

Pollock: Automatic Generation of Virtual Web Services from Web Sites

Yi-Hsuan Lu
Penn State Univ.
yx1195@psu.edu

Yoojin Hong
Penn State Univ.
yoojin.hong@hotmail.com

Jinesh Varia*
UBMatrix
jinesh.varia@ubmatrix.com

Dongwon Lee†
Penn State Univ.
dongwon@psu.edu

ABSTRACT

As the usage of *Web Services* proliferates dramatically, new tools to help quickly generate web services are needed. In this paper, we propose a methodology that helps to *automatically* generate Web Services from the FORM-based query interfaces of a web site. Since the majority of web data are rather “hidden” behind such a FORM interface, we believe turning such a human-oriented query interface into machine-oriented web services is an important problem. Toward this goal, we adopt the Wrapper technology successfully developed and deployed in Database community, and demonstrate how to generate Web Services components (e.g., WSDL, UDDI, SOAP) automatically. We present the overall architecture of our developed prototype and a few showcases based on real web sites.

Categories and Subject Descriptors

H.2.5 [Database Management]: Heterogeneous Databases—*Data translation*; D.2 [Software Engineering]: Miscellaneous

Keywords

Wrappers, Web Services, Translation

1. INTRODUCTION

Web service is a piece of XML-based software interface that can be invoked over the Internet, and can be roughly viewed as a next-generation successor of CORBA or RPC technique. Currently web services are often considered as one of the most important and vital build-

*The work was done when the author was at Penn State.

†This research was partially supported by IBM Eclipse Grant Award 2004.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'05, March 13–17, 2005, Santa Fe, New Mexico, USA
Copyright 2005 ACM 1-58113-964-0/05/0003 ...\$5.00.

ing blocks to fully achieve the vision of the “Semantic Web” [2]. To successfully and quickly migrate from human-centered web interfaces (e.g., web browsing and querying) to machine-centered web interfaces, the existence of tools for creating new web services or for migrating old interfaces to web services is a must.

Two popular approaches to creating web services are *top-down* and *bottom-up* approaches, as suggested in [14]. In the top-down approach, there is no previous interfaces. Developers first design the interface of a web service by creating, for instance, a WSDL file, and related SOAP specifications. And then developers finally implement the underlying business logic for that interface (e.g., by writing software pieces to connect to databases). On the other hand, in the bottom-up approach, an implementation of a business logic (e.g., legacy application or existing web application) has been operating successfully for a long time. Then, by examining such implementations, developers simply extract interfaces and package them according to web services standards.

As web services technologies mature, more number of novel applications will appear and they are likely to be created in the “top-down” approach; their web service is first carefully designed and then the corresponding implementation is to be carried on. However, we expect the majority of the initial stream of web services for the foreseeable future are likely to be legacy applications re-packaged as web services. For instance, Amazon.com has been providing keyword-based search interface (to human users) since its inception. In the early 2002, however, they released their first web service [1], by providing a web service interface that is linked to existing Amazon.com’s query engine or databases, following the bottom-up approach. This also has been the case for the Google [7]. To illustrate the motivation better, consider the following scenario.

Example 1. A company *UniversalBooks* is a start-up who aims at building the biggest bookstore on earth by virtually integrating all major, say 1,000, on-line bookstores. Some of the bookstores provide sophisticated web services interface to their catalog information (with nominal charges), while the majority still provide only web-based FORM search interface (which is free). That is, from *UniversalBooks*’s point of view, some are cooperative *machine-oriented* interfaces, while

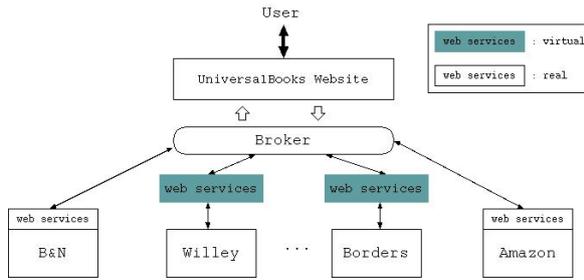


Figure 1: Overview of UniversalBooks.

