DB2 and Web services

by S. Malaika
C. J. Nelin
R. Qu
B. Reinwald
D. C. Wolfson

The World Wide Web offers a tremendous amount of information. Accessing and integrating the available information is a challenge. "Screen scraping" and reverse template engineering are manual and error-prone integration techniques from the past. Simple Object Access Protocol (SOAP) from the World Wide Web Consortium (W3C) allowed Web sites to become programmable Web services. W3C SOAP is based on XML (Extensible Markup Language) and is a lightweight protocol that provides a service-oriented architecture for applications on the Web. Clients compose requests and send SOAP envelopes to providers, who reply through SOAP responses. In this paper, we describe DB2® and Web services, with techniques for integrating information from multiple Web service providers and exposing the collective information through Web services.

Web services are part of an emerging technology that offers the dual promise of simplicity and pervasiveness. Much of the simplicity is due to the common XML (Extensible Markup Language) foundation that underlies most Web service protocols. Web services provide a ubiquitous model for offering business services over the Internet as well as within organizations. Web services are of particular interest for their ability to incorporate third-party applications or legacy applications.

In the most primitive sense, Web services can be viewed as any mechanism by which an application service may be provided to other applications on the Internet.[1] Web services are described in WSDL (Web Services Description Language).[2] The WSDL description may be registered in the UDDI (Universal Description, Discovery, and Integration) repository. UDDI provides a set of application programming interfaces (APIs) to register and search for Web services. IBM, as one of the founders of UDDI.org,[3] has provided a publicly available UDDI implementation using Database 2* (DB2*) and WebSphere*.[4]

Web services may be informational or transactional. That is, some services will provide information of interest to the requestor, whereas other services may actually lead to the invocation of business processes. Informational Web services available today range from simple weather or stock-quote services to the access of nucleotide sequence data or corporate information. Transactional Web services are being defined by organizations such as ebXML[5] and XML.org[6] to facilitate standardization of business-to-business processes.

This paper explains why Web services are important to DB2 and how DB2 is being extended to provide optimized support for Web services. DB2 users may take advantage of Web services in two ways: as a provider and as a requestor. The next section describes an application scenario and demonstrates how Web services can be used in a database environment. The following section gives a brief overview of the Web

0018-8670/02/$5.00 © 2002 IBM

services framework. The next section describes DB2 as a Web service requestor and explains how Web services are incorporated into SQL (Structured Query Language). The following section demonstrates how DB2 provides Web services, and the last section describes DB2 tools that can be used to both provide and request Web services.

Application scenario

Our application scenario describes a manufacturing company that works with a number of suppliers around the world. As a technically innovative company looking for cost-effective and speedy business processes, it works with key suppliers (several hundred around the world) to implement three capabilities that allow users to get price quotes for proposed purchases, to make a purchase, and to check on the order status. By implementing these capabilities as Web services, users in our company are able to call these services from a wide variety of platforms and application environments, while suppliers (as providers of the Web services) are free to implement the services in any manner they choose. Internal purchasing applications can use these Web services in a variety of ways. For instance, a simple purchasing application may use a private UDDI registry to look up a supplier in a user-specified city to get a price quote on a product. The UDDI search finds a supplier that offers the product and returns a link to the supplier's Web service operations. The purchasing application can then issue a Web service request to obtain a price quote from this supplier, provide it to a buyer, and allow the buyer to place the order. The application can then be used to query the order status by invoking the appropriate operation for the supplier. So to summarize, each supplier offers three Web service operations that can be expressed with the following abstract signatures:

- getQuote (in String partNum, in Integer qty, in Date desiredDate, out Decimal price, out String currency, out Date proposedDate)
- purchase (in String partNum, in Integer qty, out String poNumber, out Date commitDate)
- getStatus (in String poNum, out String status)

In the next two sections we discuss how these Web services can be invoked by DB2 applications, and how they may be implemented by the Web Services Object Run-time Framework (WORF) provided by DB2. A later section describes WORF in detail.

Table 1 Table for purchase_orders

supplier	po_num	date	part_num	qty
ASupplier	12345	3/20/02	A4	34
BSupplier	12347	6/20/02	C7	43
CSupplier	34656	5/04/02	D7	3

Web service exploitation. The basic scenario, so far, has shown how an application can find and work with a single supplier. However, we often need to perform these operations over sets. We discuss two examples: first, to find the best quote from a set of suppliers, and second, to report on the order status for all overdue orders. The remainder of this section illustrates how such set-oriented operations can easily be implemented by using Web services within database queries.

DB2 allows users to define new functions that may be invoked from SQL, thus extending the SQL language. These user-defined functions (UDFs) may be used for many purposes: calculations, transformations, or even to send messages.[3] Using this facility, we can define a new function, GET_STATUS, to perform the getStatus operation:

```
varchar(20) GET_STATUS (url varchar(80),
                        po_num varchar(20))
```

Here the return value is the purchase order (PO) status, and the input parameters are the URL (uniform resource locator) to which the request is to be sent and the identity of the purchase order in question. To find the status of a specific purchase order, say 12345, from a supplier that offers this service at http://www.Asupplier.com/getStatus, we could issue the following SQL statement:

```
values GET_STATUS (
        'http://www.Asupplier.com/getStatus','12345')
```

To continue our example, Table 1 shows outstanding purchase orders and Table 2 contains information about the Web service operations each supplier offers. Note that Table 2 could have been populated in advance by queries to UDDI, or it could be replaced by calls to UDDI. We choose to represent the information in this way to simplify the example. To obtain the status of all outstanding purchase orders from ASupplier, we could say:

Table 2 Table for supplier_ops

supplier	operation	url
ASupplier	getStatus	http://www.Asupplier.com/getStatus
ASupplier	getQuote	http://www.Asupplier.com/getQuote
BSupplier	getQuote	http://www.Bsupplier.com/services/getQuote
BSupplier	getStatus	http://www.Bsupplier.com/services/getStatus

Table 3 Parameters for the getQuote Web service

	Name	Type
Input	partNum	string
	qty	integer
	desiredDate	date
Output	price	decimal
	currency	string
	proposedDate	date

Table 4 Parameters for the GET_QUOTE table function

	Name	Type
Input	supplier	varchar(30)
	url	varchar(80)
	part_num	varchar(20)
	qty	integer
	desired_date	date
Output	supplier	varchar(30)
	url	varchar(80)
	part_num	varchar(20)
	qty	integer
	desired_date	date
	price	decimal
	currency	varchar(10)
	proposed_date	date

```
SELECT supplier, po_num,
GET_STATUS ('http://www.ASupplier.com/getStatus',
        po_num) AS po_status
FROM purchase_orders
WHERE supplier = 'ASupplier'
```

In this simple example, we explicitly state the address of the service to be invoked. To find the status of all outstanding purchase orders from suppliers who offer a Web service interface, we could issue the following query:

```
SELECT p.supplier, p.po_num,
        GET_STATUS (s.url, p.po_num) AS po_status
FROM purchase_orders p, supplier_ops s
WHERE p.supplier = s.supplier
AND s.operation = 'getStatus'
```

If this query is commonly issued, it might be convenient to define a view to provide a simpler interface. The definition of this view would be

```
CREATE VIEW order_status AS SELECT p.supplier, p.po_num,
                GET_STATUS (s.url, p.po_num) AS po_status
FROM purchase_orders p, supplier_ops s
WHERE p.supplier = s.supplier
AND s.operation = 'getStatus'
```

To get the status the following simple query could then be used:

```
SELECT *
FROM order_status
```

This query could, of course, be extended to exploit features of SQL. For instance, to sort the result by supplier, we simply append an *order by* clause, such as:

```
SELECT po_num, supplier, status
FROM order_status
ORDER BY supplier
```

All the examples so far show how a Web service that returns a single value can be integrated with DB2 SQL, but we may need to handle multiple return values. The signature for the getQuote Web service is shown in Table 3. In order to access it from DB2 we turn this service into a DB2 table function with input and output parameters as shown in Table 4.

