Modeling by Form Transformation for End-user Initiative Development

Takeshi CHUSHO and Noriyuki YAGI
Department of Computer Science, Meiji University
chusho@cs.meiji.ac.jp

Abstract

The development of Web applications should be supported by business professionals themselves since Web applications must be modified frequently based on their needs. This paper describes end-user initiative application development. The abstract forms are considered as interfaces of services based on the simple concept that "one service = one form." Web service integration can be defined as form transformation from input forms into output forms.

1. Introduction

The number of Web applications which end-users have access to has been increasing. Most of these applications are developed by IT professionals. Thus, attempting to achieve automation is limited to particular tasks which calculate profit over the development cost. Furthermore, it is difficult to develop applications quickly. Primarily, Web applications should be supported by business professionals themselves since Web applications must be modified frequently based on users' needs. Therefore, end-user initiative development has become important in the automation of end-users' fulfilling their own needs.

Our primary approach[1] is based on component-based software engineering. The front-end subsystem which main part is user interface, is developed by using application framework. The back-end subsystem which main part is work flow, is done by using a visual modeling tool for describing a domain model with a message flow diagram.

In the business world, recently, the external specifications of application software are considered as services as shown in keywords such as ASP, Web service, SOA and SaaS[2,4]. Our recent approach is for end-users to develop Web applications by service integration because end-users consider their applications as a level of service, not as a level of software. A form-based approach enables Web service integration to be defined as the form transformation from input forms into output forms.

This paper presents previous studies in Section 2, our experiences in Section 3 and the form transformation in Section 4 and Section 5.

2. Previous studies on end-user computing

The term of end-user computing (EUC) often came out in 80's. A paper summarizes trends of end-user development without an IT professional's assistance[6]. In the programming field, the technologies for programming by example (PBE)[3] were studied. The PBE implies that some operations are automated based on a user's intention inferred from examples of operations. In the database field, the example based database query languages[5] such as QBE (Query-By-Example) were studied. QBE implies that a DB query is executed by examples of concrete queries.

Our research target is for business professionals and the business world in general. Furthermore, the user's intention is clearly defined.

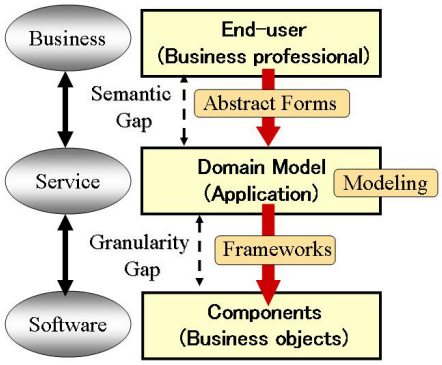

Figure 1: Technologies for end-user computing.

3. Experiences on EUC

3.1. Basic approaches

Our approach to Web application development is shown in Figure 1. The business model at the business level is proposed by those end-users who are business professionals. Then, at the service level, the domain model is constructed and the required services are specified. At the software level, the domain model is implemented by using components. In this approach, the granularity gap between components and the domain model is bridged by business objects and application frameworks. The semantic gap between the domain model and end-users is bridged by form-based technologies.

3.2. Application framework

In our UI (user-interface) driven approach, the forms are defined first and the framework is used. The business logic depending on the application is defined by the form definitions. The other business logic is embedded into the framework.

For example, the framework for booking was developed and applied to a meeting room reservation system for our department. A total of 23 forms are developed in this system for users and the administrator. Among them, 20 forms are dynamic Web pages which are defined by our visual tool. The UI transition diagram on the forms for users is shown in Figure 2. Almost all functions are directly mapped into some forms. The functional sufficiency and the usability can be evaluated and improved easily since all use cases are simulated on the forms and the UI transitions.

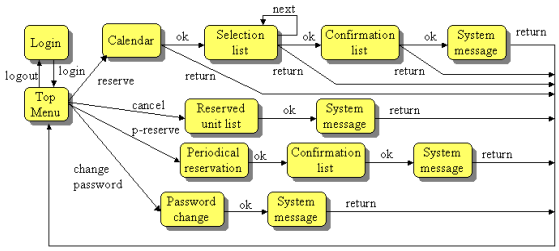

Figure 2. An example of a UI transition diagram.

However, end-users may need an IT professional's assistance for the UI to be implemented in JSP, components to be newly developed and a complicated DB management system.

