Developing XML Web services with WebSphere Studio Application Developer

by C. Lau
A. Ryman

Web services have recently emerged as a powerful technology for integrating heterogeneous applications over the Internet. The widespread adoption of Web services promises to usher in an exciting new generation of advanced distributed applications. These will support a new and growing set of specifications, such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). Extensible Markup Language (XML) and its associated family of standards also play a central role in Web services by providing a data interchange format that is independent of both programming languages and operating systems. The application developer seeking to reap the benefits of Web services is therefore faced with a significant, and potentially steep, new learning curve. Clearly, application development tools that lower this barrier are crucial for the rapid and widespread adoption of Web services. This paper discusses the development tasks associated with XML Web services and describes a new suite of tools that improve developer productivity, by reducing the requirements for detailed knowledge of the underlying specifications and standards, and allow the developer to focus on the business problem domain. This suite of XML and Web services tools is part of IBM's recently released WebSphere® Studio Application Developer product, which is based on the new Eclipse open source tool integration platform.

Web services provide a distributed computing technology for integrating applications over the Internet. Such a technology has the potential to dramatically transform our information-based economy. However, there have been many distributed computing technologies in the past, for example, Common Object Request Broker Architecture** (CORBA**), Microsoft DCOM**, and Java** Remote Method Invocation (RMI), yet none has become widespread on the Internet. Although Web services technologies are described in detail in the other papers of this issue of the *IBM Systems Journal*, it is useful here to briefly compare Web services with the other distributed computing technologies so that we can understand what they have in common and what has changed. Application developers who are already familiar with a previous distributed computing technology will be able to immediately apply much of their knowledge and experience to Web services. An overview of Web services[1] as well as in-depth technical information[2] can be found on the Web.

All distributed computing technologies, including Web services, provide a mechanism for a client program, executing on a local host, to invoke a server function that executes on a remote host and to receive the result of the remote execution. The style used by the client program to invoke the server function depends on the distributed computing technology. For example, the client program may call a re-

0018-8670/02/$5.00 © 2002 IBM

mote procedure, invoke a method on a remote object, or put messages on and get messages from a queue. These invocation styles are referred to as remote procedure call (RPC), Remote Method Invocation (RMI), and message queuing.

The programming interface supported by the remote server function is often specified using an interface definition language (IDL). The IDL typically specifies the operations provided by the remote server function, their input and output parameters, and their exceptions. Tools are provided to generate a client "stub," or proxy, and a server skeleton from the IDL. Although the terms stub and proxy are both used in practice, the term proxy is used in the following discussion. The client proxy gives the client program a convenient way to invoke the remote server function. The server skeleton provides a place to include the implementation of the remote server function. The proxy and skeleton hide the protocol details from the local client program and the remote server function implementation.

The local client program calls the client proxy through its programming language interface. The client proxy converts the programming language input parameters into a byte stream that can be sent to the remote server using some protocol and transport mechanism. The process of converting programming language parameters into a protocol-dependent byte stream is referred to as "marshaling." The remote host routes the invocation to the correct server skeleton, which converts the incoming byte stream back into programming language parameters. The processes of converting the protocol-dependent byte stream back into programming language parameters is referred to as "unmarshaling." Marshaling and unmarshaling may also be referred to as "serialization" and "deserialization." The client program and server skeleton may use different programming languages and operating systems. The server skeleton passes the input parameters to the function implementation for execution, receives the execution result, marshals it into a response, and sends it back to the client proxy. The client proxy unmarshals the response and returns the result to the local client program.

If Web services were simply another distributed computing technology, there would be little reason to get excited. However, there are some fundamental differences between Web services and its predecessors that make the technology very powerful. The main difference between Web services and the previous distributed computing technologies is that they are designed for the extremely heterogeneous environment of the Internet. The Internet is composed of a huge number of highly diverse computers, ranging from the simplest client devices to the most complex mainframe servers. In addition, these computers are not under the control of a single information technology department that can impose uniform software standards. Web services specifications are therefore completely independent of programming language, operating system, and hardware, and they enforce very loose coupling between the client and the server.