To provide more meaningful context, our table function includes as outputs all of the interesting input parameters. The GET_QUOTE table function is invoked within a query such as

```
SELECT *
FROM TABLE (GET_QUOTE ('ASupplier',
                'http://www.Asupplier.com/getQuote,
                '52435FFA',25, '7/1/2001')) AS t
```

This statement returns a table containing a single row with the response from this supplier. In order to deal with suppliers in other countries, our GET_QUOTE function contains currency units. To convert the price to dollars, we could try to maintain a table of currency conversion data manually. Given the volatile nature of foreign exchange, it would be better to invoke another Web service, perhaps provided by a foreign-exchange trading firm, to perform the conversion using the most current data. Input and output parameters for the DB2 function to invoke this service are shown in Table 5.

Table 5 Parameters for the TO_DOLLARS user-defined function

	Name	Type
Input	currency	varchar(10)
	amount	decimal
Output	amount	decimal

Using this additional service, we can now get a more accurate quote with a query such as

```
SELECT t.supplier, t.part_num, t.qty,
        (t.desired_date – t.proposed_date) AS timeliness,
TO_DOLLARS (t.currency, t.price) AS cost
FROM TABLE (GET_QUOTE ('ASupplier',
'http://www.Asupplier.com/getQuote, '52435FFA',
        25, '7/1/2001')) AS t
```

Here we make the columns explicit and, using the power of SQL, define an output column, "timeliness," to reflect the difference between our desired date and the date proposed by the supplier for the part. We also use the currency conversion Web service to convert the quoted price to United States currency. This query returns a single row with the quote from a single vendor for a single part. Now, consider the case where we require quotes for a list of parts. We define a table, needed_parts, shown in Table 6.

Table 6 Table and data for needed_parts

part_num	qty	desired_date
34dsaf	20	7/1/2001
35gfds	34	8/1/2001
809gds	10	6/30/2001

To get quotes on all of these parts from our supplier we can issue

```
SELECT t.supplier, n.part_num, n.qty,
        (n.desired_date – t.proposed_date) AS timeliness,
TO_DOLLARS (t.currency, t.price)
FROM needed_parts n, TABLE (GET_QUOTE ('ASupplier',
        'http://www.Asupplier.com/getQuote', n.part_num,
        n.qty, n.desired_date)) t
```

This query returns a table of quotes for each part listed in the needed_parts table from one supplier. To get quotes for each of our suppliers we can issue the following query:

```
SELECT n.part_num, t.supplier, n.qty,
        (n.desired_date – t.proposed_date) AS timeliness,
        TO_DOLLARS (t.currency, t.price)
FROM needed_parts n, supplier_ops s,
        TABLE (GET_QUOTE (s.supplier, s.URL,
                                n.part_num, n.qty, n.desired_date)) t
WHERE s.operation = 'GET_QUOTE'
ORDER by n.part_num, timeliness
```

This query generates quotes for all the needed parts from all the suppliers that offer the getQuote Web service and returns a table of these quotes ordered by part number and timeliness. The queries use very powerful, yet simple, standard DB2 SQL.

Finally, our company may want to expose this query as a Web service itself so that its purchasing agents can invoke the query from any location where they have access to the Internet. DB2 7.2 provides a simple mechanism that allows Web services to be created in support of such queries. (See http://www.software.ibm.com/data/webservices for information about Web service access to DB2 data and stored procedures.)

These examples show how Web services can be exploited within a database. By invoking Web services as UDFs, we can take advantage the full power of SQL to perform queries across combinations of Web services and persistent data.

Providing Web services. Web services insulate users of the service from its implementation. The three services offered by suppliers in our scenario can be implemented in any manner that fulfills the contract defined by the service. Some might use the Java** 2 Platform, Extended Edition (J2EE**) programming model implemented by the IBM WebSphere Application Server. Others might use competing J2EE or other architectures, such as .NET from Microsoft.

Table 7 Table and data for orders

po_num	customer	part_num	qty	commit_date	status
78453	yourMfgCo	A4	23	5/23/02	ONTIME
12347	myMfgCo	C7	200	6/20/02	ONTIME
53456	theirMfgCo	B12	5	4/23/02	COMPLETE
35335	yourCo	A3	7	4/15/02	SHIPPED

In this paper we discuss how DB2 data can be easily accessed using Web services. DB2 provides the Web Services Object Runtime Framework (WORF) facility that can be used in conjunction with the WebSphere Application Server to perform SQL queries, utilize DB2 XML Extender routines to manipulate XML data, and invoke stored procedures. Within our scenario, a supplier could use this facility to implement the three Web services just described. Assume that a supplier has the table, orders, shown in Table 7.

The getStatus service could be implemented by effectively wrapping a Web service around the query

```
SELECT status FROM orders WHERE po_num = :input
```

In this query the parameter *input* would be provided within the Web service request and the WORF runtime code would execute the query and return the result (status) in the Web service response. In a later section we discuss the details of the WORF facility.

Web service framework

SOAP (Simple Object Access Protocol) is a lightweight protocol for the exchange of information in a distributed environment. A Web service is described in Web Services Description Language and can be accessed via a standard protocol, such as SOAP, over HTTP (HyperText Transfer Protocol). Other bindings, such as RMI (Remote Method Invocation) over IIOP (Internet Inter-ORB [Object Request Broker] Protocol) or SOAP over WebSphere MQ (Message Queuing), can also be supported. SOAP started as an XML-based RPC (Remote Procedure Call) mechanism with a request/response message-exchange pattern. The exchange of structured, typed values is supported through XML schema. SOAP is transport-protocol independent; however, the current standard defines bindings only for HTTP. DB2's Web service support is currently based on SOAP over HTTP.

A service requestor sends a SOAP envelope to a service provider. A SOAP request envelope either contains a serialized representation of an RPC method call or is structured for sending an XML document to a service provider. The service provider acts on the request and sends back a SOAP response envelope. Input and output parameters are described in XML schema.

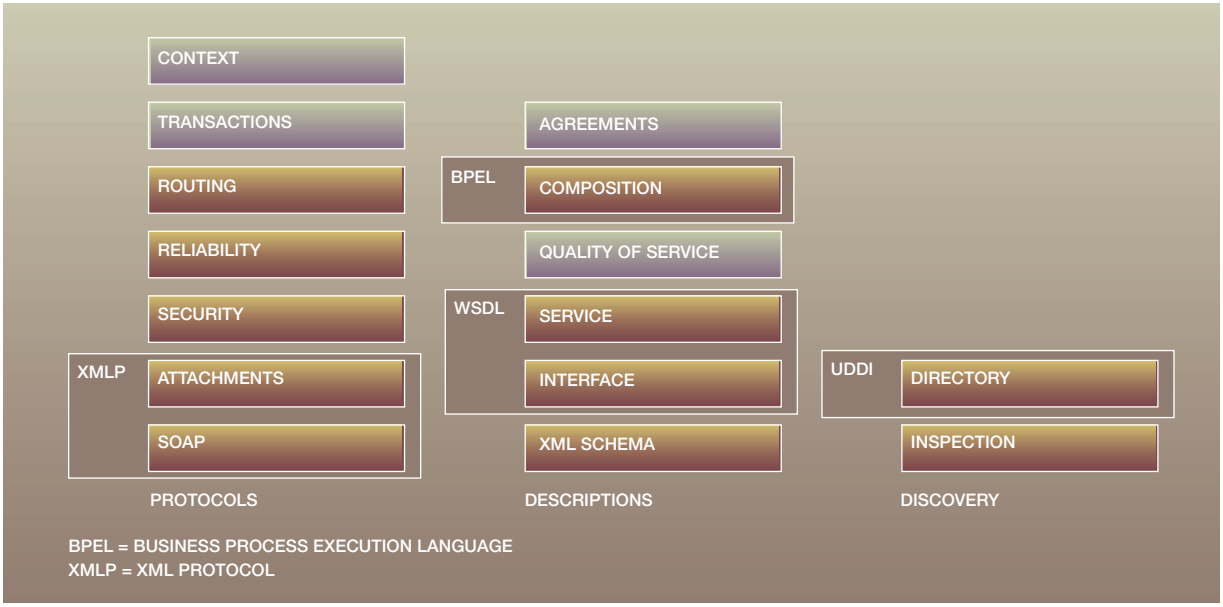
SOAP provides the protocol part of the Web services framework (Figure 1). Additional components are being defined and built on top of SOAP, such as security and transactions. WSDL (Web Services Description Language) is an XML vocabulary used to describe the interface of a Web service, including the input and output message format in terms of XML, the protocol binding and encoding, and the service endpoint. Web services can be published and advertised in a public UDDI registry. Publishing Web services in a public registry allows client applications to discover and dynamically bind to Web services.

