A Form-based Approach for Web Services by Enduser-Initiative Application Development

Takeshi CHUSHO, Katsuya FUJIWARA, Hisashi ISHIGURE and Kei SHIMADA
Department of Computer Science,
School of Science and Technology, Meiji University
Kawasaki, 214-8571, Japan
chusho@cs.meiji.ac.jp

Abstract

The number of end-users using the Internet increases on the inside and outside of offices. Enduser-initiative development of applications has become important for automation of their own tasks. Especially, applications for web services should be supported by business professionals themselves because web services must be modified frequently. This paper describes enduser-initiative application development methodologies for the MOON(multiagent-oriented office network) systems including window work in B-to-C and B-to-B electronic commerce. The window work is considered as a metaphor of the interface between service providers and their clients for web services. The front-end of the system is constructed by a multi-agent framework. The back-end is developed by modeling techniques. The multi-agent framework is a Java application framework and is designed based on the simple concept that "one service = one form." It provides form-based interfaces as a common XML-base protocol of {who, what, how } for end-users, which protocol corresponds to the UDDI protocol of {white, yellow, green } pages. Then form transformation from an input form into output forms implies service integration. Prototyping for the front-end system and the back-end system is performed in Java and XML[1].

Key words : web service, multi-agent, application framework, object-oriented technology, electronic commerce

[1]H. Ishigure and K. Shimada are with Hitachi, Ltd. and Hitachi Software Eng. Co., Ltd. at present respectively.

1. Introduction

The number of end-users using the Internet increases on the inside and outside of offices. Enduser-initiative development of applications has become important for automation of their own tasks. Especially, applications for web services should be supported by business professionals themselves because web services must be modified frequently.

As the solution based on CBSE(Component-Based Software Engineering) [2], this paper describes a form-based approach for end-user computing under distributed systems. As a typical distributed information system, we direct our attention to an application system for windows or counters in banks, city offices, travel agents, mail-order companies, etc. Some kind of window work such as mail-order business has already been put to practical use in current computer networks including the Internet 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 near future, the information society will require such new technologies that domain experts can automate their own work by themselves and that almost all clients can operate computers at home or at office without extra training or without the help of others.

In the business world, recently, the external specifications of application software is considered as services. The various infrastructures such as SOAP [15], UDDI [14], WSDL [16] are proposed for rapid web service provision. For practical use in e-marketplaces, some problems on security, trust, quality etc. must be solved. Therefore, such

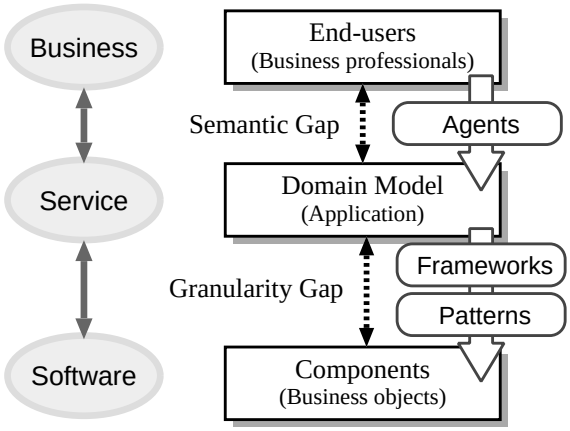

Figure 1. Technologies for bridging gaps between end-users and components.

infrastructures may pervade from local or private areas such as private UDDI to global areas such as e-marketplaces.

It becomes necessary for end-users to develop and maintain applications because web services change continuously. 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, that is, C-to-D (Client to Department) and D-to-D (Department to Department) rather than B-to-B and B-to-C.

Multi-agent systems must be the solution for these problems because end-users may teach their operations to agents without programming [1, 9, 10]. Actually, multi-agent systems are used for advanced applications based on distributed systems and the Internet such as electronic commerce support systems [11]. An agent communication language(ACL) is one of the key technologies for interactions among independently-developed applications with agents. Then the standardization is being tried by FIPA(Foundation for Intelligent Physical Agents) [6] and OMG(Object Management Group) Agent WG [13].

