January 31, 2006

RT0642
Computer Science; Service Science 8 pages

Research Report

Easy SOA: Rapid Prototyping environment with Web Services for End Users

Takayuki Yamaizumi, Takashi Sakairi, Masaki Wakao, Hideaki Shinomi, Samuel Adams

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

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Easy SOA: Rapid Prototyping environment with Web Services for End Users

Takayuki Yamaizumi, Takashi Sakairi
Tokyo Research Laboratory, IBM Japan
{*zumi,sakairi*}*@jp.ibm.com*

Masaki Wakao, Hideaki Shinomi
Yamato Software Laboratory, IBM Japan
{*wakao,shinomi*}*@jp.ibm.com*

Samuel Adams
IBM Research
ssadams@us.ibm.com

Abstract

Wikis and blogs allow end users to do their work with only a Web browser, since they do not need to install any client application to their personal computers. This implies that an application developments tool for end users should not require installing special additional programs nor adding special configurations to their personal computers. This paper describes Easy SOA. Easy SOA is a rapid prototyping model for SOA based on Ad hoc Development and Integration tool for End Users (ADIEU). With ADIEU, end users can prototype their Web applications and Web Services rapidly by putting 'cards' into a 'sheet' constructed on a Web browser. Easy SOA realizes a prototype development model for Service Oriented Architecture (SOA) as well as for Web applications and Web Services using ADIEU. Easy SOA makes it easier for end users to understand Web services, as it hides the low level details of the definition in WSDL file.

1. Introduction

Many end users edit and update their documents with tools such as a wiki [3] or a blog [2] with just a Web browser, since they do not need to install any client applications if they decided to use a wiki or a blog provided by an Internet service provider.

Some end users have also become involved in "End-user programming" by creating various tools to make their personal work more effective. Spreadsheet[27], HyperCard [23] and its clones are the most successful experiences of "End-user programming", but it is too hard for users to migrate their artifacts into other "End-user programming" environment. Because of a barrier hindering migration to the other development environments, end users prefer asking application developers to develop new applications rather than migrating their application into the other development environments.

First, this paper describes Ad hoc Development and Integration tool for End Users (ADIEU) [1]. ADIEU works on a Web browser by communicating with an ADIEU server without installing any special development environment in a personal computer, shown in Figure 1. With ADIEU, end users can prototype their application rapidly by placing cards into a sheet which is displayed in a Web browser and represents an application. ADIEU also has a function to import external Web Services, creating new cards to represent the imported Web service methods by reading WSDL (Web Services Definition Language) files [12] and by generating the

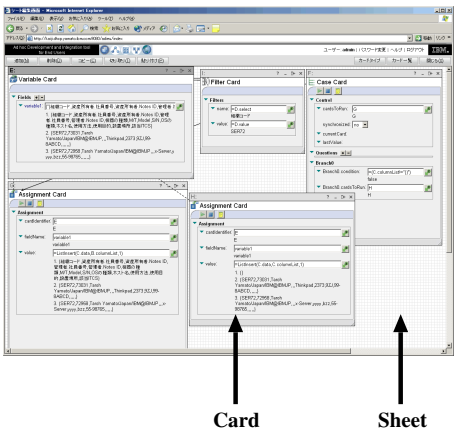


Figure 1. Sample ADIEU screen

cards for each Web service methods.

Second, we propose Easy SOA, which is a prototype development model for the Service-Oriented Architecture (SOA) to create Web applications and Web Services with ADIEU.

The rest of this paper is organized as follows: Section 2 describes the motivations for this paper. ADIEU is introduced in Section 3, followed by an explanation of how ADIEU imports an external Web service in Section 4. Section 5 illustrates our Easy SOA model and a development scenario based on it, followed by a "Historical Stock Quote in any currency units" development example. Finally, in Section 6, we discuss our work and related work. The conclusion and future work is summarized in Section 7.

2. Motivations

Many end users write tools to make their personal work more effective. However, when end users ask application developers to develop applications for their businesses, such as a system to handle transactions, end users often find it difficult to explain to the application developers what they want. This communication gap often mislead application developers to construct a different system from what the end users had asked them to construct, and to modify the system to conform to the end users' specifications. Such iterations may also cause delays in the development, which becomes unacceptable as developers are asked to develop and to deliver ever more complex applications compared to previous versions, but in a shorter period of time[21]. For this reason, prototyping tools are needed to encourage end users to participate in system design.