To see why the programming language and operating system independence of Web services is a major advance, consider the tight coupling present in CORBA, Microsoft DCOM, and Java RMI. CORBA couples the client and server with an underlying object model. In addition, the protocol used by CORBA, Internet Inter-ORB** Protocol (IIOP**), was not initially specified tightly enough to guarantee interoperation between different vendors. More recent versions of IIOP seek to achieve vendor interoperability. DCOM couples the client and server through the Microsoft Windows** operating system. Attempts to implement DCOM on other operating systems have not gained significant market acceptance. Finally, Java RMI couples the client and server through the Java programming language. In practice, successful use of Java RMI involves even tighter coupling, because Java object serialization is very sensitive to virtual machine levels and class versions. CORBA, DCOM, and Java RMI are important distributed programming technologies and will not be replaced by Web services in situations where tight coupling is both desirable and possible. However, in situations that require loose coupling, Web services will likely become the dominant technology.

Web services are designed for maximum interoperability across the Internet. Extensible Markup Language (XML)[3] plays a key role as the main data interchange format for Web services parameter marshaling. In comparison with the highly optimized binary formats used by other distributed computing technologies, Web services may appear inefficient, but those technologies entailed tight coupling between the client and server and would therefore never become dominant on the Internet. In addition, the continual improvements in network bandwidth, processor speed, and compression techniques make the more verbose nature of XML of less concern.

The use of XML in Web services is a major difference from other distributed computing technologies

and has important application development consequences. In the other technologies, the IDL makes use of data structures that correspond closely with the supported programming languages, often involving certain assumptions about an underlying object model, and the data interchange format is a highly optimized binary stream that tightly couples the client and the server. In XML Web services, the data types are specified using XML Schema (XSD)[4] and the data interchange is in XML format. Although the Web services run-time environment can automatically encode programming language data structures as XML, this introduces coupling between the programming language and the Web service interface. Application developers will find it beneficial to design interfaces using XSD in order to give them greater implementation flexibility. As application developers begin to think in terms of XML rather than programming languages when defining Web services, they will begin to need tools for authoring XSD and Web Services Description Language (WSDL).[5]

As Web services are initially adopted, client proxies and server skeletons will normally be implemented in conventional programming languages, such as Java and JavaScript**. Tools for mapping between XML Schema and programming language data structures will be used to hide the XML details from the programmer. However, as XML becomes more familiar to programmers, its role may broaden to also include processing. Examples of XML processing technology include Extensible Stylesheet Language Transformations (XSLT),[6] XML support in databases such as IBM Database 2* (DB2*) Universal Database (UDB),[7] and XML Query[8] language. These technologies allow the programmer to specify processing natively in terms of XML as opposed to converting between XML and conventional programming languages. In many cases it may be more productive for programmers to specify simple processing using an XML technology, and to call out to conventional programming languages, such as Java, for more complex processing. Tools for editing and debugging XML processing technologies will become important.

In the case of Web services, the role of the interface definition language is played by WSDL, and although WSDL can in theory be extended to describe arbitrary transports and marshaling, the use of HyperText Transmission Protocol (HTTP), Simple Object Access Protocol (SOAP),[9] and XML are likely to dominate on the Internet. Web services can also be used effectively for integrating applications within an enterprise that has control over the computing infrastructure. Other transport mechanisms, such as message queuing, may be used within the corporate intranet. However, many enterprises have a heterogeneous computing infrastructure that has many characteristics in common with the Internet. Corporate mergers and acquisitions are another factor that increases the diversity of the enterprise computing environment and that makes the use of XML Web services attractive. In fact, many enterprises may benefit from implementing Web services behind their firewall before they expose them to partners and customers.

The SOAP run-time environment

Apache SOAP[10] is a Java Open Source implementation of the SOAP specification that runs in WebSphere*, Apache Tomcat,[11] and other Java 2 Platform, Enterprise Edition (J2EE**) -compliant application servers. IBM SOAP adds security, administration, and tracing enhancements to Apache SOAP. IBM SOAP is a fully supported component of WebSphere 4.0 that provides a production-ready environment for deploying Web services.

Due to the rapid evolution of Web services specifications and implementations, developers can expect to see a steady stream of new versions of Apache SOAP. Many developers will want to experiment with the latest Apache SOAP versions before they become part of the fully supported IBM SOAP product implementation. The Web services tools described here therefore support deployment to both Apache SOAP and IBM SOAP so that developers can track the latest advances and be well positioned to quickly exploit them as soon as they become part of the supported production environment.

The Apache SOAP run-time environment includes support for both Java clients and Java servers. A Java client can access SOAP Web services implemented in any programming language. Indeed, the power of Web services is that the programming language used to implement the service is both unknown and irrelevant to the client. The Apache SOAP run-time environment has built-in support for implementing Web services as Java classes or scripting language programs and has an extension mechanism, called the Pluggable Provider Interface, for implementing other types of Web services. Providers for Enterprise JavaBeans** (EJB**), database stored procedures, and Microsoft COM objects are included with Apache SOAP. IBM SOAP includes a provider for DB2 XML Extender, which is described later in more detail. Providers allow developers to deploy existing compo-

nents as Web services, and to develop new Web services using languages other than Java.