This paper presents the enduser-initiative approach in Section 2, prototyping of the front-end system in Section 3, prototyping of the back-end system in Section 4.

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. For "just-in-use," enduser-initiative development is required.

Our approach to how to make web applications supporting web services is shown in Figure 1. 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 business professionals or 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 is defined. At the software level, the domain model is implemented by using components to be combined.

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 [8] and application frameworks [5] based on CBSE(Component-Based Software Engineering). On the other hand, the semantic gap is bridged by multi-agent systems.

2.2. Metaphors for web services

As a typical distributed information system, we direct our attention to application systems for window work. Such window work is not limited to the actual window work in the real world. For example, in a supply chain management

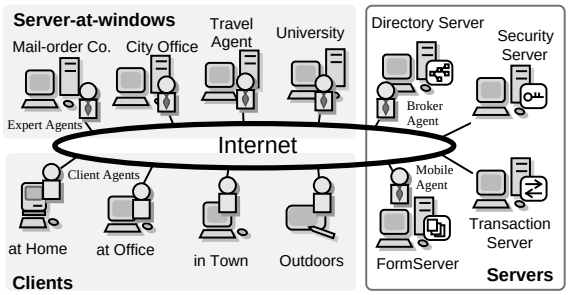

Figure 2. A MOON(multiagent-oriented office network) system.

system (SCM), exchanges of data among related applications can be considered as the virtual window work.

Of course many kinds of window work 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 agent-based applications by end-users.

Generally, window work or counter work is considered as service requests between clients and service providers. 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”

Our actual XML-base protocol of {who, what, how } for end-users corresponds to the UDDI protocol of {white, yellow, green } pages, and form transformation from an input form into output forms implies service integration, as mentioned later.

2.3. Application architecture

Agent-based applications are constructed on a multiagent-oriented office network (MOON) for window work [3]. The MOON system is based on a client/server model and is partitioned into the following three parts as shown in Figure 2:

1. Client terminals with client agents for sending written applications to windows, such as personal computers and workstations both at home and at office, public telephones with terminals in town, portable computers for mobile computing outdoors, etc.

2. Server-at-windows with expert agents for receiving written applications, such as windows in mail-order companies, city offices, travel agents, universities, etc.

3. MOON servers for managing the network system.

The MOON servers imply the following four servers and some of them may be located physically in server-at-windows:

1. A directory server with a broker agent manages network addresses of server-at-windows to receive written applications as service directories of windows.

2. A form server with a mobile agent manages various application forms for services at these windows, which forms are defined with help messages and selection menus by domain experts.

3. A transaction server stores written applications received by server-at-windows with the identification numbers, manages the states of the process and replies to inquiries about the states. It may be connected with a workflow system in the organization including the server-at-window.

4. A security server controls access rights to server-at-windows and the MOON servers, and manages authentication of clients.

In our experiences of prototyping, the actual system configuration is the 4-tier architecture of browsers, web servers, application servers and DB servers. The front end of the system is supported by application frameworks and multiagents. The back end is supported by domain modeling and business objects.

In the remainder of this paper, these prototyping experiences are described.

3. Prototyping of Front End

3.1. Features of agent-based applications

A customizable multi-agent system is developed as an application framework which implies a reusable semi-complete application that can be specialized to produce custom application. Then the customization of the hot spots in the application framework implies the agent development by end-users.

The first feature is electronic form processing which is navigated by agents both in client terminals and in server-at-windows. Clients can teach the fixed operations of filling in a form about such plain words as their names, addresses and phone numbers to their agents. Then their agents do so instead. Domain experts can teach their expertise to their agents. Then the agents guide clients in filling in the form and check the written form.