SOAP is rightfully seen as the base for Web application-to-application interoperability. The fast availability of SOAP implementations, combined with wide industry backing, contributed to the quick adoption of SOAP.

DB2 as a Web service requestor

A database is a powerful vehicle for information integration. The ability to pull information from a variety of service providers puts databases in a unique position to analyze and combine information and to provide powerful querying capabilities. As we discussed in an earlier section, we want to make the use of Web services a natural extension to the DB2 SQL environment. To achieve this we must address two sets of problems. First, the signature of the UDF must be mapped to the signature of the Web service it implements. Then these data must be used to construct and send the SOAP message to the indicated service provider. After the response is re-

Figure 1 Web service framework



ceived, the reply must be decomposed into the set of result parameters that the user expects. Our implementation architecture uses two layers of functions: a set of SOAP UDFs that are specific for each WSDL operation and a set of underlying functions that actually perform the Web service invocation. The following subsections describe the design of these functions in further detail.

Web service concept in SQL. For a service requestor to send an invocation to a service provider, the following information is necessary:

- The URI (uniform resource identifier) of the target object, including optional header information, such as SOAP action
- The name of an operation to execute, including its input and output message format
- Binding information with respect to transport protocol, encoding style, name spaces, etc.

The abstract interface (operations and messages), the protocol bindings, and the access ports for deployed services are described in WSDL. Figure 2 shows the WSDL description of a sample Web service that returns the current stock quote for a given stock symbol.

Figure 3 shows a SOAP request envelope. The request is submitted through HTTP to a service endpoint as specified in the HTTP header. The SOAP body shows the method name and the name space for the method, as well as the input parameters. Figure 4 shows the SOAP response envelope. The response is returned through HTTP to the requestor.

At a conceptual level, the deployed Web service shown in Figure 2 has an abstract interface, such as

```
float stockQuote (symbol string)
```

SQL is extensible through functions, which may be either built-in or user-defined. Functions accept input parameters and return scalar values for the case of scalar functions, or entire tables for the case of table functions. SQL functions provide the necessary language hooks for using Web services. An SQL function for a Web service acts as a SOAP requestor, and we call these functions SOAP UDFs. SOAP UDFs compose SOAP requests, submit the request to the provider, receive the response, and return the response to the SQL engine. For composing and sending the SOAP request, the SQL function needs the service endpoint (URL of the service provider), name of the

Figure 2 WSDL description of simple stock quote Web service, getStockQuote

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="StockQuoteServiceRemoteInterface"
      targetNamespace="http://www.stockquoteservice.com/definitions/StockQuoteServiceRemoteInterface"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:tns="http://www.stockquoteservice.com/definitions/StockQuoteServiceRemoteInterface"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
 <message name="getQuoteRequest">
  <part name="symbol" type="xsd:string"/>
 </message>
 <message name="getQuoteResponse">
  <part name="result" type="xsd:float"/>
 </message>
 <portType name="StockQuoteServiceJavaPortType">
  <operation name="stockQuote">
   <input name="getQuoteRequest" message="tns:getQuoteRequest"/>
   <output name="getQuoteResponse" message="tns:getQuoteResponse"/>
  </operation>
 </portType>
 <binding name="StockQuoteServiceBinding" type="tns:StockQuoteServiceJavaPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="stockQuote">
   <soap:operation soapAction="" style="rpc"/>
   <input name="getQuoteRequest">
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://tempuri.org/StockQuoteService"/>
   </input>
   <output name="getQuoteResponse">
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://tempuri.org/StockQuoteService"/>
   </output>
  </operation>
 </binding>
<service name="StockQuoteServiceService">
   <port name="StockQuoteServicePort" binding="binding:StockQuoteServiceBinding">
    <soap:address location="http://localhost:8080/soap/servlet/rpcrouter"/>
   </port>
 </service>
</definitions>
```

method (SQL functions and SOAP methods might have different names), and name space, as well as the input and output parameters. All these parameters are provided in the WSDL description of the Web service. The abstract interface for the stockQuote Web service just shown could be registered as the SQL function

```
CREATE FUNCTION STOCK_QUOTE (symbol char(3))
        RETURNS double;
```

The implementation of the SQL function STOCK_QUOTE generates a SOAP request envelope as shown in Figure 3, sends the request to the service provider, receives the response as shown in Figure 4, retrieves the stock quote return value, and returns it as a function result.

An SQL statement that combines the results from calling the stockQuote Web service with data from the stock_watch table is

```
SELECT name, symbol, STOCK_QUOTE (symbol) AS quote
FROM stock_watch
```

The results from execution of this SQL statement are shown in Table 8.

Figure 3 SOAP request envelope over HTTP

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost:8080
Connection: Keep-Alive
Content-Type: text/xml
SOAPAction: ""
Content-Length: 393
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <SOAP-ENV:Body>
  <ns:getQuoteRequest xmln:ns="http://tempuri.org/StockQuoteService">
   <symbol xsi:type="xsd:string">IBM</symbol>
  </ns:getQuoteRequest>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4 SOAP response envelope over HTTP

```
HTTP/1.0 200 OK
Date: Fri, 15 Mar 2002 00:19:47 GMT
Status: 200
Content-Type: text/xml; charset=utf-8
Servlet-Engine: WebSphere Application Server (JSP 1.1; Servlet 2.2; Java 1.3.0; Linux 2.4.7-10smp
x86; java.vendor=IBM Corporation)
Content-Length: 465
Set-Cookie: JSESSIONID=JvGWBVyjLplbVdPzNG92M04d;Path=/soap
Server: WebSphere

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
 <ns1:getQuoteResponse xmlns:ns1="http://tempuri.org/StockQuoteService"
       SOAP-ENV:encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/">
   <return xsi:type="xsd:float">57.0</return>
  </ns1:getQuoteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web services may require complex input parameters and generate complex XML output. A database application might directly feed XML input to the function, retrieve the entire XML output, and process it in native XML. Other applications may require a simpler function interface that provides basic input parameters and returns basic output parameters. The

Table 8 Results from execution of SQL statement

name	symbol	quote
International Business Machines	IBM	114.35
Microsoft	MSFT	70.58

Table 9 Results from SQL query

symbol	quote
IBM	<stock_quotes> <stock_quote> <last>114.35</last> <ask>N/A</ask> <bid>110.00</bid> <change>+1.35000002</change> <pctchange>+1.19%</pctchange> <symbol>IBM</symbol> <time>4:01PM</time> </stock_quote> </stock_quotes>
MSFT	<stock_quotes> <stock_quote> <last>70.59</last> <ask>70.54</ask> <bid>70.31</bid> <change>−2.41000366</change> <pctchange>−3.30%</pctchange> <symbol>MSFT</symbol> <time>4:20PM</time> </stock_quote> </stock_quotes>

complexity of constructing complex XML input from basic SQL input parameters, as well as decomposing complex XML output into basic SQL output parameters, may be hidden in wrapper functions. The following subsection describes the various options.