others are non-cooperative *human-oriented* information sources. To tackle this problem, for each bookstore who does not have web services yet, **UniversalBooks** first builds a virtual web service that is wired to each public web-based FORM interface so that all 1,000 bookstores will have either real or virtual web service interfaces. This is possible, since regardless of their cooperativeness to **UniversalBooks**, all bookstores at least provide public web interface. Then, **UniversalBooks** builds a broker that provides a universal catalog system based on all those 1,000 web service interfaces. Now, customers can search books using **UniversalBooks**'s broker interface that amounts to concurrently searching books at 1,000 different bookstores. The key technique to the success of **UniversalBooks** is to be able to generate *virtual* web service interfaces for a large number of *non-cooperative* bookstores quickly. The example is illustrated in Figure 1. □

Therefore, as illustrated in Example 1, we believe the need for tools to help migrate existing applications (e.g., database interfaces, web-based query interface, etc) to web services is substantial and immediate. Such needs arise not only in commercial sectors but also in scientific communities. For instance, [16] describes an experience to build federated interfaces to astronomical data using web services. [9] is another example providing web services based interface to scientific digital library.

We especially observe that one of the most popular forms of public services on the current Web environment is web-based query interfaces, especially FORM-based interfaces in HTML. It ranges from a simple textbox where users type in a few keywords to a sophisticated query interface where users can select various input conditions. Many commercial web sites already provide such query interfaces to human users for various purposes. Therefore, when the needs for web services finally arrive at those commercial organizations, these *public* web-based query interfaces are likely to be the first candidate to be migrated to web services. Figure 2 is such an example from Amazon.com website.

As the popularity of web services increases drastically, new tools for web services started to appear. For instance, IBM's recently released WebSphere Studio Application Developer [10] supports various tasks such as discovering existing web services, composing existing web services into new web services, deploying web services, and testing and publishing web services. However,

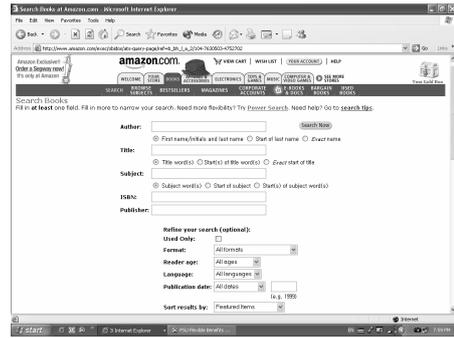


Figure 2: Amazon.com's FORM-based interface.

to our best knowledge, there is no tool that supports the "automatic" generation of web services from web sites. The core technique to achieve this goal is the ability to "mine" hidden information from HTML FORM-based query interface and "transform" the found knowledge into web services. When a large number of existing web query interfaces must be transformed into web services, such an automatic generation tool will greatly improve the productivity and decrease errors. To solve the problem, in this paper, we propose to utilize the *Wrapper* technologies developed in the context of heterogeneous data integration.

1.1 Background and Related Work

Web Services. *Web Services* pass content between applications using industry-wide standard format (XML), a public registry (UDDI), a template (WSDL), and a programmatic binding interface (SOAP) to enable applications to find and interact with each other over the Internet to transport information and data between co-operating applications [3].

1. *Universal Description, Discovery, and Integration* (UDDI) provides a mechanism for clients to find web services. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications.
2. *Web Services Description Language* (WSDL) defines services as collections of network endpoints or ports. A port is defined by associating a network address with a binding; a collection of ports define a service.
3. *Simple Object Access Protocol* (SOAP) is a message layout specification that defines a uniform way of passing XML-encoded data. It also defines a way to bind to HTTP as the underlying communication protocol. SOAP is a technology to allow for RPC over the Web.

Since these are all industry-wide standards, web services are gaining advantage above its traditional counterparts. We foresee web services will proliferate as simple compiled piece of code that are published on

the Internet for everyone to use. However, the key to the success of Web Services is on-the-fly software creation through the use of loosely coupled, reusable software components. Standards will mature with time and not all companies or organizations are going to embrace web services and integrate the web services module with their web sites. Until that time, the data “hidden” behind the web sites will remain unexplored. Hence a software application that automatically generates web service interface that utilizes the above standards is an immediate necessity.

Wrappers. Database community has successfully developed and deployed tools to integrate heterogeneous data from web sites. The core technique to such integration tools is the capability to (semi-)automatically generate a programmable interface to the FORM-based query interface of web sites. Often such programs wrapping the web site providing query interfaces are called “Wrappers”, while other client programs that call the query interfaces provided by wrappers are called “Mediators”. A few well-known research projects are TSIMMIS [4], Information Manifold [11], XWrap [15], Ariadne [12], etc. and a few commercial companies include Jungle, Nimble, Wizbang, and Fetch. For a better survey, refer to [5].