Therefore, many software tools have developed for end users to design and to develop applications [25] by selecting items from graphical menus, placing text fields in windows, and by performing other graphical operations. However, to use those applications, end users have to install these applications on their personal computer. Many of end users are bothered by the installation. Even if an end user installs them successfully, it is more difficult to find useful components or user interfaces, because those components or user interfaces must have been developed by users who have skills to develop them. Finding such users are all too often impossible. Therefore, these tools cannot be adopted as development tools for end users. There are also still some technical challenges in delivering a graphical development environment for end users and non-professional programmers. The card-based programming model is one of the solutions. In the card-based programming model, an application is decomposed into separate interfaces for each effective function and some code snippets which attach the interfaces, while a traditional procedural programming language typically rendered an application to lines of text. Those separate interfaces become a kind of Integrated Development Environment (IDE) for each statement in a program, which provides a number of advantages in providing statement-specific guidance, help, and features such as activity logging and control flow-based execution tracing. Since each card represents a statement in the programming language of ADIEU, new cards, whether developed in ADIEU itself or in Java™,[1], effectively provide the ability to extend and reshape the programming language to better fit the problem domain which concerns users.

At the same time, many Web services are available to the public through the Internet [8, 11, 16], although most of them are provided as test services. They are quite useful for skilled developers who can implement the interfaces for such Web services with stub code, because these Web services prevent them from having to"reinventing the wheel". Likewise, these public Web services can be accessed through the Web interfaces generated from WSDL files [12, 15]. However, since some of the Web services require specifying complicated data structures for the input data, users must have a skill to handle complicated data, and this should not be required of end users. Web services have come to a vendor-independent open standard and can be used widely. If a development tool can import WSDL files and use the Web service methods described in the WSDL files, then users can develop applications with less cost and time.

SOA also gives us another motivation to construct a rapid prototyping environment for end users. SOA is a framework and an emerging methodology to construct enterprise systems [5], as well as to integrate existing systems into larger systems. Web services now play an important role in SOA because most SOA systems are constructed by combining Web services. However, current tools to construct SOA systems are very demanding for personal computers and lack functions to examine how the models defined in the tools actually work. Therefore, rapid prototyping tools for SOA, such as Easy SOA will become important to construct IT infrastructures. Users can define the relationships between any points in the data structures which are used as the inputs or outputs of a Web service even if their data structures are very complicated, as shown in Section 5.2.

[1] Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

3. ADIEU Overview

ADIEU is a programming environment which operates within a Web browser[2]. It can be installed in the WebSphere Application Server or Apache Tomcat. A typical screen shot is shown in Figure 1. Users can develop and deploy Web applications and Web services without installing any software onto their personal computers, because all of the information about the ADIEU environment is stored in the Web application server and an applet which creates this communication channel between the Web browser and an ADIEU server is automatically downloaded from the Web server to a client when users launch the ADIEU environment.

An application in the ADIEU environment consists of two parts, a sheet and cards. These parts correspond to an application and its logic, respectively. Users can develop the application by using collections of cards, each of which acts like single-function applications in a form-based, desktop-like environment. The data fields in cards can be used like cells in a spreadsheet and can contain either data or an expression that evaluates the data at run time. Cards can also run other cards which has a `CardsToRun` field, such as "If card", "Sequence card" and "Iterator card", by specifying a sequence of card IDs separated by commas.

Cards can also refer to a value which is held by a field in another card by inserting a special anchor to point at the field using the following format (1):

$$\langle card\ id \rangle . \langle field\ name \rangle \qquad (1)$$

For the end users' convenience, this anchor can be inserted by clicking the "insert variable" button at the right side of a text area (Figure 2), and the field name can be selected directly from the menu list. In the example shown in Figure 2, `A.variable` is inserted into the `html` field.

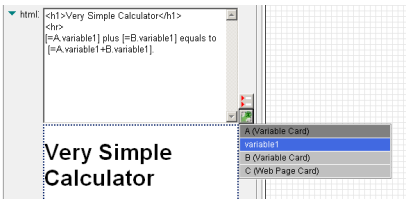


Figure 2. "Insert variable" button and a menu to select a field

These capabilities provide the basic flow control and the data transfer necessary for programming concepts such as decision branching, sequences, and loops.