Java/XML type mapping. As previously discussed, any distributed computing technology must provide support for marshaling and unmarshaling data. In the case of Apache SOAP, the run-time component is implemented in Java code and the data interchange format is XML, so rules for specifying the mapping between Java and XML types must be provided as part of the implementation of every Java client or Web service. The SOAP specification includes a definition of the SOAP encoding style, which further guides the marshaling and unmarshaling process, but other encoding styles can be used. The Apache SOAP run-time environment allows the developer to specify how to map between Java and XML types for a given encoding style. The developer can specify a serializer to marshal the Java type to XML, a deserializer to unmarshal the XML type to Java, or both for two-way mapping.

The rules for mapping between Java and XML types are stored in a SOAP mapping registry object that is used by either the Java client proxy or the Web service. The SOAP mapping registry has predefined rules for mapping between simple Java and XML types. However, if the Web service interface involves complex XML types, then the developer must specify how they are mapped to corresponding Java types.

The simplest way to map XML types is to use the Java type org.w3c.dom.Element, which represents a generic XML element in the Document Object Model (DOM).[12] This technique is referred to as literal XML encoding. However, this approach requires that the developer understand the details of the XML type and the Java application programming interface (API) for DOM. The use of Element is error prone, because the developer can create an invalid element, but for XML types that are not too complex, this may be the best approach.

If the developer is deploying a Java class as a Web service, and the Java class uses a Java bean in its interface, then the bean can be mapped to XML using the SOAP encoding style. The SOAP specification defines how to represent programming language data types as XML. The Apache SOAP run-time environment includes a bean serializer that implements SOAP encoding for Java beans. The bean serializer can marshal an instance of a Java bean into XML or unmarshal XML into an instance of a Java bean.

If the developer is proceeding from the WSDL definition of a Web service and its interface includes a complex XML type, then the XML type can be mapped to a custom Java class that is generated from the XML Schema type definition using tools provided in the WebSphere Studio Application Developer suite. The custom Java class includes methods for marshaling and unmarshaling. The ability to generate a Java class that corresponds to an XML Schema complex type is useful for general application development and is included as part of the XML Schema Editor tool, which is described later in greater detail.