The wrapper is to extract data from a web site. The problem lies in the fact that web sites are usually *unstructured* HTML documents for human browsing, not for data extraction. Therefore, the wrapper research has developed various methods to automatically extract data from given web pages: (1) developing customized grammars to pin-point HTML data to be extracted (e.g., [8]), and (2) Machine-Learning based approach to learn extraction rules from a few given example HTML pages (e.g., [13]). Different wrappers use different algorithms and features and thus have different pros/cons. Since we merely use wrapper technique to achieve automatic generation of web services, using which wrapper technique is orthogonal to our proposal. In our prototype, we used XWrap¹ as the choice of the wrapper due to its availability to public. However, let us emphasize that replacing XWrap with other wrapper techniques does not affect the validity of our proposal.

Related work. To our best knowledge, the only work closely related to our proposal is the Proteus project² [6]. However, their focus is on integrating heterogeneous web services (as information sources) using mediator-based techniques, whereas our proposal is concerned on how to automatically generate web services for information sources. We believe two works complement each other well.

2. MAIN IDEAS

Figure 3 illustrates an overview of the system, named as Pollock, that we have developed. As shown, when a web site provides a web service layer (i.e., top portion),

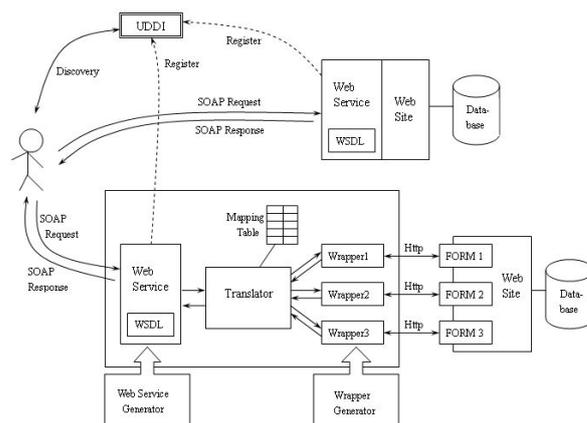


Figure 3: Overview of Pollock.

client program can directly interact with the web site via SOAP. However, when such a web service layer is not provided (i.e., bottom portion), a *virtual* web service is created as shown in the box of Figure. Note that from the client program’s point of view, both web sites appear to be providing web services in the same fashion. In a nutshell, the following steps are needed to generate virtual web services:

1. First, the *Wrapper Generator* (WG) generates wrappers using XWrap when a URL to a FORM-based interface is given. Depending on the characteristics of the given FORM interface, one to many wrappers can be generated. During the generation, WG retrieves specific information per each web site and stores them in the mapping table. Furthermore, the mapping table also keeps track of the bijective mappings between specific input sequence and wrappers.
2. Next, the *Web Service Generator* (WSG) generates a WSDL file using web site-related information such as input/output types or service names that are passed from the Translator and WG.
3. Since SOAP messages going through web service layer cannot be directly understood by wrappers, a middleware called the *Translator* must map SOAP-based XML message (i.e., SOAP Request) into HTTP-based URL message. In a reverse direction, data extracted from web page must be also translated into a proper SOAP message (i.e., SOAP Response).
4. Once a web service is successfully generated, the details of the web service need to be published into UDDI so that other clients can discover proper services.

3. DETAILS

In this section, we describe details of each step using an example web site, *Foodgeeks*, that we have successfully developed.

¹<http://disl.cc.gatech.edu/XWRAPelite/>

²<http://dblabb.usc.edu/WebServices>

3.1 Generating Wrappers

In this step, not only one has to generate wrappers for web sites, but also has to mine information needed to generate WSDL later. Especially, note that specific FORM controls (e.g., textbox, checkbox, pull-down menu, etc) will have different counterparts in WSDL `<types>` schema, and such a relationship must be carefully mined and stored for a later use. We have successfully generated wrappers and their corresponding web services for the eight web sites listed in Table 1.

- **String type.** FORM controls such as textbox or textarea can be easily captured as the string type in WSDL.
- **Enumeration type.** We found that most web sites provide a list of valid input values to avoid incorrect query string by users. For instance, **Bigfoot** provides a pull-down menu having 50 state names for users to choose.

A bit more complicated example is when a choice made by users and a value sent to web sites are different. For instance, when a user pick “California” from a pull-down menu, the actual query string to be sent is “CA”, not “California”. Such a knowledge must be also mined by parsing the source of the web page. **eBay**, **Wine**, and **Gamestop**, for instance, have such a case. Nevertheless, all these cases can be captured as the enumeration type in WSDL.