Users can access their applications as Web applications by clicking the link on a Web Page Card. With a Web Service Card, users can also execute their applications as Web services without any special operations for deployment.

The Web application sample and the Web service sample are shown in Figure 3 and 4, respectively. In 4, the Web application consists of two Variable Cards and one Web Page Card and the Web service consists of two Variable Cards and one Web Service Card. The "run" card (Card *E*) is a generated card by consuming the WSDL file of the Web service. The WSDL file is automatically generated by the ADIEU environment.

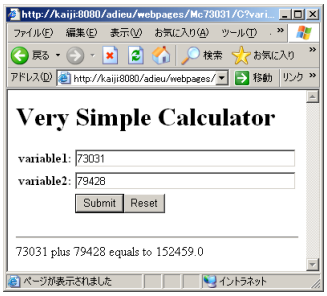


Figure 3. Running a Web application developed in the ADIEU environment

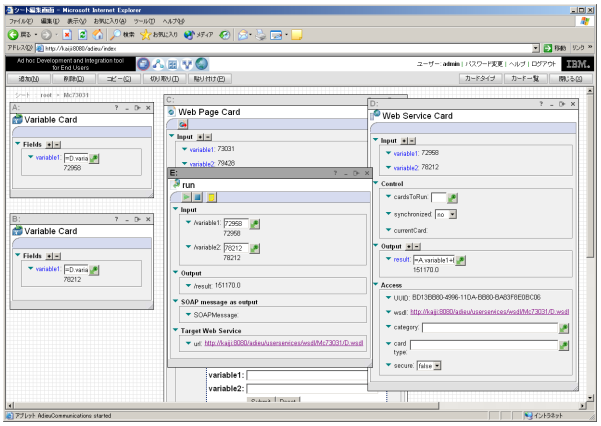


Figure 4. Specifying the URL of WSDL in the ADIEU environment itself

This function can be also used to import external Web Service. We will discuss this in the next section.

[2]Currently, ADIEU works only on the Internet Explorer.

4. Importing Web Services into ADIEU

The ADIEU environment can generate cards by reading the WSDL files on remote Web application servers. Users can use these cards to compensate for the gaps between the original functions which is provided by built-in cards and the users' requirements for their tools. Users can import an external Web service into an ADIEU environment by specifying an URL of a WSDL file, as shown in Figure 4.

In this section, we discuss these generated cards with a sample Web service. This Web service has BankInfo, UserInfo and AccountInfo classes and BankInfo class has five methods which are exported as Web service methods, as illustrated in Figure 5.

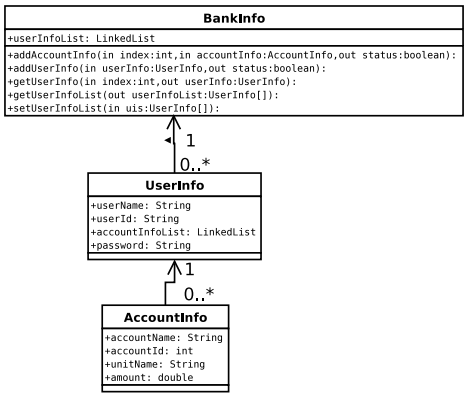


Figure 5. Class diagram of a Web service sample (internal operations are omitted)

4.1. Card Generation Example

ADIEU generates one card for each Web service method. For example, in the Web service example as illustrated in Figure 5, five cards are generated, such as addAccountInfo card, addUserInfo card, getUserInfo card, getUserInfoList card and setUserInfo card. These generated cards can be used in exactly the same way as the built-in cards. For example, users can refer to fields in these generated cards from the other cards, and vice versa. After ADIEU imports a WSDL file which includes the data structure of Web service methods, it parses the WSDL file to extract the data structures, and finally generates the generic treeview-like interfaces as shown in Figure 6.

Each card has a treeview-like interface with input fields to help an end user to understand the data structure, because Web service methods generally handle complex data types, which are defined as complexType data in a WSDL file, as their inputs and outputs. The number of fields on the card can be changed if the data include more than one array of complexType data by clicking the "Add element" button or "Delete element" button on the card. As a result, users can easily integrate external Web services into their applications.