3.3. Visual modeling

The next approach is for end-users to develop Web applications by using a visual modeling tool for the back-end subsystem which main part is work flow. One solution is given as a formula of "a domain model = a computation model." This formula implies that one task in a domain model of cooperative work corresponds to one object in a computation model. Therefore, it is not necessary for the end-user to convert a domain model into a computation model.

The system behavior is expressed as a message-driven model. Then, user interfaces are generated automatically. The system behavior is verified by using a simulation tool. This process was confirmed through a feasibility study. The domain model for the program chair's role of a particular academic conference was constructed as shown in Figure 3 while introducing eleven kinds of objects.

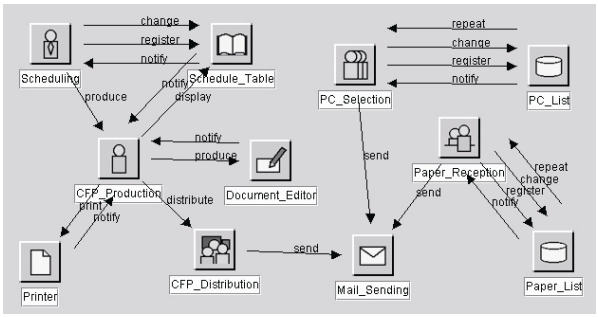

Figure 3. An example of a domain model.

However, end-users may need an IT professional's assistance for components to be newly developed and complicated DB management systems.

3.4. Web service

Next, we direct our attention to services and consider a service counter as a metaphor of a service. Such a service counter is not limited to the actual service counter in the real world. For example, in a supply chain management system, exchanges of data among related applications can be considered as data exchanges at the virtual service counter.

A form is considered as the interface for service. The concept of "One service = One form" is essential for our approach. The integration of some individual Web services is considered as transformation from some input forms into some output forms. 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.

Our previous two approaches are unified by these concepts. The frameworks for front-end subsystems are treated as a special case where a part of the abstract forms are actually visual forms. The visual modeling for back-end subsystems are treated as a special case when the message flow is used instead of the form flow.

4. Form transformation in XSLT

4.1. Basic XML merger

In order to apply the basic merger shown in Figure 4 to practical Web service integration, we select an target application which generates an individual examination schedule for each student from the

individual timetable for classes and the examination schedule.

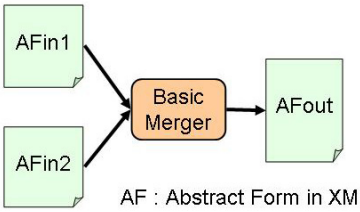

Figure 4: The basic merger for service integration.

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 to modify the system when the inputs change. In order to solve this problem, the procedure is described in a script language, and a visual tool supports the end-user. The procedure extracting classes which are included in both inputs, is composed of the following four steps:

1. Assign the nodes of two input 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 if these elements are the same.

For these processes, the XSLT stylesheet is used. 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 even though this scripting is rather easy compared to programming.

We developed a visual tool which generates the XSLT stylesheet. First, file names are input. These 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 elements to be compared. That is, in this application, the subject names are selected and displayed as follows:

F0 /examine[1]/subject[n]/name[1]
S0 /personaltimetable[1]/subject[n]/name[1]

F0 and S0 are symbols for the corresponding XPaths. The user can define conditions for the comparison by using these symbols as F0 = S0. Finally, the XSLT stylesheet is generated by entering the output file name.

4.2. 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 with complicated business logic. As an example, we chose a system to advise students on graduation requirements. 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 the credits 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, where each category has individual conditions.

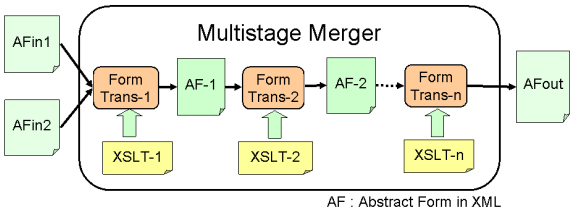

Figure 5. A configuration of the multistage merger.

For dealing with such complicated business logic, the multistage merger is introduced as shown in Figure 5. 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, in the template for compulsory subjects, the “totalunits” is calculated by using Xpath with the sum function and the current function. The “full” is calculated by using Xpath with the not function.

Finally, in this application, the seven stages were necessary for the confirmation of conditions required for graduation.