SOAP UDFs. We now introduce various options for SOAP UDFs. A database application may more or less directly invoke a service provider with the body of the SOAP request envelope as input, receiving the body of the SOAP response envelope as output. The body of the SOAP request envelope is usually constructed from SQL values using SQL/XML publishing functions and XML data types.[4] Depending on the application, some values of the body of the SOAP response envelope may be extracted as SQL values for further processing. The logic for constructing a SOAP request and extracting SQL values from SOAP responses may be extensive. Hiding this logic in SQL-bodied wrapper functions improves usability but reduces flexibility. In this subsection, we describe the various options and discuss their trade-offs.

We use a stock quote example for demonstration purposes. The stock quote example uses a "ticker" symbol as input and returns a structured value as output. The input message

```
<stockticker>
  <symbol>IBM</symbol>
</stockticker>
```

might result in the output message

```
<stock_quotes>
  <stock_quote>
    <last> 113.40 </last>
    <ask> 113.50 </ask>
    <bid> 113.26 </bid>
    <change> +0.310005188 </change>
    <pctchange> +0.27% </pctchange>
    <symbol> IBM </symbol>
    <time> 1:38 PM </time>
  </stock_quote>
</stock_quotes>
```

Option 1: XML input and output. The user of the SOAP UDF provides the input parameters in XML format, and the SOAP UDF returns the service provider output in XML format. Using WSDL, we can register an SQL-bodied SOAP UDF, which sends a stock-ticker XML fragment in a SOAP envelope to the service provider and returns the stock quote as an XML fragment, as follows:

Figure 5 STOCK_QUOTE3 function definition

```
CREATE FUNCTION STOCK_QUOTE3 (stockticker varchar(5))
 RETURNS table
              (last    varchar(8),
       ask       varchar(8),
       bid       varchar(8),
       change    varchar(15),
       pctchange  varchar(10),
       symbol    varchar(5),
       time     varchar(20))
 LANGUAGE SQL READS SQL DATA
 EXTERNAL ACTION NOT DETERMINISTIC
 RETURN
  WITH
--1. Perform type conversions and prepare SQL input parameters --
--   for SOAP envelope
    soap_input (in)
        AS
     (VALUES XMLElement(NAME “getRTQuote”,
                XMLElement(NAME “stockticker”,
                    XMLElement(NAME “symbol”, stockticker)))),
--2. Submit SOAP request with input parameter and receive SOAP --
--   response
    soap_output (out)
        AS
   (values soaphttp (
    ‘http://localhost:8080/soap/servlet/rpcrouter’
     (SELECT in FROM soap_input)))
--3. Shred SOAP response and perform type conversions to get SQL
--   output parameters
     SELECT x.last, x.ask, x.bid, x.change, x.pctchange,
          x.symbol, x.time
       FROM Table (TableEXTRACT (
            (select out from soap_output),
            ‘/stock_quotes/stock_quote’,
            ‘./last’, ‘./ask’, ‘./bid’, ‘./change’,
            ‘./pctchange’, ‘./symbol’, ‘./time’)
         AS x (last AS varchar(8), ask AS varchar(8),
               bid AS varchar(8), change AS varchar(15),
               pctchange AS varchar(10), symbol AS varchar(5),
               time AS varchar(20));
```

Table 10 Results from SQL query

symbol	last	ask	bid	change	pctchange	time
IBM	114.35	114.35	111.04	+1.35000002	1.19%	4:01PM
MSFT	70.58	70.58	70.55	−2.41999817	3.32%	6:21PM

```
CREATE FUNCTION STOCK_QUOTE1 (stockticker XML)
                                        RETURNS XML
LANGUAGE SQL CONTAINS SQL
EXTERNAL ACTION NOT DETERMINISTIC
RETURN
(VALUES soaphttp (
 ‘http://localhost:8080/soap/servlet/rpcrouter’,
 ‘’, stockticker));
```

Figure 6 Custom EJB components compared to DADX files

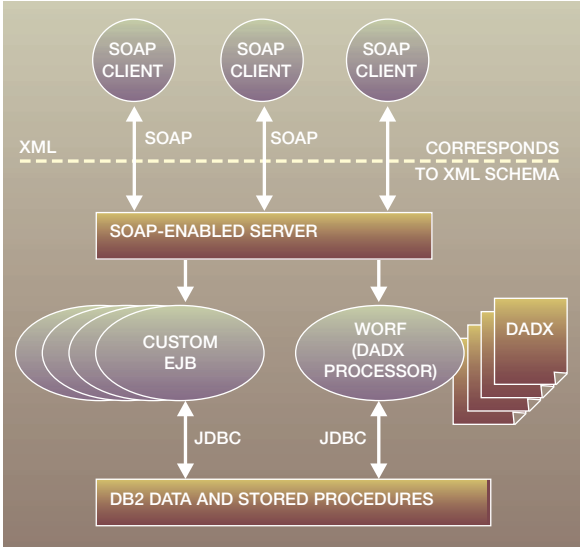
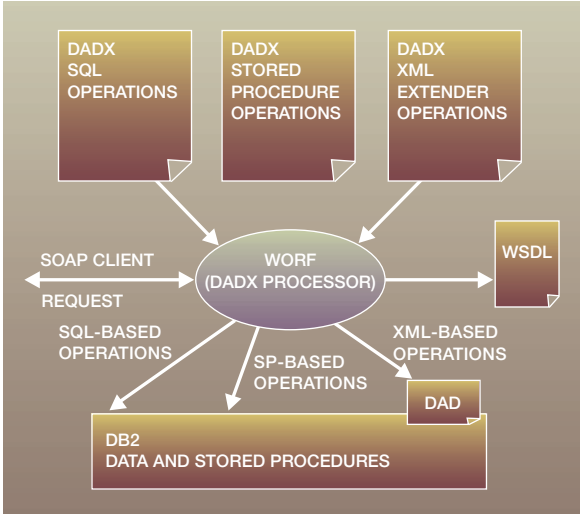


Figure 7 Three DADX operations



An SQL application can use SQL/XML functions to construct the XML input for the STOCK_QUOTE1 function, and retrieve the XML output for stock data stored in the MYPORTFOLIO table. An example of the SQL statement using the STOCK_QUOTE1 function is

```
SELECT symbol, STOCK_QUOTE1 (
          XMLElement (NAME “getRTQuote”,
          XMLElement (NAME “stockticker”,
          XMLElement (NAME “symbol”, symbol)))) AS quote
FROM myportfolio;
```

The results from this query are shown in Table 9. SOAPHTTP() is the DB2 SOAP requestor. This function generates the SOAP envelope and communicates with the SOAP provider through HTTP. It receives the SOAP response, parses the response, and returns the SOAP body to the invoker.

Option 2: SQL input and XML output. If an application does not need the flexibility of XML input, then usability can be improved by moving the XML construction part into the SOAP UDF, as follows:

```
CREATE FUNCTION STOCK_QUOTE2
  (stockticker varchar(5))
  RETURNS XML
  LANGUAGE SQL READS SQL DATA
  EXTERNAL ACTION NOT DETERMINISTIC
  RETURN
  WITH
- -1. Perform type conversions and prepare SQL input
- -    parameters for SOAP envelope
  soap_input (in)
  AS
     (VALUES XMLElement(NAME “RTQuote”,
        XMLElement(NAME “stockticker”,
        XMLElement(NAME “symbol”, stockticker))))
- -2. Submit SOAP request with input parameter and
- -    receive SOAP response
     (VALUES soaphttp (
     ‘http://localhost:8080/soap/servlet/rpcrouter’, ‘’,
     SELECT in FROM soap_input)));
```

In option 2, the client application provides SQL input; the SOAP UDF constructs the XML input according to the WSDL description of the provider. The client application receives the SOAP body as XML output.