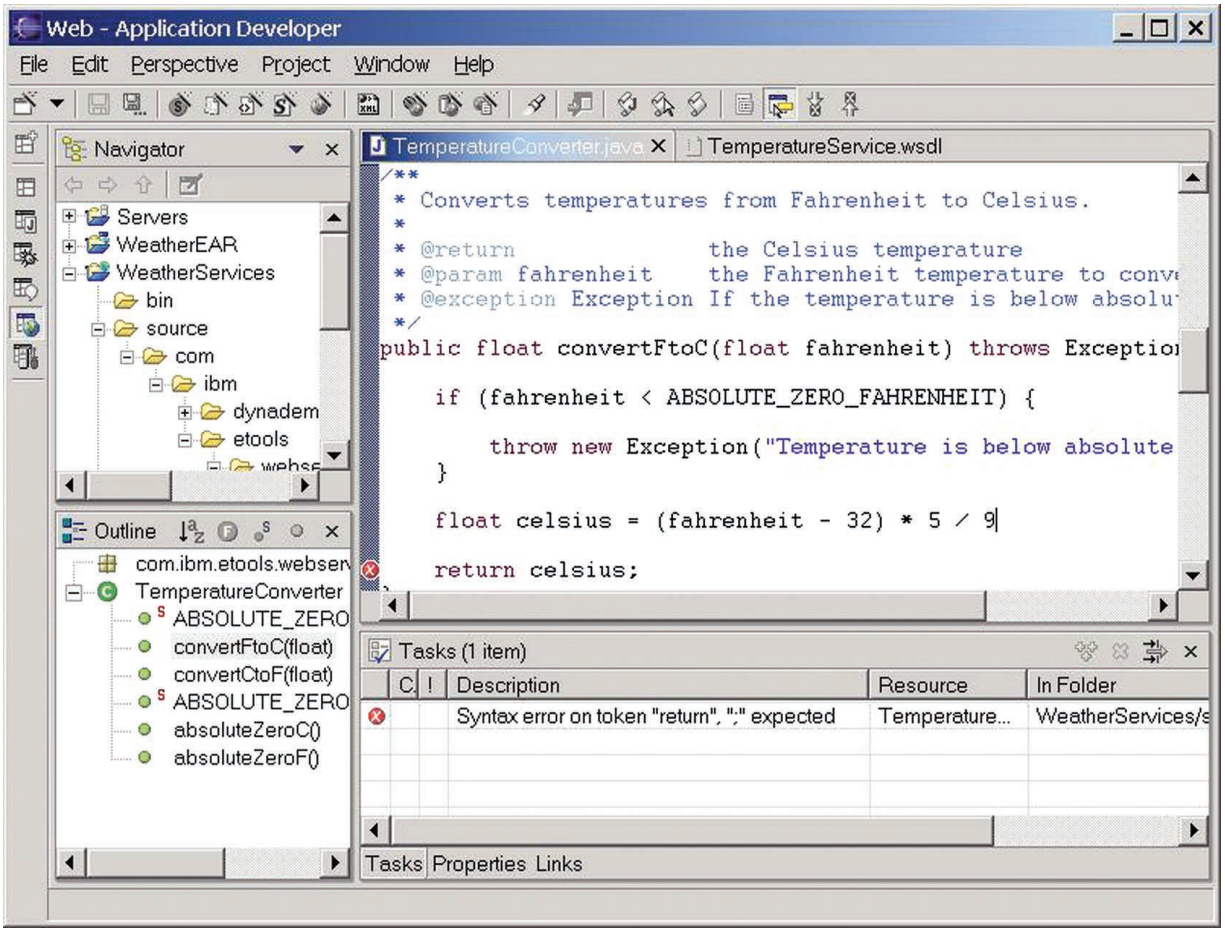
WebSphere Studio Application Developer

The following sections describe application development tools for XML Web services that are included as part of WebSphere Studio Application Developer, a new integrated development environment that is based in the Eclipse platform.[13] Eclipse is a Java open source development tool integration platform that will provide a common environment for all future IBM tools. Eclipse is designed to be easily extended by other tool vendors and customers. An early version of Eclipse, and the tools described here, was released on alphaWorks* as the IBM XML and Web Services Development Environment.[14] In addition to XML and Web services tools, WebSphere Studio Application Developer includes a complete suite of tools for Java programming, Web application development, database access, EJB creation, debugging, tracing, and performance monitoring.

WebSphere Studio Application Developer is a project-oriented, team-based development environment that allows the developer to launch editors and wizards against resources. A detailed description of Eclipse and WebSphere Studio Application Developer is beyond the scope of this paper. Instead, a few features of the environment are briefly described here.

Figure 1 shows the workbench window, which is the main window in the development environment. It contains a set of perspectives. The developer can switch between perspectives by clicking on an icon along the left of the window or using the Perspective menu. Each perspective contains a set of views and editors that are relevant to a particular task. The screenshot shows the Web perspective, which is useful for developing Web applications and Web services. Other perspectives are defined for Java programming, database development, server configuration, and XML development. In this perspective,