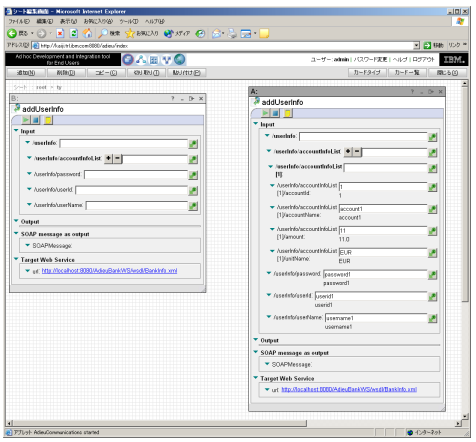


Figure 6. A screen shot example of generated cards

4.2. Support for complexType Data

In general, Web service methods are invoked with input data which is too complex for end users to handle [18, 19, 20], because they must build SOAP [13] messages in a format which it is acceptable for a Web service. Output data from a Web service method, written in the SOAP format, is also too complex for users to handle, though raw SOAP messages are useful for developers and programmers, because they can analyze them to debug their software. Accordingly, users need some tools which can support to handle both inputs and outputs data.

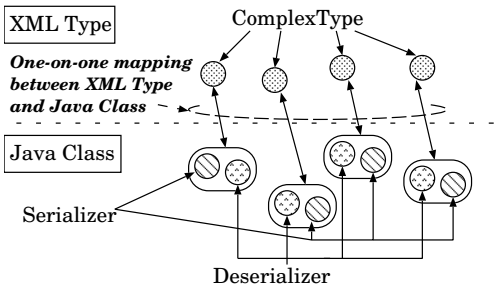
4.2.1. SimpleType and ComplexType. The format of a SOAP message is defined in a WSDL file in the XML Schema format [14]. Developers must classify the data into two types, complex types and simples type which are called complexType and simpleType in the specification of XML Schema, respectively. ComplexType data include other complexType data or simpleType data, while simpleType data can include neither complexType data nor other simpleType data. In the example shown in Figure 5, BankInfo, UserInfo, and AccountInfo are classified into complexType data, and userName, userId, password, accountName, accountId, unitName and amount are classified into simpleType data. In addition, userInfoList and accountInfoList are classified into arrays of complexType, since they hold

complexType data as elements. If a data structure defined in a WSDL file is represented as a tree graph, this tree graph possesses the following properties.

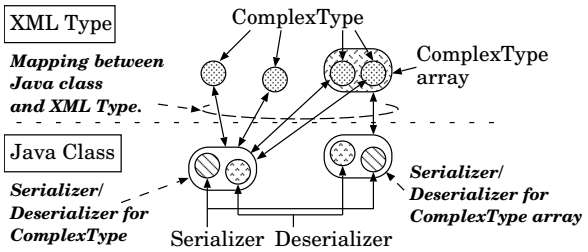
1. Data in a non-leaf node must be complexType data.
2. Data in a leaf node must be simpleType data or complexType data that does not have any child nodes, i.e. complexType data with no child data.

For this reason, we propose to display the data structure as a tree, because users can easily recognize nodes which hold complexType data. Additionally, we define an array of complex type as consisting of two types of node, one is an "element node" which is a node representing an element of an array and included in a "parent node", and the other is a "parent node", which represents an array of complexType. By this definition, users can know the data type that is represented by a node.

4.2.2. Serialization and deserialization model. The comparison between ADIEU's serialization and deserialization model and Apache Axis' original serialization and deserialization model is illustrated in Figure 7. Apache Axis expects a developer to implement and



(a) The model in Apache Axis



(b) The model in ADIEU

Figure 7. The comparison of serialization and deserialization model between Apache Axis and ADIEU

to assign Java classes to serialize and to deserialize the XML type data which is defined as complexType data in a WSDL file. Apache Axis also expects a developer to store the relationships between the Java classes and the XML types. As a result, a developer have to implement the same number of pairs of a serializer and a deserializer as those of XML type which a WSDL file includes.

An ADIEU environment handles data with two pairs, each consisting of a serializer and a deserializer. One pair which handles any complexType data, and the other pair which handles any arrays of complexType data. These serializers and deserializer are provided as a part of the ADIEU environment and can handle any XML types which are defined as complexType data in a WSDL file, so that they liberate an ADIEU user from implementation of a serializer and a deserializer. ADIEU's serializer for a complexType array overrides the ArraySerializer's `serialize()` method in Apache Axis version 1.1[17] to work on arrays of complexType data, since Apache Axis version 1.1 cannot handle relationships if more than one XML type shares the generic serializer and deserializer for complexType data.