The second feature is standardization of ACL for communication between client agents and expert agents. Design of ACL depends on features of multi-agent systems. This paper describes cooperative multi-agent systems because clients and domain experts are cooperative in the most cases of the form-based application domain. Furthermore, every agent is developed independently and has an individual goal in the heterogeneous environment, because each of client agents and expert agents can perform each task alone in the form-based application domain.

3.2. The basic form of web service interface

Messages of requests to windows include the following elements:

- Who receives your request?
- What do you request to the window?
- How do you request it?

The name of the multi-agent framework, wwHww, is derived from ‘who-what-how with WWW’ and is pronounced as ‘who’ for convenience. The following element is added to these three elements.

- Which is your request?

That is, the basic form is shown as follows:

(who, what, how, which)

The semantics is based on a message passing concept of conventional object-oriented programming languages. The four parameters correspond to elements of a message between objects respectively as follows:

who : A message receiver object

what : A method name

how : Parameters for a method invocation

which : A message number

In the form-based approach, the who-parameter implies a window where a written application is sent to. The what-parameter implies the title of the application form. The how-parameter implies contents of the application form. The which-parameter implies a receipt number stamped on the received written application.

The states of values of these parameters affect semantics of the message. If a value of a message parameter is unknown, the message implies an inquiry about the parameter. This semantics is quite different from conventional object-oriented programming languages. This extension, however, produces attractive effect. Examples are given in the next subsection.

3.3. End-user interface

The actual end-user interface for filling in the form is different from the basic form which is the internal representation in the system. Examples of requests are given in the basic form for convenience, where the following notations are used:

a, b, ... : Parameters with known values.

?a, ?b, ... : Inquiries about the parameters with known values, which request help messages.

x, y, ... : Parameters without values.

?x, ?y, ... : Inquiries about the parameters themselves, which request all possibles for selection.

1. (a, b, c, x)

   The written application, b, with the contents, c, is sent to the window, a. A message number will be assigned to the variable, x, by the window receiving this message.

2. (a, b, , ?d)

   The state in the process of the written application, b, of the message number, d, is inquired of the window, a.

3. (a, b, ?x,)

   The application form, b, to be sent to the window, a, is displayed. How to fill in the form is navigated by the expert agent. Some typical items are filled automatically by the client agent.

4. (a, ?x, ,)

   The title list of all application forms which the window, a, receives, is displayed.

5. (?x, ?y = (a list of keywords), ,)

   The list of titles of all application forms which relate to the list of keywords, is displayed with the names of windows receiving them. The system retrieves forms whose titles include the keywords or in which help messages include the keywords.

6. (?a, , ,)

   The explanation on the work of the window, a, is displayed.

7. (a, ?b, ,)

   The explanation on the application form, b, to be sent to the window, a, is displayed.

These inquiries must be simpler for end-users than those of UDDI, although our actual XML-base protocol of {who, what, how } for end-users corresponds to the UDDI protocol of {white, yellow, green } pages.

3.4. Software architecture

The first version of the multi-agent framework, wwHww, has been developed with a library system, which is used in our laboratory. Such a form-base system is helpful to us since there are no librarians in our laboratory, For example, we can know who borrowed some book because everyone fills in an electronic application form when taking out a book from our laboratory. We can know whether some book has been already registered or not because everyone

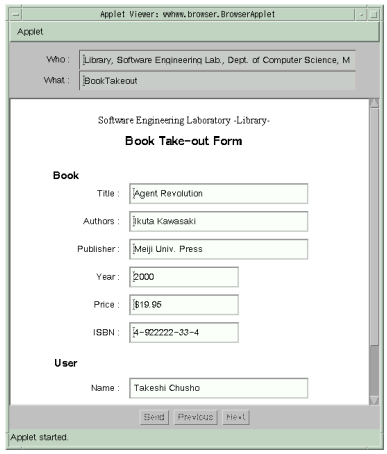

Figure 3. An example of the wwHww browser at a client terminal.

fills in an electronic application form after he or she bought the book for our laboratory.