Figure 1 The workbench window



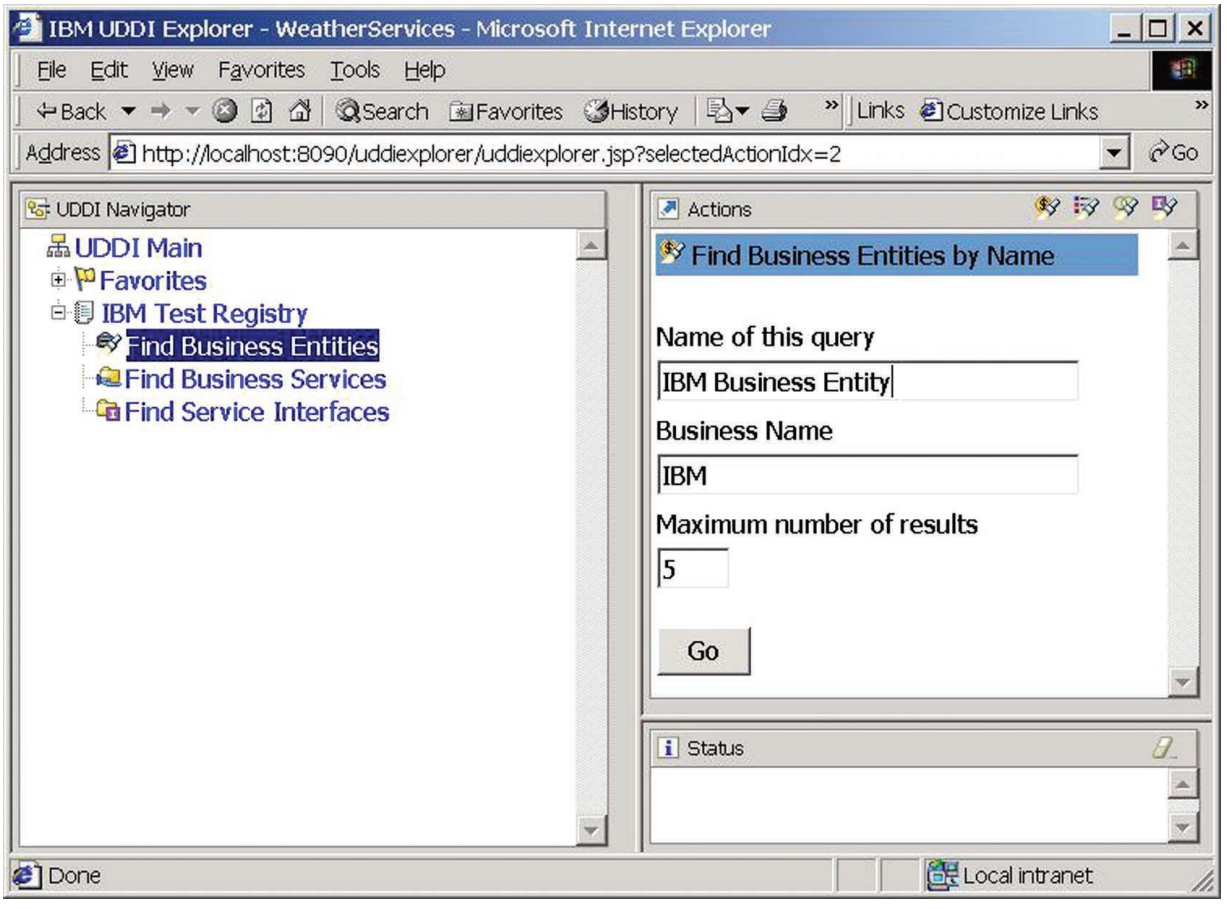
editors for two resources (TemperatureConverter.java and TemperatureConverter.wsdl) and the Navigator, Outline, and Tasks views are open. The Navigator view displays the resources in all the development projects. The Outline view displays the contents of the currently active editor, which in this example, is open on a Java source file. The Tasks view displays problems and user-defined tasks. In this example, the Task view displays an error message caused by a missing semicolon in the open Java source file. The environment has a user interface style that should be familiar to many developers, but it is unique in its extensibility. Developers can not only easily define new perspectives, but can also create new views and editors that integrate seamlessly with the environment.

Web services development tasks

Web services development adds several new tasks to the application development process. Some of these tasks should be familiar to those developers who have prior experience with other distributed programming technologies. It is useful to group the tasks according to the following development life-cycle activities:

- Discover existing Web services
- Access existing Web services and compose them into new applications and Web services
- Create new Web services by using existing components or developing new components
- Deploy new Web services into an application server
- Test new and existing Web services
- Publish new Web services

Figure 2 The UDDI Explorer



Discover. The *discover* task consists of locating the Web services needed to build a new application or Web service and importing their WSDL files into the development project. As the number of available Web services increases, programmers will have trouble discovering them. This problem is addressed by the Universal Description, Discovery, and Integration (UDDI) business registry,[15] which indexes Web services so they can be discovered quickly.

Developers can access UDDI by several means. UDDI itself is exposed as a SOAP Web service, so developers can write custom applications that search UDDI nodes. However, this is labor-intensive and requires a detailed knowledge of the UDDI programming interface. Each UDDI node also offers its own Web-based user interface, but this user interface differs from node to node, depending on the operator,[16] so a developer who needs to search nodes from several operators would have to learn multiple user interfaces.