4.2.3. Data identification in XPath representation. Once a data structure is represented as a tree graph, users can point to any data included in complexType data uniquely using an XPath format[9].

To preserve the uniqueness of names between the fields, when there is more than one card on a sheet and they have the same complexType data structure, the user specifies a data node in the complexType data in the following format (2):

$$\#\{\langle \textit{card ID} \rangle . \langle \textit{data name in XPath format} \rangle\} \quad (2)$$

The data name in the XPath format follows a card ID which includes the data, and the concatenated string is put in curly brackets to coexist with the other expressions such as mathematical expressions by using the escaping slash characters such as are used for the division symbol in a mathematical expression. Since format (2) may be long in some cases, these data names in format (2) can be automatically entered by clicking on the "Insert variable" button to select a suitable field in a menu, as shown in Figure 2.

5. Prototyping on Easy SOA

Users can build applications in a very simple style of Service Oriented Architecture, because an output of a Web service can be connected to an input of another Web Service in ADIEU in a Web browser and users can exports the application in a Web browser as a Web ap-

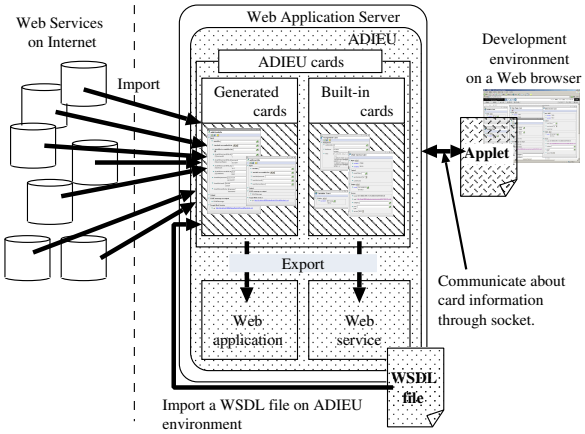


Figure 8. Easy SOA development model

plication or a Web service without any deployment operations, as we discussed in the previous section. We name this approach Easy SOA and demonstrate that users can realize Easy SOA with a simple sequence of operations.

5.1. Development Scenario with Easy SOA

The infrastructure for Easy SOA is illustrated in Figure 8. An user can develop a Web application and a Web service with the ADIEU environment by the following three steps:

1. The user imports some Web services which provide what they want to integrate their system.

2. The user orchestrates the Web service methods by defining relationships between them, as discussed in Sections 3 and 4.

3. The user exports their artifact as a Web application or a Web service. If the artifact is exported as a Web service, other users may integrate it into their application even if they integrate it without ADIEU. A WSDL file for the Web service is automatically generated by the ADIEU environment.

The above development scenario shows that an user can construct another Web service recursively from those which has been imported by the user without any code.

In Easy SOA model, built-in cards which are supplied with an original ADIEU environment mainly help the orchestration of the generated cards.

5.2. Development example using Easy SOA: Historical Stock Quotes

For example, suppose we have decided to construct a Historical Stock Quotes Web application as shown in Figure 9, which can return a stock price for any past date. This Web application also displays the result in Japanese Yen (JPY) by calling a Web service which can convert the currency unit at the rate on the past day to check the profits which we receive from our shares. We can construct the service by following these steps:

1. The user imports a Web service and generates a card which can convert a currency unit on any past date [16], and place it on a sheet . The ADIEU environment assigns *A* as the card ID.

2. The user places a Web Page Card (card *B*), an Assignment Card (card *C*), and a Variable Card (card *D*) onto the sheet and fills the fields in the Input fieldset, as shown in Figure 10.

3. The user imports another Web service and generates a card (card *E*) which can return a stock price on any past date [8], and place a card which represents a suitable method on the sheet.

4. The user defines the relationships between the cards by typing a sequence of evaluations such as `C,A,E` in the `cardToRun` field on card *B*.

We can examine how the card works at each step in the development, so that we can easily test the application. Figure 9 shows a typical ADIEU screen after the user has been finished the above steps. There are five cards placed in a Web browser.