The following query returns the same result as the query described in option 1:

```
SELECT symbol, STOCK_QUOTE2 (symbol) AS quote
FROM myportfolio
```

Option 3: SQL input and SQL output. In option 3, construction of the SOAP request as well as decomposition of the SOAP response is performed in the SQL-bodied function (see Figure 5). An SQL state-

Figure 8 DADX for getStatus and getAllStatus Web service operations

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:dtd1="http://schemas.myco.com/sales/getstart.dtd"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 <wsdl:documentation>Queries orders at myco.com.</wsdl:documentation>
    <operation name="getStatus">
   <wsdl:documentation>Gets the Status of an Order</wsdl:documentation>
   <query>
    <SQL_query>SELECT STATUS FROM ORDERS where PO_NUM = :inputponum</SQL_query>
     <parameter name="inputponum" type="xsd:string"/>
   </query>
 </operation>
 <operation name="getAllStatus">
   <wsdl:documentation>Get the status of all orders</wsdl:documentation>
   <query>
     <SQL_query>SELECT * FROM ORDERS</SQL_query>    </query>
</operation>
</DADX>
```

Figure 9 Output for the getStatus Web service

```
<?xml version="1.0" ?>
 <xsd1:getStatusResponse
       xmlns:xsd1="http://schemas.ibm.com/sales/malaikaorders.dadx/XSD"
       xmlns:xsd="http://www.w3.org/1999/XMLSchema"
       xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
      <return>
        <xsd1:getStatusResult
             xmlns:xsd1="http://schemas.ibm.com/sales/malaikaorders.dadx/XSD"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <getStatusRow>
                <STATUS>ONTIME</STATUS>
            </getStatusRow>
         </xsd1:getStatusResult>
      </return>
 </xsd1:getStatusResponse>
```

ment that uses this function is

```
SELECT p.symbol, x.last, x.ask, x.bid,
     x.change, x.pctchange, x.time
FROM myportfolio p,
   TABLE (Stock_Quote3(p.symbol)) AS x;
```

The results from this select statement are shown in Table 10.

DB2 as a Web service provider

DB2 data and applications (stored procedures) can be accessed as Web services by developing Web service implementations, which could be Java classes or Enterprise JavaBeans** (EJB) components, which access DB2 through JDBC** (Java Database Connectivity) requests. However, each time a different type of data access is desired, a new Web service imple-

Figure 10 Output for getAllStatus Web service

```
<?xml version="1.0" ?>
 <xsd1:getAllStatusResponse
      xmlns:xsd1="http://schemas.ibm.com/sales/malaikaorders.dadx/XSD"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema"
      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
     <return>
       <xsd1:getAllStatusResult
            xmlns:xsd1="http://schemas.ibm.com/sales/malaikaorders.dadx/XSD"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <getAllStatusRow>
             <PO_NUM>78453</PO_NUM>
             <CUSTOMER>yourMfgCo</CUSTOMER>
             <PART_NO>A4</PART_NO>
             <QTY>23</QTY>
             <COMMIT_DATE>2002-05-23</COMMIT_DATE>
             <STATUS>ONTIME</STATUS>
          </getAllStatusRow>
          <getAllStatusRow>
             <PO_NUM>12345</PO_NUM>
             <CUSTOMER>myMfgCo</CUSTOMER>
             <PART_NO>C7</PART_NO>
             <QTY>200</QTY>
              <COMMIT_DATE>2002-06-20</COMMIT_DATE>
             <STATUS>ONTIME</STATUS>
          </getAllStatusRow>
          <getAllStatusRow>
             <PO_NUM>53456</PO_NUM>
             <CUSTOMER>theirMfgCo</CUSTOMER>
             <PART_NO>B12</PART_NO>
             <QTY>5</QTY>
             <COMMIT_DATE>2002-04-23</COMMIT_DATE>
             <STATUS>COMPLETE</STATUS>
          </getAllStatusRow>
          <getAllStatusRow>
             <PO_NUM>35335</PO_NUM>
             <CUSTOMER>yourCo</CUSTOMER>
             <PART_NO>A3</PART_NO>
             <QTY>7</QTY>
             <COMMIT_DATE>2002-04-15</COMMIT_DATE>
             <STATUS>SHIPPED</STATUS>
          </getAllStatusRow>
       </xsd1:getAllStatusResult>
     </return>
 </xsd1:getAllStatusResponse>
```

mentation would be needed. A simpler, and more generic, alternative is to use the support shipped in DB2 v 8.1 that processes DADX (Document Access Definition extension[5]) files in conjunction with a general SOAP-enabled server, such as WebSphere. A DADX file is an XML document that represents a DB2 Web service. No application-specific code is required to use the DADX support. Figure 6 compares using custom user-written EJB components with using the WORF.

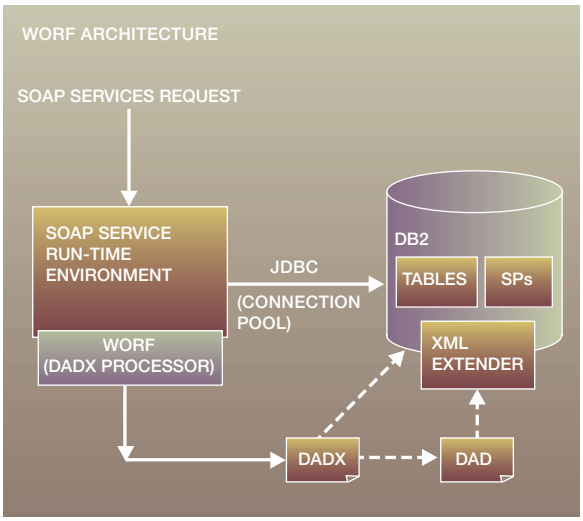
DADX. The DADX file supports three types of operations for making database requests as Web services (see Figure 7). The relational results from these requests are tagged as XML by the WORF in a customizable way. The three types of operations are:

1. SQL request: Any SQL request can be issued, including SELECT, UPDATE, user-defined functions, and DB2 XML Extender column facilities.

Figure 11 Operations findAll, findByColor, and findByMinPrice within a Web service