An example of the wwHww browser for taking out books is shown in Figure 3. The head part indicates the name of the server-at-window in the who-parameter and the name of the service in the what-parameter. The white part implies the how-parameter, that is, the electronic application form itself requested.

The software architecture is shown in Figure 4. The wwHww browser of the client side is composed of two subsystems, that is, the form browser and the directory browser. The wwHww server of the server side is composed of three subsystems, that is, the directory server, the form server and the transaction server. This system was implemented in Java and there are Java applet versions and Java application versions for two browsers.

3.5. An application building procedure

Domain experts build expert agents by using the framework as follows:

1. Service definitions : Services at the window, are defined.

2. Form definitions : Electronic forms for these services are defined while embedding navigation information into these forms.

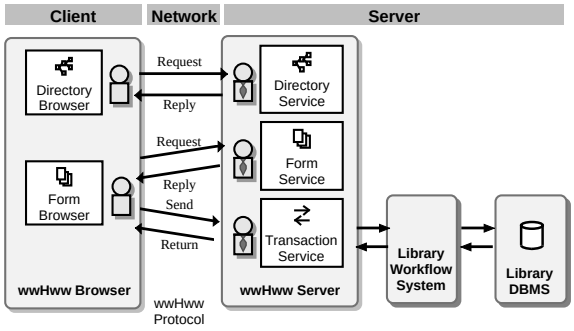

Figure 4. An application and the multi-agent framework.

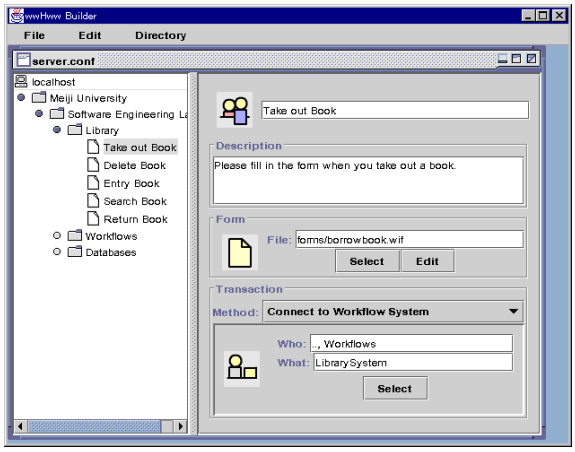

Figure 5. An example of the browser for system definitions by domain experts.

3. Registration : These definitions are registered into the corresponding servers.

An example of a browser for the library system definitions is shown in Figure 5. The left-hand part implies a hierarchical directory. The right-hand part implies definitions about the service for taking out books.

3.6. Examples of XML-base navigation

Intelligent navigation by agents is implemented in XML base [7]. The meta data for a window is described in an RDF(Resource Description Framework) style for automation. An example of a library is partly shown as follows:

```
<!DOCTYPE RDF [
  <!ENTITY site 'http://se.cs.meiji.ac.jp'>
]>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
 xmlns:d="http://purl.org/dc/elements/1.1/"
 xmlns:w="http://wwhww.org/1.0/">
<w:Agent rdf:about="&site;/library/">
  <d:Title>Library</d:Title>
  <d:Description>The library sys-
tem of the SE lab.</d:Description>
  <d:Text>The library sys-
tem of the SE lab.&lt;br&gt;Book take-out ser-
vice</d:Text>
  <w:name>/Meiji-U/CS/SE/Library</w:name>
  <w:service rdf:resource="&site;/library/takeout"/>
  <w:service rdf:resource="&site;/library/return"/>
  <w:service rdf:resource="&site;/library/entry"/>
  <w:service rdf:resource="&site;/library/modify"/>
  <w:service rdf:resource="&site;/library/search"/>
</w:Agent>
<w:Form rdf:about="&site;/library/takeout">
  <d:Title>take-out</d:Title>
  <d:Description>A procedure for book take-
out</d:Description>
</w:Form>
</rdf:RDF>
```

