbox is located in front of items such as subject name, instructor name or room number in this timetable as shown in Figure 7 although the original timetable does


Figure 7: A part of the timetable embedded with checkboxes (in Japanese).

not have such checkboxes. The user selects data to be extracted from the HTML document by checking the checkboxes. At the last step, XPath is identified for selecting elements to be processed, and the XSLT stylesheet is generated by entering the output file name.

### 5.3 XML Documents Merging

The WS-B subsystem merges two XML documents, 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>Txx Cxx</instructor>
  <grade>3</grade>
  <class>14・15</class>
  <room>0405</room>
  <date>7/24</date>
  <time>3</time>
 </subject>
 <subject>
   ....
   ....
 </subject>
</examine>
```

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 four steps:

1. Assign the nodes of two input XML files into variables.
2. Extract the element to be compared 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.

For these processes, the XSLT stylesheet is used. 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 for processing, 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, as follows:

```
 <xsl:variable name="first" select=
"document('xml-1.xml')" />
 <xsl:variable name="second" select=
"document('xml-2.xml')" />
```

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

```
<xsl:for-each select=
"$first/examine[1]/subject">
 <xsl:variable name="firstposition" select=
"position()" />
 <xsl:for-each select=
"$second/personaltimetable[1]/subject">
  <xsl:variable name="secondposition" select=
"position()" />
   <xsl:if test=
"$first/examine[1]/subject[$firstposition]/
name[1]= $second/personaltimetable[1]/
subject[$secondposition]/name[1]">
   <xsl:if test=
"$first/examine[1]/subject[$firstposition]/
instructor[1]= $second/personaltimetable[1]/
subject[$secondposition]/instructor[1]">
   ....
```

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 although this scripting is rather easy compared to programming. This is because the end-user must understand the concepts of iterative processes with variables.

We developed a visual tool which generates the XSLT stylesheet as shown in Figure 8: This tool is used in six steps: In the first step, file names are input into the text input areas. In the second step, two input XML documents are transformed into HTML documents in which the checkboxes are located 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 subject name and the instructor name are selected. In 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
```

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

### 5.4 Generality of the Merging Method

This process can be applied to other general domains. For example, let us consider that a client wants to acquire a concert ticket from C company, a flight ticket from A airline company and to reserve a room at H hotel during a specified period, by using the system with Web service integration of these three companies.

In this case, there are the following four input abstract forms(AFs): AFin1 with the specified period, AFin2 with information about the concert ticket reservation from C company, AFin3 with information about the flight reservation from the A airline company, AFin4 with information about the room reservation from H hotel.

The basic merger is executed repeatedly for multiple inputs. That is, the merging processes as shown in Figure 9, are required for the client request. At the first step, a list of dates with the available concert ticket is extracted by merging the dates of AFin1 and the dates with available tickets of AFin2, and then is output as AF-1. At the second step, a list of dates with both the available concert ticket and the available flight ticket, is extracted by merging the dates of the AF-1 and the dates with available flight tickets of AFin3, and then is output as AF-2. At the last step, a list of dates in conjunction with the client's request is extracted by merging the dates of AF-2 and the dates with available rooms of AFin4, and then is output as the AFout.

Since these merging processes are the same as the process for generating an individual examination schedule, the XSLT stylesheet can be generated by using our visual tool.


Figure 8: A visual tool for generating the XSLT style sheet. (in Japanese).

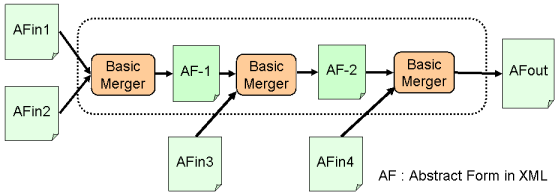


Figure 9: Repetition of the basic merger for multiple inputs.

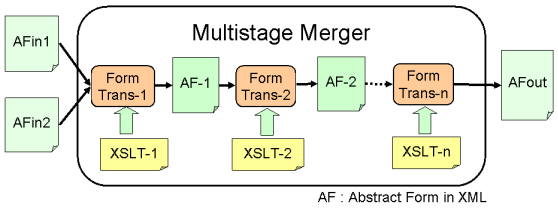


Figure 10: A configuration of the multistage merger for complicated business logic.

### 5.5 XML-to-HTML Transformation

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

## 6 Multistage XML merger

The basic merger transforms input abstract forms in XML into output abstract forms in XML with simple business logic. Next, this merger is extended for dealing complicated business logic. As a sample application, the advisory system for graduation is chosen. This system advises a student to take specific subjects necessary for graduation from inputs of both his/her school report and a manual on graduation requirements. The school report shows a list of subjects and its units which the student has already completed. The manual shows conditions for graduation, which are considered as complicated business rules. That is, subjects are categorized into several groups which compose a hierarchy, and each category has individual conditions.