```
<?xml version="1.0"?>
<DADX xmlns="urn:ibm.com:dxx:dadx"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <documentation <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
 Provides queries for part order information at myco.com.</documentation>
 <operation name="findAll">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
       Returns all the orders with their complete details.</documentation>
  <retrieveXML>
     <DAD_ref>getstart_xcollection.dad</DAD_ref>
     <SQL_override>select o.order_key, customer_name, customer_email,
      p.part_key, color, quantity, price, tax, ship_id, date, mode  from order_tab o,
      part_tab p,
       table(select substr(char(timestamp(generate_unique())),16) as ship_id,
        date, mode, part_key from ship_tab) s
       where p.order_key = o.order_key and s.part_key = p.part_key
    order by order_key, part_key, ship_id
     </SQL_override>
  </retrieveXML>
 </operation>
 <operation name="findByColor">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Returns all the orders
     that include one or more parts that have the specified color, and only shows
     the details for those parts.</documentation>
    <retrieveXML>
     <DAD_ref>getstart_xcollection.dad</DAD_ref>
     <SQL_override>
      select o.order_key, customer_name, customer_email,
        p.part_key, color, quantity, price, tax, ship_id, date, mode
      from order_tab o, part_tab p,
        table(select substr(char(timestamp(generate_unique())),16) as ship_id,
         date, mode, part_key from ship_tab) s
       where p.order_key = o.order_key and s.part_key = p.part_key
        and color = :color
       order by order_key, part_key, ship_id
      </SQL_override>
      <parameter name="color" type="xsd:string"/>
  </retrieveXML>
 </operation>
<operation name="findByMinPrice">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
   Returns all the orders that include one or more parts that have a
  price greater than or equal to the specified minimum price, and only shows the details for
   those parts.</documentation>
  <retrieveXML>
   <DAD_ref>getstart_xcollection.dad</DAD_ref>
   <SQL_override>
    select o.order_key, customer_name, customer_email,
     p.part_key, color, quantity, price, tax, ship_id, date, mode
    from order_tab o, part_tab p,
     table(select substr(char(timestamp(generate_unique())),16) as ship_id,
       date, mode, part_key from ship_tab) s
    where p.order_key = o.order_key and s.part_key = p.part_key
     and p.price >= :minprice
    order by order_key, part_key, ship_id
   </SQL_override>
   <parameter name="minprice" type="xsd:decimal"/>
 </retrieveXML>
 </operation>
</DADX>
```

Figure 12 WORF architecture



2. Stored procedure call: Any DB2 stored procedure can be invoked.
3. DB2 XML Extender stored procedure call: Any DB2 XML Extender stored procedure can be invoked. These stored procedures can provide two types of functionality. First, composition is used for detailed control on the tagging of the generated document using a DAD (Document Access Definition). This approach can be used to control where element repetition takes place, or to control the number of documents produced, for example. Second, decomposition is used to "shred" an XML document and transform the result into relational data.

Figure 8 shows a DADX file for the getStatus service described earlier. It contains two operations, both SQL-based: getStatus has one input parameter, poNum, and getAllStatus lists the status of all orders and has no parameters.

The simplicity of the SQL-based DADX file clearly illustrates how easy it is to build Web services based on the SQL language. The client is unaware that SQL is being used; for example, to invoke getStatus the client simply passes a purchase-order number. Figure 9 illustrates the results of getStatus for part "12345." Figure 10 illustrates the results of getAllStatus, which corresponds to the content of the orders table described earlier. Note that the column names are used as the basis for the XML tagging of the results.

The DADX file shown in Figure 11 defines three Web service operations that utilize the DB2 XML Extender functionality. The findAll operation takes no parameters. The returned data are formatted as XML in accordance with the DB2 XML DAD getstart_xcollection.dad, which is referred to in the DADX. The SQL request in the DB2 XML DAD is overridden by the SQL statement in the SQL_Override section. The findByColor operation takes one parameter, color, which forms part of the WHERE clause in the SQL request. The notation for parameters is very similar to that used for host variables in SQL. The findByMinPrice operation takes one parameter, minprice.

In general, only desired parameters are exposed. For example, if a stored procedure has many input parameters, it is possible to expose just one parameter to the client, leaving the other parameters as default values set in the DADX. The DADX can be created using a text or XML editor or with WebSphere Studio Application Developer as described in the next section. To deploy a DADX-based Web service, the required DADX files are placed in the appropriate SOAP-enabled Web server directory along with the WORF.

Web Services Object Runtime Framework (WORF). WORF provides the run-time environment for Web services defined by DADX. WORF is implemented as an extension to an Apache SOAP 2.2 run-time component. It runs with the Apache Web server[6] and WebSphere application server. Figure 12 shows WORF hosted in a SOAP service run-time environment. WORF receives an HTTP SOAP GET or POST service request from the SOAP RPC router. The service endpoint of the request specifies a DADX or DTD (Document Type Definition) file and the requested action. (Note that WORF converts DTDs to XSDs. DB2 uses either DTDs or XSDs, but WSDL uses only XSDs.) The requested action can be a DADX operation or a command, such as TEST for the SOAP client test page, WSDL for the WSDL generations, or XSD (XML Schema Definition) for XML schema generation. WORF loads the DADX file specified in the request and connects to DB2 to run any SQL statements. Parameter markers in SQL statements are replaced with the requested values. WORF formats the result of the SQL statement into XML, converting types as necessary, and returns a SOAP response envelope to the SOAP client.