While forms are defined in HTML in the conventional way, the semantics of forms are defined in an RDF style for automation also. An example of meta data for form definitions is partly shown as follows:

```
<RDF
 xmlns="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
 xmlns:d="http://purl.org/dc/elements/1.1/"
 xmlns:w="http://wwhww.org/1.0/">
<w:Form about="http://se.cs.meiji.ac.jp/library
/takeout/">
 <d:Title>take-out</d:Title>
 <d:Description>A procedure for book take-
out</d:Description>
 <w:input>
  <w:FormItem>
   <w:name>usr</w:name>
   <w:datatype resource="http://imc.org/vCard/3.0#FN"/>
   <w:value resource="urn:userprofile:@user.name.fulln
ame"/>
   <w:help>http://inside.se.cs.meiji.ac.jp/library/tak
eout/help.html#usr</w:help>
  </w:FormItem>
 </w:input>
</w:Form>
</RDF>
```

An example of a request message for displaying a list of services which are provided by the library system,

(a, ?x, ,),

is shown as follows:

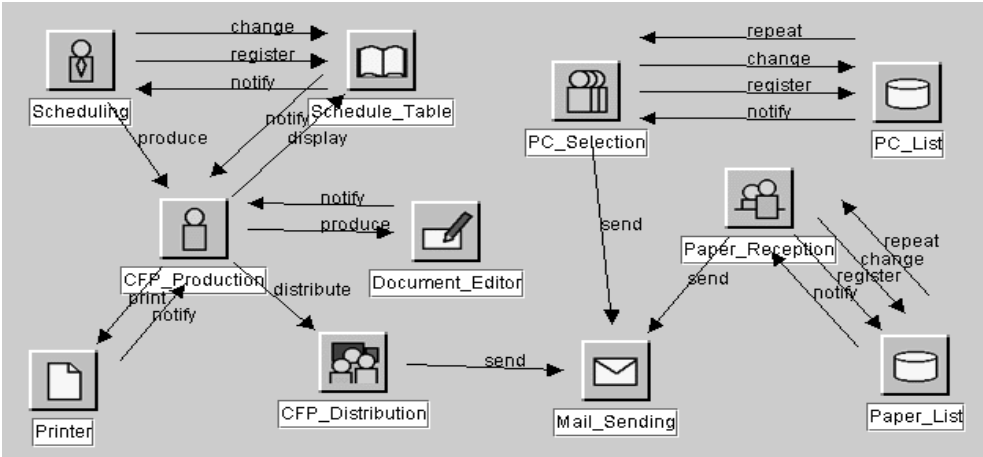

Figure 6. An example of domain model constructed by using the modeling tool.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
 xmlns:w="http://wwhww.org/1.0/">
 <w:Message rdf:about="">
  <w:who>/Meiji-U/CS/SE/Library</w:who>
  <w:what><w:query/></w:what>
 </w:Message>
</rdf:RDF>
```

4. Prototyping of Back End

4.1. Modeling process

The business logic of the back-end system must be constructed by end-users of business professionals or domain experts. One solution for enduser-initiative application development 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 based on an object-oriented model. From this formula, the other formula of

"analysis ≡ design"

is derived since it is not necessary to convert a domain model into a computation model with application architecture. This process requires necessarily a fixed architecture and ready-made components as business objects for component-based development. Application development environment, M-base, supports these formulas [4].

In our component-based development process by using M-base, an application architecture is fixed and the behavior of a domain model is first constructed. That is, the application architecture is composed of a model, a user interface and components. At the first stage, the system behavior is expressed as a message-driven model by using a modeling tool while focusing on message flow and components. At the second stage, a user interface is generated automatically and may be customized if necessary. Then transition diagrams of user interfaces are generated automatically and are used for confirmation of external specifications of the application. Finally, the system behavior is verified by using a simulation tool.

4.2. An Example

The following modeling procedure is described while giving an example:

1. Definitions of external specifications
2. Construction of a domain model
3. Refinement of user interfaces
4. Simulation of behavior

We use a given example of "tasks of a program chair for an international conference" which is defined and is used since 1998 by the Working Group on Requirement Engineering, the Special Interest Group on Software Engineering, Information Processing Society of Japan [12].

Figure 7. The form transformation for web service integration