For dealing with such complicated business logic, the multistage merger is introduced as shown in Figure 10. The previous basic merger is considered as a special case in this multistage merger that the output of the first stage, AF-1, is the final output, AFout. In the multistage merger, generally, the intermediate output, AF-k, is transformed into the next intermediate output, AF-(k+1).

In the multistage merger, four kinds of templates for the root element, the subject category, compulsory subjects and semi-compulsory subjects are introduced. For example, the template for compulsory subjects is used at the second stage. Compulsory subjects have four kinds of attributes. The attribute of “requiredunit” is the required number of units. The attribute of “totalunits” is the number of units which the student has already completed. The attribute of “comparison” is the condition

of comparison of the number of completed units with the required number of units. The attribute of "full" is the state of satisfaction of conditions.

In the template for compulsory subjects, the "requiredunits" and "comparison" in AF-1 are output into AF-2 without modification. The "totalunits" is calculated by using Xpath with the sum function and the current function as follows:

```
   <xsl:attribute name="totalunits">
    <xsl:value-of select="sum(current()/
subject[@done = &quot;true&quot;]/@units)"/>
   </xsl:attribute>
```

The "full" is calculated by using Xpath with the not function as follows:

```
   <xsl:attribute name="full">
    <xsl:value-of select="not(current()/
subject[@done = &quot;false&quot;])"/>
   </xsl:attribute>
```

Finally, all data on elements concerning compulsory subjects in AF-1 are output into AF-2.

In this application, the seven steps were necessary for the confirmation of conditions required for graduation.

## 7 Conclusions

The form-based approach for Web services integration by end-user initiative application development was proposed. Our experiments of prototyping were herein described. For communication with conventional Web applications, the Web page wrapping tool for HTML-to-XML transformation was developed. For merging XML-base Web services, the basic merger for simple business logic and the multistage merger for complicated logic were developed. In these methods, application-specific processes are described in XSLT stylesheets with visual tools, and application-independent processes are generated automatically.

## References


[1] Amazon, "Web Services," *http://www.amazon.co.jp/*, 2006.

[2] Brown, A. W., (Ed.), *Component-based software engineering*, IEEE CS Press, 1996.

[3] Chusho, T., Ishigure, H., Konda, N. and Iwata, T., "Component-Based Application Development on Architecture of a Model, UI and Components," *APSEC2000*, IEEE Computer Society, pp.349-353, 2000.

[4] Chusho, T. and Fujiwara, K., "FACL : A Form-based Agent Communication Language for Enduser-Initiative Agent-Based Application Development," *COMPSAC2000*, IEEE Computer Society, pp.139-148, 2000.

[5] Chusho, T., Fujiwara, K., Ishigure, H. and Shimsda, K., "A Form-based Approach for Web Services by Enduser-Initiative Application Development," *SAINT2002 Workshop (Web Service Engineering)*, IEEE Computer Society, pp.196-203, 2002.

[6] Chusho, T., Yuasa, R., Nishida, S. and Fujiwara, "A Form-based Approach for Application Development By Web Service Integration," *Applied Computing 2006*, IADIS, pp.600-605, 2006.

[7] Fayad, M. and Schmidt, D. C. (Ed.), "Object-Oriented Application Frameworks," *Commun. ACM*, V39, N10, pp. 32-87, 1997

[8] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns*, Addison-Wesley, 1995.

[9] Gold, N., Mohan, A., Knight, C. and Munro, M., "Understanding Service-Oriented Software," *IEEE Software*, V21, N2, pp.71-77", 2004.

[10] Lieberman, H. (Ed.), "Special issue on Programming by example," *Comm. ACM*, V43, N3, pp.72-114, 2000.

[11] Malloy, B. A., Kraft, N. A., Hallstrom, J. O. and Voas, J. M., "Improving the Predictable Assembly of Service-Oriented Architectures," *IEEE Software*, V23, N2, pp. 12-15, 2006.

[12] OASIS, "Advancing Web Services Discovery Standard," *http://www.uddi.org/*, 2004.

[13] Ozsoyoglu, G. and Wang, H., "Example-Based Graphical Database Query Languages," *IEEE Computer*, V26, N5, pp.25-38, 1993.

[14] Peltz, C., "Web Services Orchestration and Choreography," *IEEE Computer*, V36, N10, pp.46-52, 2003.

[15] W3C, "Latest SOAP versions," *http://www.w3.org/TR/soap/*, 2003.

[16] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," *http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/*, 2006.

[17] W3C, "The Extensible Stylesheet Language Family (XSL)," *http://www.w3.org/Style/XSL/*, 2006.

[18] Yahoo-Japan, "Yahoo!Developer Network," *http://developer.yahoo.co.jp/*, 2006.