Figure 13 Generated WSDL fragment for the findAll, findByColor and findByMinPrice Web services

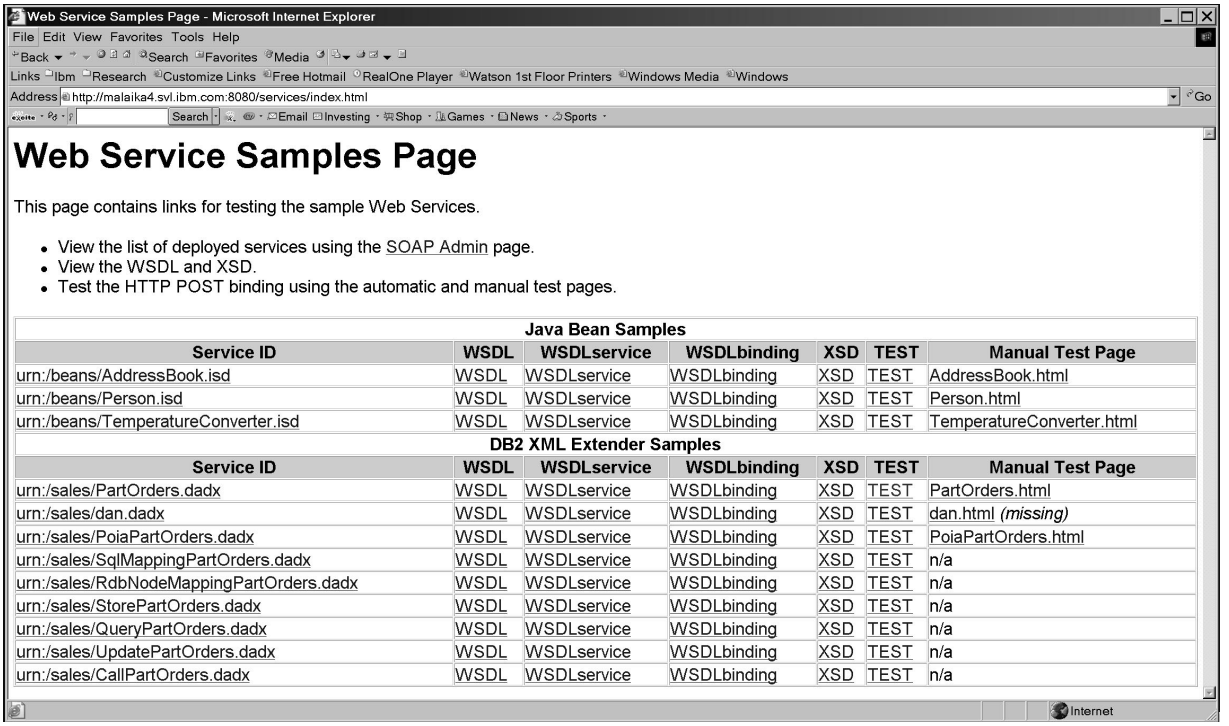
```
        <element name="findAllResult">
          <complexType>
            <sequence>
                    <element maxOccurs="unbounded" minOccurs="0"
                     ref="imp1:Order" />
              </sequence>
            </complexType>
          </element>
        <element name="findByColorResult">
          <complexType>
            <sequence>
                    <element maxOccurs="unbounded" minOccurs="0"
                     ref="imp1:Order" />
              </sequence>
            </complexType>
          </element>
        <element name="findByMinPriceResult">
          <complexType>
            <sequence>
                    <element maxOccurs="unbounded" minOccurs="0"
                     ref="imp1:Order" />
              </sequence>
            </complexType>
          </element>
        </schema>
      </types>
      <message name="findAllInput" />
    <message name="findAllOutput">
        <part element="xsd1:findAllResult" name="return" />
      </message>
     <message name="findByColorInput">
        <part name="color" type="xsd:string" />
      </message>
     <message name="findByColorOutput">
        <part element="xsd1:findByColorResult" name="return" />
      </message>
    <message name="findByMinPriceInput">
        <part name="minprice" type="xsd:decimal" />
      </message>
    <message name="findByMinPriceOutput">
        <part element="xsd1:findByMinPriceResult" name="return" />
      </message>
```

Generating WSDL and XML schema from the DADX. The WORF produces WSDL and XML schema for the operations included in the DADX. As illustrated in the fragment of WSDL shown in Figure 13, generated from the DADX illustrated in Figure 11, the client application is not aware that it is invoking DB2 or executing SQL. The Web services client sees operations and parameters only.

Testing DADX-based Web services through WORF. WORF provides a facility to test DADX files. The DADX illustrated in Figure 11 appears in Figure 14 as PartOrders.dadx. The generated WSDL illustrated in Figure 13 is generated by clicking on the WSDL link in the appropriate row.

Figure 15 illustrates what happens when the TEST link is selected for PartOrders.dadx: the PartOrders.dadx operations are displayed. They can be selected in turn to test the Web service, and input parameters can be typed in to complete the testing. As well as support for testing, the WORF provides support for connection pooling and security management.

Figure 14 Web service samples page



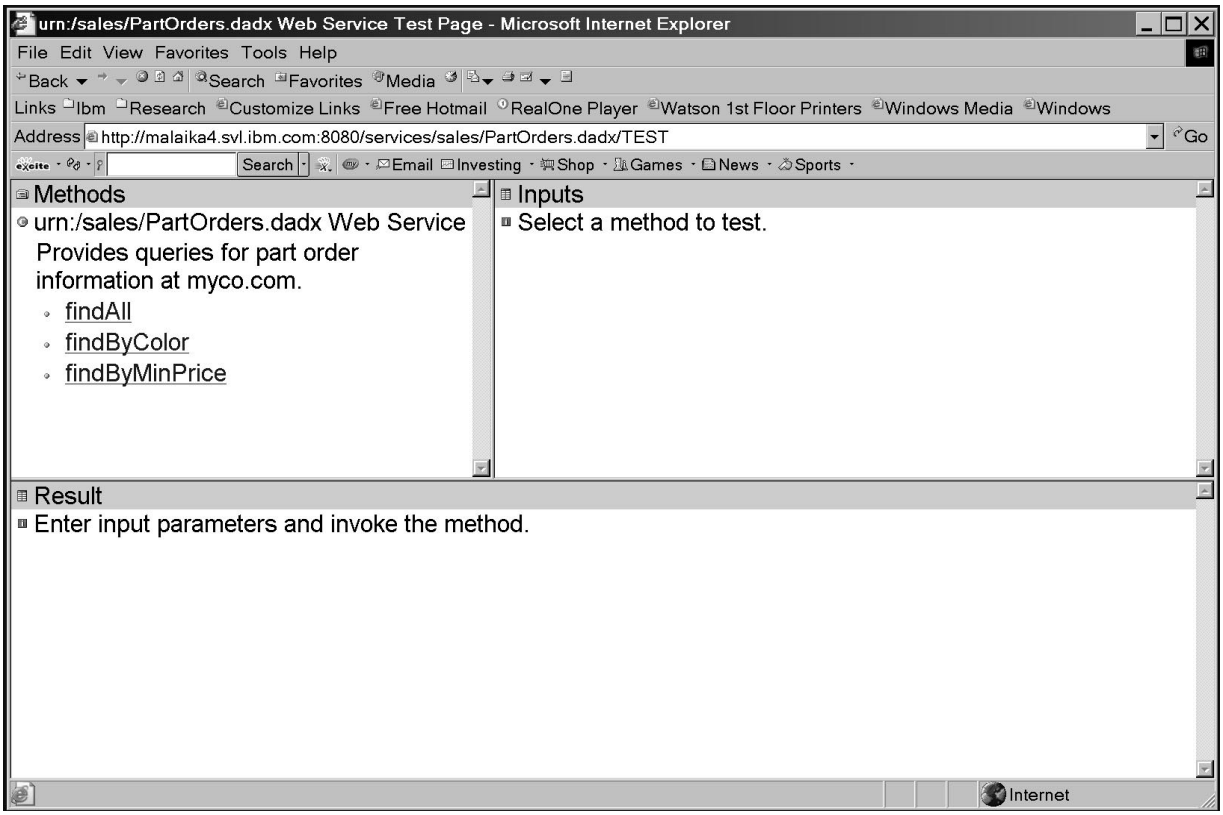
Tools

Development tools have kept pace with the rapid emergence of Web services. They provide a number of facilities for users to easily browse and find Web services, write applications that access these Web services, and develop, test, and publish new Web services. Both the IBM WebSphere Studio product family, including WebSphere Studio Application Developer and WebSphere Studio Site Developer, and Microsoft Visual Studio**.NET include extensive general Web service development support, and the WebSphere Studio product has a number of features specifically in support of DB2 Web services. The remainder of this section highlights some of the functionality provided in WebSphere Studio Application Developer (see Figure 16). For more details on this functionality see Reference 5, and for more information on using WebSphere Studio to develop DB2 Web services, see References 7 and 8.

WebSphere Studio simplifies the task of developing DB2 Web services; a user can build a Web service without writing any code. A rich set of XML-based tools is provided, including tools for building XML documents, DTDs, and XML schemas, mapping relational data to XML, and producing XML-extender DAD files. An integrated graphical query builder supports point-and-click construction of the SQL statements used in composing Web services. A wizard leads the developer through the steps to produce the Web service DADX file described previously. Developers select one or more SQL statements or DAD files to be mapped to Web service operations, and the DADX file is produced and saved in a DADX group that holds the database connection-specific information. A second wizard is used to turn the DADX file into a Web service. Options allow the user to also generate a proxy to access the Web service and launch the test client, immediately testing the execution of the new Web service using the integrated WebSphere test environment. There are also additional options to generate sample clients and to publish Web services by launching the integrated UDDI explorer. When the developer is satisfied, the Web service can be easily deployed to a production server.

WebSphere Studio also provides a rich set of functions for building the applications that access Web

Figure 15 Web service test page



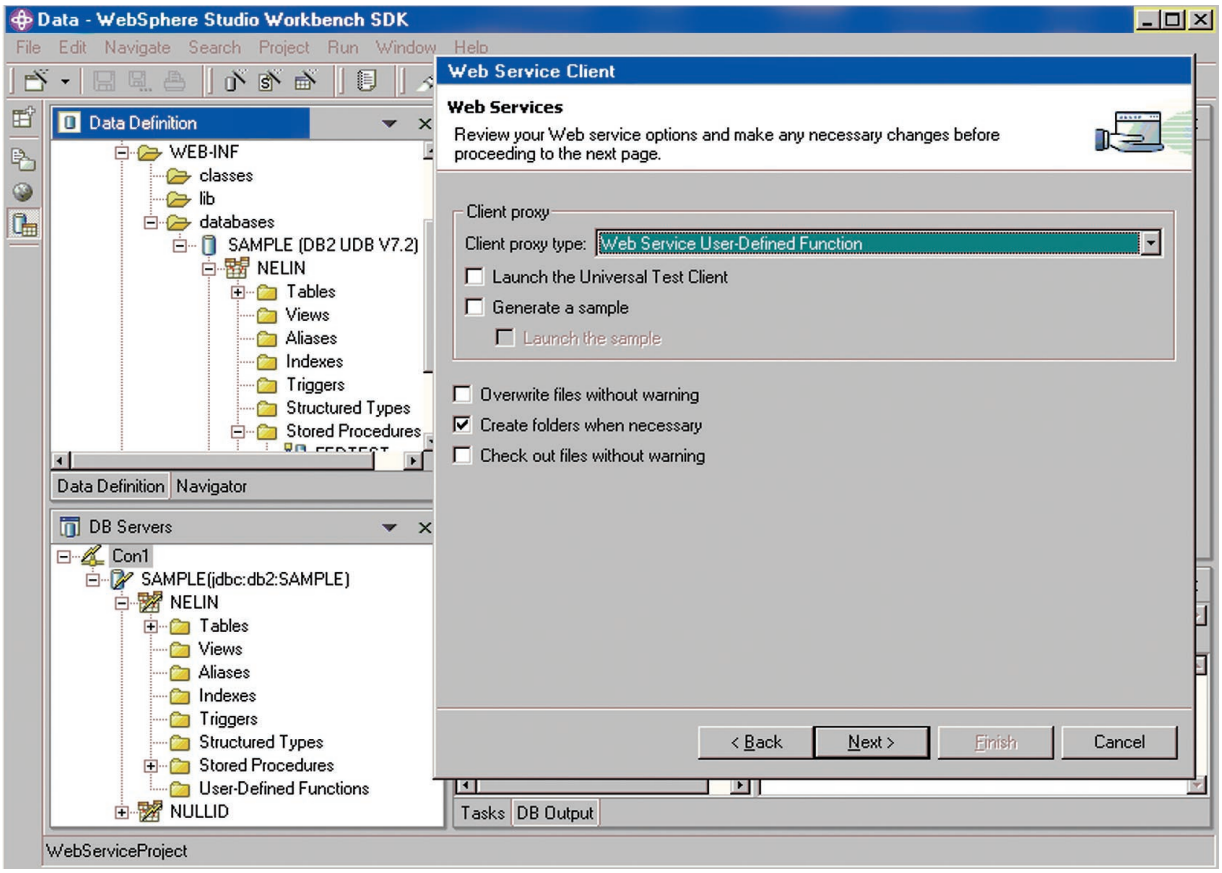
services. Today's support includes the automatic creation of Java proxies from WSDL files, and in the future the support will be enhanced to generate SQL-bodied UDFs as described earlier. The SQL UDF generator allows developers to specify the WSDL files of interest. The operations defined in a WSDL file can then be mapped to UDFs. The input and output of the operations is mapped to parameters and return values of the generated UDFs. In addition, two styles of UDF proxies are supported. The first is an early-binding form with the UDF mapped to a specific endpoint. In this case, when the UDF is executed one specific instance of a Web service is invoked and its results returned by the UDF. The second is a late-binding form with the endpoints passed in at run time. In this case, the UDF can be invoked against a set of Web services that support the same interface, and the results are returned in table form. Both the command line and wizard form of the tools generate SOAP UDFs with no coding required by developers.