The domain modeling step is most essential. The modeling and simulation tool is used for constructing the domain model by mouse manipulation as a kind of visual programming tool. A dynamic model was constructed as shown in Figure 6 while introducing eleven kinds of objects. Objects are defined by drag-and-drop from the palette of icons. A message between objects is defined by drawing an arrow line from the source object to the drain object.

4.3. Form transformation

Generally business objects may be replaced with web services while the message flow is replaced with the form flow. Then as the main functions of a business object are specified in message transformation from an input message into output messages, the main functions of some web service may be specified in form transformation from an input form into output forms as shown in Figure 7. It provides a way for web service integration.

For example, when the CFP_production object in Figure 6 is received an input form for CFP production, it sends a form for requesting the schedule to the Schedule_Table object, a form for editing the CFP to the Document_Editor object, a form for printing it to the Printer object and a form for requesting the distribution to the CFP_Distribution object.

5. Conclusions

The Form-based approach for web services by enduser-initiative application development was proposed. The front end of the system is supported by application frameworks and multi-agents. The back end is supported by domain modeling and business objects. Our experiences of prototyping were described.

Acknowledgment

The authors express their gratitude to members of the survey team for dynamic software service technology, The Strategic Software Research Forum, for invaluable discussions.

References

[1] Bradshaw, J. M., “An Introduction to Software Agent,” Software Agent, MIT Press, pp.3-46, 1997.
[2] Brown(Ed.), A. W., “Component-based software engineering,” IEEE CS Press, 1996.
[3] 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, Oct. 2000.
[4] 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, Dec. 2000.
[5] Fayad, M. and Schmidt, D. C. (Ed.), “Object-Oriented Application Frameworks,” Commun. ACM, Vol. 39, No. 10, pp. 32-87, Oct. 1997.
[6] FIPA, “Agent Communication Language,” FIPA Spec 2-1999, Draft ver.0.1, Apr. 1999.
[7] Fujiwara, K. and Chusho, T., “Enduser-Oriented Distributed Application Framework, wwHww, - Intelligent Navigation based on XML,” (in Japanese), IPSJ sigSE, No. 2000-SE-128, pp.1-8, July 2000.
[8] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., Design Patterns, Addison-Wesley, 1995.
[9] Griss, M. L., and Pour, G., “Accelerating Development with Agent Components,” IEEE Computer, Vol. 34, No. 5, pp.37-43, May 2001.
[10] Jennings, N. R., “An Agent-Based Approach for Building Complex Software Systems,” Commun. ACM, Vol. 44, No. 4, pp.35-41, Apr. 2001.
[11] Maes, P., Guttman, R. H. and Moukas, A. G., “Agents That Buy and Sell,” Commun. ACM, vol.42, no.3, pp.81-91, Mar. 1999.
[12] Ohnishi, A., “Requirements Engineering Working Group (in Japanese),” Winter Workshop in Kouchi, IPSJ Simposium series Vol.99, No. 1 pp.21-26, Jan. 1999.
[13] OMG Agent Working Group, “Agent Technology, Green Paper,” OMG Document no. ec/99-12-02, Dec. 1999.
[14] UDDI, “UDDI Technical White Paper,” http://www.uddi.org/, Sep. 2000.
[15] W3C Note, “Simple Object Access Protocol (SOAP) 1.1 ,“ http://www.w3.org/TR/SOAP/, May 2000.
[16] W3C Note, “Web Services Description Language (WSDL) 1.1,“ http://www.w3.org/TR/SOAP/, Mar. 2001.