Third kind of enumeration type is for checkbox. For this case, we used the enumeration type in WSDL so that only two fixed values, “checked” and “unchecked”, are allowed. For instance, when a user checked the checkbox at **Vendio** site, it can be re-written to “&use_desc=on” string. Similarly, a SOAP message with the enumeration type “checked” would be re-written into the same string.

In addition to setting up inputs to web sites, another important task of wrappers is how to extract specific data from returned web pages. Much efforts of wrapper research has developed various ways to remedy this problem. Since such techniques are beyond the scope of this paper, we refer to [15] for details.

3.2 Generating WSDL

The WSDL is a specification describing web services, and has several components such as type, message, and portType. The *type* is the definition of special data types that are used in WSDL. The *message* defines a service provided by a web service provider and the definition has a name of each service and its parameters. For each service, there is a pair of messages – request and response. The *portType* ties a request message and a response message as one operation.

For illustration, consider the “Advanced Recipe Search” FORM interface at the **Foodgeeks** web site shown in Figure 4.

1. **<message> element.** Web service of “Advanced Recipe Search” is defined here. For the service,

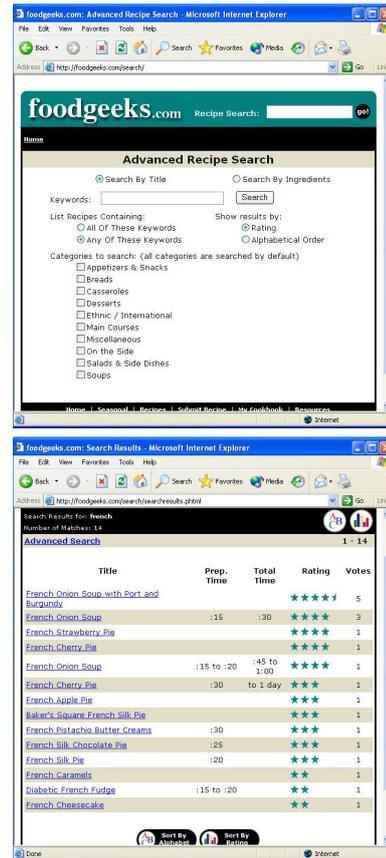


Figure 4: FORM interface and result page at Foodgeeks.com.

two messages are involved – *FoodgeekSearch* and *FoodgeekSearchResponse*. *FoodgeekSearch* requires five input parameters, where each of them is a way to search recipe, keywords, a way to search with keywords, a way to show results, and categories to search. Each input parameter is described with its own name and data type as one `<part>` element inside the `<message>` definition. The snippet of the WSDL generated for **Foodgeeks** is shown in Table 2. Similarly, the message for *FoodgeekSearchResponse* is defined according to the search result page of Figure 4.

2. **<types> element.** In Table 2, one can find several types: a built-in type such as *string*, and several special types such as *searchType* and *categoryType*. For instance, *categoryType* limits its legal values which are defined as enumerations of simple type. Pull-down menu, checkbox, and radio button of a web site are typically defined in this way.
3. **<portType> element.** Two corresponding messages, *FoodgeeksSearch*, as input, and *FoodgeeksSearchResponse*, as output, are combined into a service.

3.3 Handling SOAP messages

Web Site	URL to a FORM interface	Characteristics
eBay	http://half.ebay.com/	Single input box
BigFoot	http://www.bigfoot.com/	Four input boxes
Wine	http://www.wine.com/search/default.asp?ct=1631	Pull-down menu
Gamestop	http://www.gamestop.com/search.asp?	Pull-down menu
Yahoo Real Estate	http://list.realestate.yahoo.com/	Radiobutton and sorting
Monstor	http://jobsearch.monster.com/	Multi-select list
Vendio	http://srch.vendio.com/usearch/	Single checkbox
Foodgeeks	http://foodgeeks.com/search/	Multiple checkboxes