WebSphere Studio Application Developer includes the UDDI Explorer, which provides a common user interface to any compliant UDDI business registry. The UDDI Explorer is seamlessly integrated into the development environment and shares its common look and feel. The UDDI Explorer lets the developer perform powerful queries against UDDI business registries and allows the results to be filtered and viewed in flexible ways. Figure 2 is a screenshot of the UDDI Explorer. As queries are executed, the results are added to the UDDI Navigator view and can act as the starting point for further queries. Working in this

way, the developer can navigate through the UDDI hierarchy to locate Web services that satisfy the application requirements. When the developer finds a suitable Web service, its WSDL description can be easily imported into the development project where it can be used in further development tasks. For example, the developer can generate a sample application that allows him or her to test the Web service. The UDDI Explorer supports the recommended best practices for using WSDL in UDDI business registries.[17]

Initially, UDDI registries were hosted only by IBM and Microsoft, who developed the specification. Now UDDI is beginning to be deployed by other organizations. For example, developers can find many interesting Web services in the UDDI registry maintained by XMethods.[18]

Access. The *access* task consists of developing client code that invokes the Web service. The input to this task is the WSDL file that describes the Web service to be accessed. The client code may be used as part of a new Web application or Web service. After the client code has been developed, it can be incorporated into the new application or Web service using standard programming tools such as editors, compilers, and debuggers. WebSphere Studio Application Developer supports Web service access from Java and JavaScript clients.

JavaScript clients. A JavaScript Web service client would normally run in a Web browser, although desktop JavaScript applications are also possible. WebSphere Studio Application Developer includes a JavaScript development environment that is integrated with the Web application development tools. The JavaScript development environment includes an editor that supports syntax highlighting and code completion, wizards for generating complex JavaScript code snippets, and a debugger. Support for JavaScript clients is important because of the wide adoption of JavaScript by Web application developers. Many developers find JavaScript easier to use than Java, and for simple applications, JavaScript may be the best choice.

The simplest form of JavaScript client uses the HTTP GET or POST Web service bindings to obtain an XML response for processing in the Web browser. This approach requires that the JavaScript client parse and process the XML response, perhaps using XSLT. Web browser-based XML processing has been available for some time now, and this approach is likely to be popular due to its simplicity.

A more sophisticated form of JavaScript client is based on SOAP. Microsoft Internet Explorer 5, and later versions, supports an HTML component, called the WebService Behavior,[19] which takes a WSDL file as input and dynamically creates a JavaScript object that acts as a SOAP client proxy for the Web service. With this approach, the developer programs directly in JavaScript, not XML.

Web browser access to Web services raises security concerns similar to those raised by Java applets. Specifically, if the Web browser allowed JavaScript code to access Web services located at any URL (uniform resource locator), then a Web page could attempt access to Web services located behind the firewall. Therefore, Web browsers are likely to restrict access by a Web page to those URLs that are located on the host that served the Web page. This means that if a Web page needs to compose Web services that are located on different hosts, then those Web services must be accessed indirectly by wrapping them with another Web service that resides on the Web page host and that redirects the requests. The wrapping Web service is in effect a server proxy. Therefore, Web browser access to Web services will not necessarily reduce the traffic on the Web page host. However, the benefit in creating server proxies is to make Web services more accessible to the very large community of developers who are skilled in client-side JavaScript programming. The restrictions on Web service access imposed by Web browsers can be entirely avoided by performing server-side access.

Java clients. A Java Web service client would normally be a server-side component such as a servlet, JavaServer Pages** (JSP**) Web page, Java bean, or EJB bean, but it could also be an applet or desktop application. The easiest way to access a Web service from a Java client is to use the Web Service Client Wizard to generate a client proxy from the WSDL file. The client proxy is a Java class that has a method for each operation defined in the WSDL file. The signature of each method matches the message parts for the operation. The client proxy uses the Apache SOAP run-time component to marshal the Java method arguments into XML and unmarshal the XML response into the Java return value. The rules for mapping between Java and XML are defined in a SOAP mapping registry object for the client. The client proxy may therefore also include custom Java classes that are generated by the XML Schema tools.

The generated Java client proxy accesses a generic Call object that is part of the Apache SOAP run-time component. The Call object takes the operation name and a list of arguments as input parameters, and can dynamically invoke the remote Web service operation in a way that is analogous to the way that ordinary Java methods can be invoked using reflection. Sophisticated Java programmers may find it useful to access the Call object directly without the use of a generated client proxy. Direct access to the Call object allows highly dynamic applications where, for example, the WSDL for the Web service is obtained at run time.

The Web Service Client Wizard can also generate and launch a sample JSP test client that lets the developer immediately test the Web service and provides a starting point for further development. The test client will be discussed later in more detail.

Create. The *create* task involves creating a new Web service. In general, a complete Web service implementation consists of a WSDL file that describes the service, a deployment descriptor that specifies a component that implements the service and other information, a component that implements the operations of the service, and associated code that performs type mapping or other functions. Depending on the implementation technology, some of these development artifacts may not be required.