Conclusion and future work

Web services have an increasingly important role, on the Internet as well as within corporations, as a convenient way to expose services and data to applications. In this paper, we show how a relational database such as DB2 can both support and exploit Web services technologies, to simplify access to data and stored procedures and to extend the reach of DB2 queries into external sources addressable through Web services. Using the inherent power of the DB2 engine, we can easily perform set-oriented queries over external data and enrich traditional SQL queries with real-time data. Furthermore, integrated tools allow us to take advantage of these features without writing programs.

The functionality described in this paper is powerful, allowing DB2 to integrate existing relational business data with external data. The external data are usually owned by different organizations, and there-

Figure 16 Web services development in WebSphere Studio Application Developer



fore cannot be stored in the same database. Furthermore, the external data are not static—they change based on external factors, for example weather forecast, stock price, airfare, or news feed. Although DB2 runs SOAP requests in a reliable and scalable environment, the fact that a database query depends on an external, potentially unreliable, low-bandwidth SOAP provider sets some limits on the overall query performance. Future work on SOAP response caching, in cooperation with service providers, will improve reliability and scalability of DB2 Web services. We have several projects underway that continue to extend our Web services support. As Web services continue to evolve and mature, we will augment our work to support emerging standards and practices. In particular, standards activities around security and transactions will need to be part of future development.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., or Microsoft Corporation.

Cited references and note

1. Web Services Activity, W3C Architecture Domain, see http://www.w3.org/2002/ws/.
2. E. Christensen, F. Curbera, G. Merideth, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, W3C Note (March 2001), available at http://www.w3.org/TR/wsdl.
3. For details on user-defined functions, see for example: (1) the IBM DB2 Application Development Guide, available at http://www-3.ibm.com/software/data/db2/udb/ad/v7/adg/db2aOe71.pdf, or (2) D. Chamberlin, *A Complete Guide to DB2 Universal Database*, Chapter 6.4, Morgan Kaufmann Publishers, San Francisco, CA (1998).
4. J. E. Funderburk, S. Malaika, and B. Reinwald, "XML Programming with SQL/XML and XQuery," *IBM Systems Journal* **41**, No. 4, 642–665 (2002, this issue).

5. DB2 Web Services, see http://www.ibm.com/software/data/webservices/.
6. The Jakarta Project, Apache Tomcat, see http://jakarta.apache.org/tomcat/index.html.
7. "Developing XML Web Services with WebSphere Studio Application Developer," *IBM Systems Journal* **41,** No. 2, 178–197 (2002).
8. R. J. Brunner, F. Cohen, F. Curbera, D. Govoni, S. Haines, M. Kloppmann, B. Marchal, K. S. Morrison, A. Ryman, J. Weber, and M. Wutka, *Java Web Services Unleashed*, Sam's Publishing, Indianapolis, IN (2002), Chapter 23.

Accepted for publication August 19, 2002.

Susan Malaika *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: malaika@us.ibm.com).* Ms. Malaika is a senior software engineer with IBM's Silicon Valley DB2 development group. She works in the area of XML, DB2, and the Web.

Constance J. Nelin *IBM Software Group, 11501 Burnett Road, Austin, Texas 78758 (electronic mail: nelin@us.ibm.com).* Ms. Nelin is a Senior Technical Staff Member in the IBM Database Advanced Technology area. She has worked for IBM since 1987, with a focus on database application development support and tooling. She has responsibility for application development tooling strategy, architecture, and development for data management. This covers the application development support for the full DB2 family spanning the areas of core relational database, federated database, XML, Web services, and messaging features.

Rong Qu *IBM Software Group, 11501 Burnett Road, Austin, Texas 78758 (electronic mail: qu@us.ibm.com).* Ms. Qu is a software engineer in the IBM Database Technology Institute for e-Business with seven years' experience in software development. Her recent projects include DB2 integration with MQSeries® and Web services.

Berthold Reinwald *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: reinwald@almaden.ibm.com).* Dr. Reinwald joined the IBM Almaden Research Center in 1993, after finishing his Ph.D. degree in computer science from the University of Erlangen-Nuernberg. His Ph.D. thesis on workflow management received the "best Ph.D. thesis" award from the university and was published as a book. At IBM Research, Dr. Reinwald contributed to SMRC (shared memory-resident cache) in DB2 Common Server, query explain tools, workflow management with Lotus Notes®, FlowMark®, and MQSeries, researched and delivered in DB2 Universal Database® support for OLE/COM, OLEDB, XML, and most recently Web services. Dr. Reinwald is active in the design, architecture, and implementation of SQL extensions for XML.

Daniel C. Wolfson *IBM Software Group, 11501 Burnett Road, Austin, Texas 78758 (electronic mail: dwolfson@us.ibm.com).* Mr. Wolfson is a Senior Technical Staff Member and manager in the Database Technology Institute for e-Business. With more than 15 years of experience in distributed computing, his interests have ranged broadly across databases, messaging, and transaction systems. He is a lead architect in the information integration area, focusing on DB2 integration with WebSphere, MQSeries, workflow, Web services, and asynchronous client protocols.