Table 1: Example web sites.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
<types>
<xsd:simpleType name="categoryType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Appetizers and Snacks"/>
<xsd:enumeration value="Breads"/>
<xsd:enumeration value="Casseroles"/>
<xsd:enumeration value="Desserts"/>
<xsd:enumeration value="Ethnic or International"/>
<xsd:enumeration value="Main Courses"/>
<xsd:enumeration value="Miscellaneous"/>
<xsd:enumeration value="On the Side"/>
<xsd:enumeration value="Salads and Side Dishes"/>
<xsd:enumeration value="Soups"/>
<xsd:enumeration value="default"/>
</xsd:restriction>
</xsd:simpleType>
</types>
<message name="FoodgeeksSearch">
<part name="Search_by_TITLE_or_INGREDIENTS"
type="typens:searchType"/>
<part name="keywords" type="xsd:string"/>
<part name="List_Recipes_Containing_ALL_or_ANY_of"
type="typens:listType"/>
<part name="Show_Results_by_RATING_or_ALPHABETICAL_ORDER"
type="typens:showresultsType"/>
<part name="Categories" type="typens:categoryType"/>
</message>
<portType name="FoodgeeksPort">
<operation name="FoodgeeksSearch">
<input message="typens:FoodgeeksSearch"/>
<output message="typens:FoodgeeksSearchResponse"/>
</operation>
</portType>
</definitions>
```

Table 2: The WSDL generated from foodgeeks's FORM interface.

Handling SOAP messages is an interaction among three parties – users, virtual web service layer, and wrappers. Since the web sites are hidden by the wrappers, users are not aware of its existence. Once a user locates a web service from UDDI, he/she submits a SOAP request message to the virtual web service layer. Typically, a user is a computer program acting as the client of the web service. To the user, whether the target web service is a real or virtual is irrelevant.

The intermediate web service layer then extracts input parameters from the incoming SOAP request message, and re-writes them into a string message (i.e., HTML POST or GET message) that can be interpreted by wrappers. Further, when results are returned from wrappers, they are packaged as a SOAP response message and sent back to users. The knowledge as to how to translate is pre-determined by wrappers and stored in the Translator. When multiple wrappers exist for a given web service, a mapping table has information to determine which wrapper must be invoked for a specific service name or parameters, etc. Once the wrapper gets result pages from a web site, it extracts only required data that was specified by the original SOAP request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Body>
<ns1:FoodgeeksSearch
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org
/soap/encoding/"
xmlns:ns1="urn:Foodgeeks">
<Search_by_TITLE_or_INGREDIENTS
xsi:type="typens:searchType">
title</Search_by_TITLE_or_INGREDIENTS>
<keywords xsi:type="xsd:string">french</keywords>
<List_Recipes_Containing_ALL_or_ANY_of
xsi:type="typens:listType">
any</List_Recipes_Containing_ALL_or_ANY_of>
<Show_Results_by_RATING_or_ALPHABETICAL_ORDER
xsi:type="typens:showresultsType">
rating</Show_Results_by_RATING_or_ALPHABETICAL_ORDER>
<Categories xsi:type="typens:categoryType">
desserts soups</Categories>
</ns1:FoodgeeksSearch>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Table 3: SOAP request message.

message.

For instance, If a user chooses input parameters as shown in Figure 4, the corresponding SOAP request message that a user sends to the intermediate web service layer is shown in Table 3. Then, the web service layer parses the SOAP request message, and passes to wrappers input parameters like “**title french any rating desserts+soups**”. Some of these values should be changed to real values that are used internally in the web site, and automatically taken care of by wrappers. For this example, the real values that are passed from wrappers to a web site are “**title french OR rating+DESC&category=9&category=17**”. After a wrapper extracts results from the result page, it again passes them to the intermediate web service layer, where the results are re-written into a SOAP response message format and returned to the user. An example SOAP response message is shown in Table 4.

3.4 Publishing to UDDI

To publish the newly generated web service to UDDI, The four data elements of UDDI data model are needed: The *businessEntity* describes the business that provides a web service. It includes business name, business description, contact information and so on. The actual information about the web service is represented in the *businessService*. The *bindingTemplate* indicates how and where a user can access the web service. Finally, the *tModel* represents a description and a pointer to external technical specifications. The address receiving

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:FoodgeeksSearchResponse
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org
      /soap/encoding/" xmlns:ns1="urn:Foodgeeks">
      <return>
        <resultElements