The Apache SOAP run-time environment supports several styles of Web service implementation based on traditional software components, such as Java beans, EJB beans, database stored procedures, and scripting language programs. Each of these implementation styles is handled by a provider class that is part of an extensible framework. It is therefore possible to create Web services based on new types of components by defining an appropriate provider. For example, WebSphere Studio Application Developer supports Web services based on DB2 XML Extender, which will be described later. The provider type, and associated parameters such as the name of the component that implements the service and type mapping information, is defined in an Apache SOAP deployment descriptor, which is an XML file that normally uses the namespace prefix "ISD." We often refer to deployment descriptors as "ISD files" and use "ISD" as the file extension.

In addition to SOAP-based Web services, it is possible to create Web services that use the HTTP GET and POST bindings. These non-SOAP Web services can be implemented by servlets, JSP Web pages, Common Gateway Interface (CGI) programs, or other URL-addressable Web programming technologies.

The tasks involved in creating a Web service depend on the starting point. The two main scenarios are bottom-up and top-down. In the bottom-up scenario, the developer first creates or reuses a component that implements some business function, and then transforms it into a Web service by creating its WSDL file, deployment descriptor, and any other required supporting code. The bottom-up approach will be widely used in the initial phase of Web services adoption, because it allows developers to reuse existing components. In the top-down scenario, the developer first creates or is given the WSDL file for the service, and then must create a component to implement the operations. The top-down approach will become more important as industry groups begin to agree on standard WSDL interfaces to common business functions. A meet-in-the-middle approach is also possible. In this case the starting point consists of both the WSDL file and the implementation component, and the developer must create additional support code that maps between the two.

WebSphere Studio Application Developer supports the bottom-up approach for Java beans, EJB beans, Structured Query Language (SQL), and URLs, and the top-down approach for Java beans. Over time, the tool coverage for the top-down, bottom-up, and meet-in-the-middle approaches will expand to include more implementation technologies.

All major relational database vendors are incorporating XML capabilities into their products, and work is underway to extend the SQL standard to directly support XML. DB2 UDB includes the XML Extender, which allows XML documents to be stored in columns or composed and decomposed into tables as relational data. The mapping from XML to relational data is specified by an XML Document Access Definition (DAD) file. WebSphere Studio Application Developer includes the Relational Database to XML Mapper tool, which allows the developer to easily generate DAD files. This tool is described later in more detail.

The support for XML in DB2 makes it a natural choice for implementing Web services. WebSphere Studio Application Developer supports the creation of Web services implemented in DB2 XML Extender (DXX) that are defined by DAD Extension (DADX) files. A DADX file is an XML file that contains operations defined by normal SQL statements and calls to the spe-

Table 1 Developing the Flights Web Service

Task	Input	Output	Tool
Set up the Web application Add the Flights Web service group to the Web application. This group accesses the Airline database.	web.xml	web.xml, group.properties	DADX Group Configuration Wizard
Create the operation Create the SQL statement that lists the flights. This will become the listFlights operation of the Web service.	Airline database schema	listFlights.sqx	SQL Query Builder
Create the Web service Create the DADX file that contains the listFlights operation.	listFlights.sqx	Flights.dadx	XML from SQL Query Wizard
Deploy the Web service Deploy to the application server and create the WSDL.	Flights.dadx	Flights.isd, Flights.wsdl	Web Service Wizard
Test the Web service Create the Java client proxy and a JSP sample application to test the Web service.	Flights.wsdl	Flights.java, TestClient.jsp	Web Service Client Wizard

cial stored procedures included in DB2 XML Extender. IBM SOAP includes a provider for handling DADX files. WebSphere Studio Application Developer includes an SQL Builder tool that can generate DADX files. Developers can edit the generated DADX using a standard text editor or the XML Editor tool that is included with WebSphere Studio Application Developer. The SQL Builder and XML Editor tools are described later in more detail.

Deploy. The *deploy* task consists of configuring an application server to run the Web service and installing the Web service on the application server. WebSphere Studio Application Developer includes server tools for setting up the application server. A version of WebSphere is included with the development environment and developers can also use Tomcat. The Web Services Wizard automatically associates an instance of an application server with the Web project, installs the SOAP run-time component and all required run-time libraries, starts the server, and deploys the Web service to the application server.