5. Form transformation by mapping

5.1. Form-to-form transformation (FTFT)

One solution to the problem of the form transformation in XSLT is the form transformation by mapping from input forms to output forms. The end-users need not to learn XML and XSLT technologies since they can define the form transformation procedure by only mouse manipulations to relate items in input forms to items in output forms. After defining of this procedure, the form transformations from input forms into output forms is executed.

Let’s consider a Web application of goods sales. For example, a login form is transformed into a query form to a database for member management and then

the result form from the DB is transformed into a form for the member. Figure 6 shows one part of form flows and form-to-form transformations. FTFT (2:1) implies the form transformation from two inputs into one output. That is, the sales management subsystem gets data of the stock report from the stock management DB and data of a user balance from the account management DB, and displays the sales window. Next, FTFT(1:2) implies the form transformation from one input into two outputs. That is, the sales management subsystem gets data of the purchase order from a client, and sends one sales report to the stock management DB and another sales report to the account management DB.

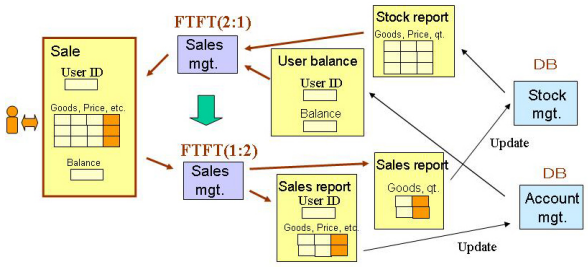

Figure 6. A part of form-to-form transformation.

In this figure, the sales window on the left hand side is an actual visual form, but four other forms imply abstract forms which are not displayed visually at the application execution time. Such an abstract form is used by end-users when constructing an application.

5.2. A visual tool for FTFT

A tool for definitions and executions of the form-to-form transformations was developed. The user interface was implemented in HTML and JavaScript. The generated procedure in XML is sent to the server and stored there. The interpreter of this procedure in XML was implemented in Java.

Figure 7 shows an example of a form-to-form transformation definition. The input form and the output form are displayed on the left. The palette with operators, variables and functions is displayed on the right. The circled numbers are added for explanation of the order of mouse manipulation. Whenever a column of forms or an operation item of the palette is clicked, the order and the operation item are displayed below for confirmation. The first three operations define that the value of the item name column in the input form is copied into the item name column in the output form. The INIT operation implies initialization of execution as the previous execution result is not used. The following five operations define that x := Price * Quantity. The last three operations define that Total := x. Alternatively, of course, the operations of Total := Price * Quantity is possible without use of the variable.

As shown in this example, the variables of x, y and z are used to temporarily store execution results. The functions of f, g and h are used for complex business logic. For example, operations for definition of Total := f(Price, Quantity) is a sequence of clicks as {Price, =, f, INIT, Quantity, =, f, INIT, f, =, Total}. The body of the function, f, is defined in a scripting language.

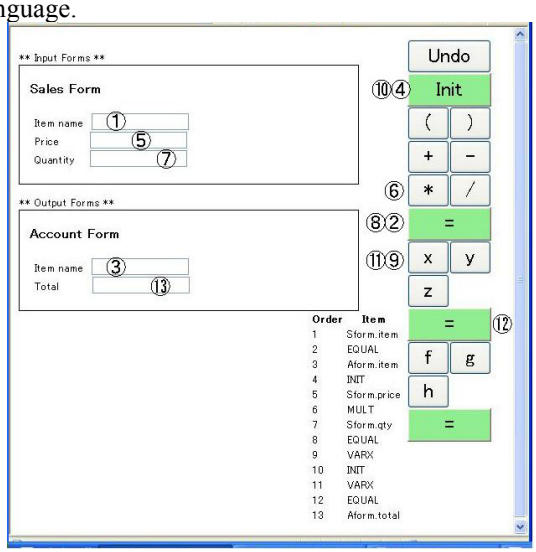

Figure 7. Form-to-form transformation definition.

6. Conclusions

The form-based approach for Web services integration by end-user initiative application development was proposed. Our experiments confirmed the effectiveness of this approach.

References

[1] Chusho, T., Tsukui, H. and Fujiwara, K. : A Form-base and UI-Driven Approach for Enduser-Initiative Development of Web Applications, Applied Computing 2004, IADIS, pp.II/11-II/16, 2004.

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

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

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

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

[6] Sutcliffe, A. and Mehandjiev, N.(Guest Ed.), End-user development, *Comm. ACM*, V47, N9, pp. 31-32, 2004.