          soapenc:arrayType="ns1:ResultElement[4]"
          xsi:type="soapenc:Array">
          <item xsi:type="ns1:ResultElement">
            <title xsi:type="xsd:string">French Onion Soup
              with Port and Burgundy</title>
          </item>
          <item xsi:type="ns1:ResultElement">
            <title xsi:type="xsd:string">French Onion Soup</>
          </item>
          <item xsi:type="ns1:ResultElement">
            <title xsi:type="xsd:string">French Onion Soup</>
          </item>
          <item xsi:type="ns1:ResultElement">
            <title xsi:type="xsd:string">French Cherry Pie</>
          </item>
        </resultElements>
      </return>
    </ns1:FoodgeeksSearchResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Table 4: SOAP response message.

SOAP messages from users in businessService element and the address having WSDL file of a web service in tModel need to be our SOAP server and the address where we store WSDL files. However, other information for UDDI data elements would be entered manually. Among two ways to save the UDDI data (i.e., web-based interface or programmatic interface), we used the programmatic interface and UDDI4J (UDDI for Java).

4. CONCLUSION AND FUTURE WORK

In this paper, we have presented a preliminary work on automatically generating web services from FORM-based query interfaces of web sites. Given the explosive adoption of web services and its promise as a building block toward the “Semantic Web”, we believe such a technique is timely and necessary. For this goal, by using Wrapper techniques developed in Database community, we demonstrated how to build web services automatically.

We are currently looking into several improvements: (1) During the generation of web services, there are several places where user’s intervention could be beneficial. To support such a case, a GUI tool is planned to be developed. Ideally, this tool must be integrated with a GUI tool for wrapper generation as well; and (2) Once a WSDL file is generated, it is much more convenient for the end users to have a client library in various programming languages (e.g., Java, C++) for accessing the web services defined in the WSDL file. There are many small details that need to be worked out in order to fully automatically generate this kind of library from the input web site and WSDL file. Further, an example client program can be generated in synchronization with the client library to help users.

5. REFERENCES

[1] Amazon. “Amazon.com Web Services 2.0”. Web page, 2002.

<http://www.amazon.com/gp/aws/landing.html>.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. “The Semantic Web”. *Scientific American*, May 2001. <http://www.sciam.com/issue.cfm?issuedate=May-01>.

[3] J. Clabby. “*Web Services Explained, Solutions and Applications for the Real World*”. Prentice-Hall, 2003.

[4] H. G. et al. “The TSIMMIS approach to mediation: Data models and Languages”. *J. Intelligent Information Systems (JIIS)*, 8(2):117–132, Mar. 1997.

[5] D. Florescu, A. Y. Levy, and A. Mendelzon. “Database Techniques for the World-Wide Web: A Survey”. *ACM SIGMOD Record*, 27(3):59–74, Sep. 1998.

[6] S. Ghandeharizadeh, C. A. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J. L. Ambite, M. Cai, C.-C. Chen, P. Pol, R. Schmidt, S. Song, S. Thakkar, and R. Zhou. “Proteus: A System for Dynamically Composing and Intelligently Executing Web Services”. In *Int’l Conf. on Web Services (ICWS)*, Las Vegas, CA, Jun. 2003.

[7] Google. “Google Web APIs”. Web page, 2002. <http://www.google.com/apis/>.

[8] J.-R. Gruser, M. E. V. L. Raschid, and L. Bright. “Wrapper Generation for Web Accessible Data Sources”. In *Int’l Conf. on Cooperative Information Systems (CoopIS)*, 1998.

[9] Y. Hong, B.-W. On, and D. Lee. “System Support for Name Authority Control Problem in Digital Libraries: OpenDBLP Approach”. In *European Conf. on Digital Libraries (ECDL)*, Bath, UK, Sep. 2004.

[10] IBM. “IBM WebSphere Studio”. Web page, 2002. <http://www-3.ibm.com/software/webservers/studio/index.html>.

[11] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. “The Information Manifold”. In *AAAI Symp. on Information Gathering*, 1995.

[12] C. A. Knoblock and S. Minton. “The Ariadne Approach to Web-based Information Integration”. *IEEE Intelligent System*, 13(5), 1998.

[13] N. Kushmerick, R. Doorenbos, and D. Weld. “Wrapper Induction for Information Extraction”. In *AAAI IJCAI*, 1995.

[14] C. Lau and A. Ryman. “Developing XML Web services with WebSphere Studio Application Developer”. *IBM Systems J.*, 41(2):178–197, 2002.

[15] L. Liu, C. Pu, and W. Han. “XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources”. In *IEEE ICDE*, San Diego, CA, Feb. 2000.

[16] T. Malik, A. S. Szalay, T. Budavari, and A. R. Thakar. “SqkQuery: A Web Service Approach to Federate Databases”. In *Biennial Conference on Innovative Data Systems Reserch (CIDR)*, Asilomar, CA, Jan. 2003.