By clicking on the link on a Web Page Card, the Historical Stock Quotes application is started, as shown in Figure 10. Since the foreign exchange Web service provides the exchange rates at noon on the specified date and the Historical Stock Quote Web service provides a stock price at the end of the day on the specified date, this Historical Stock Quote Web application provides a rough figure as a result. However, this value may be still important for end users, because they sometimes want to check whether or not their stocks are profitable, especially if they hold stock in foreign companies.

6. Related Work

To develop the enterprise system based on SOA, developers can define Web services behaviors for business processes with the Business Process Execution Language for Web Services (BPEL4WS[10]). Tool support has also been provided to make the definitions

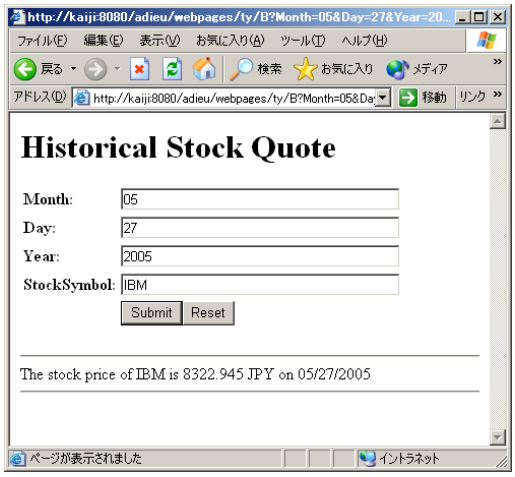


Figure 9. Screenshot of"Historical Stock Quote in JPY" Web application

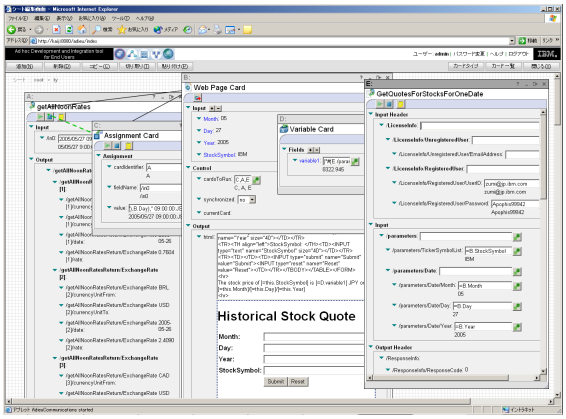


Figure 10. Cards which consist of "Historical Stock Quote in JPY" Web application

in BPEL4WS easier. However, developers must have deep understandings and analyze their business processes and the interactions of the Web services which they wants to define in the BPEL file. It is also difficult for developers to examine how it works before the actual system has been constructed.

The data structure of a Web service is written in XML Schema [14] format. Therefore, the Web interface can be generated by reading the data structure definitions in a WSDL file from the Web service provider [15], and users can confirm and test the functions of the methods in the Web services. However, this test service [15] lacks functions to develop applications by defining the relationships between Web service methods. Huy proposed Toshiba Web Service Gateway [22], which can easily convert information sources in the Internet into Web services with a Web service wrapper. The wrapper generator tool generates the Web service wrapper by reading a wrapper description file, but this approach is not suitable for end users because a wrapper description file must be designed with some modules and an user also must design the modules if they do not exist.

HyperCard[23] and its clones are the most successful experience of "End-user programming", but it is too hard for users to migrate their artifacts into the other "End-user programming" environment. Moreover, Users must create cards themselves by placing small GUI parts and write code to define the relationships between cards.

Tanaka [7] applies their Meme media technologies to the publication and reuse of intellectual resources on the Web. Users can organize resources in Web pages by editing Web pages which are wrapped by meme media wrappers. Pautasso[26] and Alonso designs BioOpera and JOpera. They are fully implemented in a development environment for Web Service composition with usability features emphasizing rapid development and visual scalability. However, users have to install the special application to their personal computer and meme media wrapper cannot wrap Web services. BioOpera and JOpera still need manual intervention to resolve ambiguities and connect parameters between Web Services. In this paper, we show how to connect parameters with an XPath representation.

7. Conclusions and future work

In this paper, we first described ADIEU, which allows end users to develop Web services and Web applications without installation by placing cards on a sheet which is displayed in a Web browser, and then by defining the relationships between cards with minimal mouse and keyboard operations. For this reason, we conclude that this tool can be used most effectively in the prototyping phase of development, because end users can participate in a design process of an application's development and communicate with the developers about their applications with their artifacts in the ADIEU environment. Collaborative development may contribute a great deal to shorten the development time for an application and result in improved satisfaction for end users. Unlike other development tools for end users, ADIEU can import an external Web service and can generate cards which allow end users to integrate it into their own application in the ADIEU environment on a Web browser. Thus, end users can concentrate on their own work without "reinventing the wheel". We defined two

pairs of generic user interfaces, serializers and deserializers for complexType data and for arrays of complexType data. This approach with generic implementations is completely different from existing approaches and liberates end users from the maintenance work of Web client code for their business.

Second, we proposed the concept of Easy SOA by connecting the interfaces between cards which represent Web service methods using a Web browser with simple operations. The XPath representation style for complexType data structures will provide a starting point to enhance this representation to a simpler format, even if another representation is required for end users. We also described that users can easily construct Web service from the other Web services which they import into ADIEU environment with a minimal effort.

We have described two development scenarios. One is a very simple calculator Web application, and the other is a Historical Stock Quotes service created by orchestrating two Web services which are deployed by different organizations. Those examples demonstrate that the ADIEU environment can realize SOA services in a simple way.

Some technical challenges for future works still remain. In this paper, we do not discuss about the user interface of the ADIEU card. XPath representation of nodes is a convenient method for a computer, but it is not so convenient for an end user. YAML [24] or another simplified representation may be a solution, but, this investigation will be an important future works.

References

[1] Ad Hoc Development and Integration Tool for End Users (ADIEU), http://www.alphaworks.ibm.com/tech/adieu

[2] B. Nardi, D. Schiano, M. Gumbrecht, and L. Swartz; "Why We Blog", *Communications of ACM*, December 2004, 47(12), pp.41-46.

[3] Wiki: "What Is Wiki", http://wiki.org/wiki.cgi?WhatIsWiki

[4] Endrei, M. et al, Patterns: Service-Oriented Architecture and Web Services, http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf

[5] Service-Oriented Architecture, http://www.ibm.com/software/info/openenvironment/soa

[6] S. M. Kim and M. Rosu, "A Survey of Public Web Services", In *Proceedings of the 13th International World Wide Web Conference 2004*, pp.312-313, Chiba, Japan, 2004.

[7] Y. Tanaka, "Meme Media Architecture for Intuitively Accessing and Organizing Intellectual Resources", *Intuitive Human Interfaces 2004*, Springer-Verlag, LNAI 3359, pp. 108-126.

[8] StrikeIron, Your Trusted Web Services Marketplace, http://www.strikeiron.com/

[9] W3C, XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath

[10] Business Process Execution Language for Web Services version 1.1, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[11] XMethods web site, http://www.xmethods.net/

[12] W3C, Web Services Description Language (WSDL) 1.1, W3C Note, 2001.

[13] W3C, SOAP - Simple Object Access Protocol, http://www.w3.org/TR/SOAP

[14] W3C, XML Schema, http://www.w3.org/XML/Schema

[15] Mindreef: Comprehensive Web services diagnostics and testing, http://www.mindreef.com/

[16] Federal Reserve Bank of New York, Pilot Noon Foreign Exchange Rates Web Service, http://www.ny.frb.org/markets/pilotfx.html

[17] Web Services - Axis, http://ws.apache.org/axis/

[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] M. Bochicchio and N. Fiore, WARP: Web Application Rapid Prototyping, *ACM Symposium on Applied Computing*, March 14-17, 2004, Nicosia, Cyprus, pp.1670-1676.

[22] H.P.Huy, T.Kawamura and T.Hasegawa, How to Make Web Sites Talk Together - Web Service Solution, In *Proceedings of WWW 2005*, May 10-14, 2005, Chiba, Japan.

[23] D. Goodman, The The Complete HyperCard 2.2 Handbook, Iuniverse Inc, 1998.

[24] YAML Ain't Markup Language (YAML) Version 1.1, http://yaml.org/spec/current.html, 2004.

[25] C. Kelleher and R. Pausch, Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers, *ACM Computing Surveys*, Vol. 37, No. 2, June 2005, pp. 83-137.

[26] C. Pautasso and G. Alonso, Visual Composition of Web Services, In *Proceedings of VL/HCC* 2003, pp 92-99.

[27] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet and C. Wallace, End-User Software Engineering with Assertions in the Spreadsheet Paradigm, In *Proceedings of the 25th International Conference on Software Engineering*, 2003.