The Apache SOAP run-time component manages deployed Web services using a pluggable configuration manager. The default configuration manager allows Web services to be deployed and removed at run time, and stores the list of deployed services in a serialized Java object that is read when the server starts. The Apache SOAP run-time component also has an XML-based configuration manager; it stores the list of deployed services as an XML file, which provides more portability. Allowing services to be freely deployed and removed at run time is convenient for developers but may not be suitable for production. The IBM SOAP run-time component includes a configuration manager suitable for production use. The IBM configuration manager also uses an XML file, dds.xml, to store the list of deployed services, and allows defined services to be suspended and resumed, but new services cannot be added at run time. The Web Services Wizard automatically creates dds.xml for the IBM configuration manager.

Deploying DADX Web services requires a further development step. The association between a DADX Web service and the database it accesses is defined by a Web services group. Each DADX service within a group accesses the same database. The database connection information and other properties com-

mon to the group are specified in a standard Java properties file, group.properties, which is placed in the home directory for the group along with all the DADX files in the group. The DADX provider is based on the Web Services Object Runtime Framework (WORF), an IBM extension to the Apache SOAP runtime environment. WORF includes the DXX invoker servlet, which instantiates a subclass of the standard SOAP RPC router servlet. The DXX invoker servlet extends the standard RPC router servlet by adding HTTP GET and POST bindings and automatically generating WSDL and a test page for the Web service. An instance of the DXX invoker servlet must be added to the Web application for each group of DADX Web services.

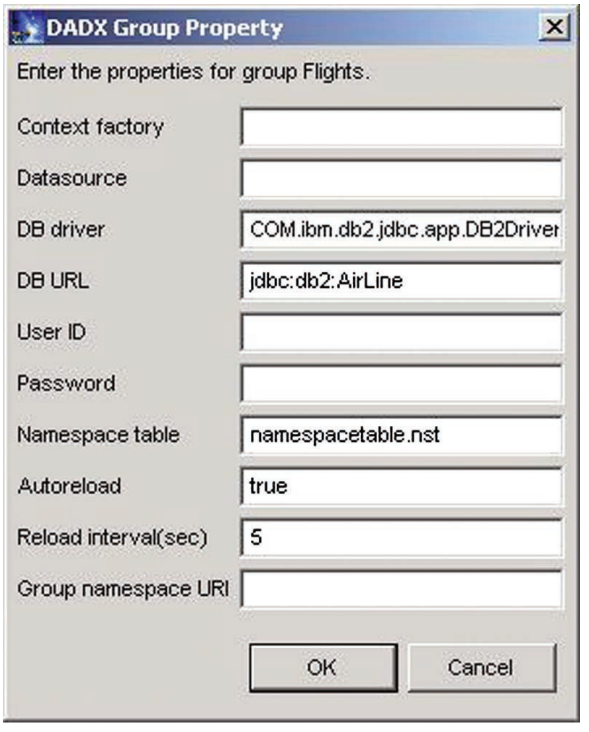
The task of setting up a DADX group is simplified by the DADX Group Configuration Wizard, which adds an instance of the DXX invoker servlet, creates the required directory structure, and writes the group.properties file.

Test. The *test* task consists of invoking the operations of a Web service to validate or understand its behavior. WebSphere Studio Application Developer supports Web service testing in several ways. As mentioned previously, the Web Service Client Wizard can generate a client proxy and a sample JSP-based test client that accesses it. The sample test client is intended to act as a starting point for further development but can also be used for testing the Web service. In addition to generating a sample test client that uses the Java client proxy, WebSphere Studio Application Developer includes the Universal Test Client, which can dynamically test the proxy or any other Java class without generating any code. Finally, Web services deployed using the WORF extension to the Apache SOAP run-time environment can dynamically generate a test page that enables any potential user of the Web service to test it from the user's browser.

Although these test methods are useful for unit test of the operations of a Web service, problems may occur at the protocol or transport level. For this class of problems, WebSphere Studio Application Developer includes a TCP/IP (Transmission Control Protocol/Internet Protocol) monitor that can be used to inspect HTTP requests and responses.

Publish. The *publish* task consists of registering a description of the Web service in a UDDI business registry so that it can be located by other application developers. The UDDI Explorer lets the developer update UDDI business registries in addition to searching them. The WSDL files created by WebSphere Studio Application Developer follow the UDDI best practices guidelines and are therefore ready to publish. The developer can easily publish a Web service by selecting its WSDL file in the navigator view and exporting it to UDDI. This action launches the UDDI Explorer.

Figure 3 The DADX Group Configuration Wizard

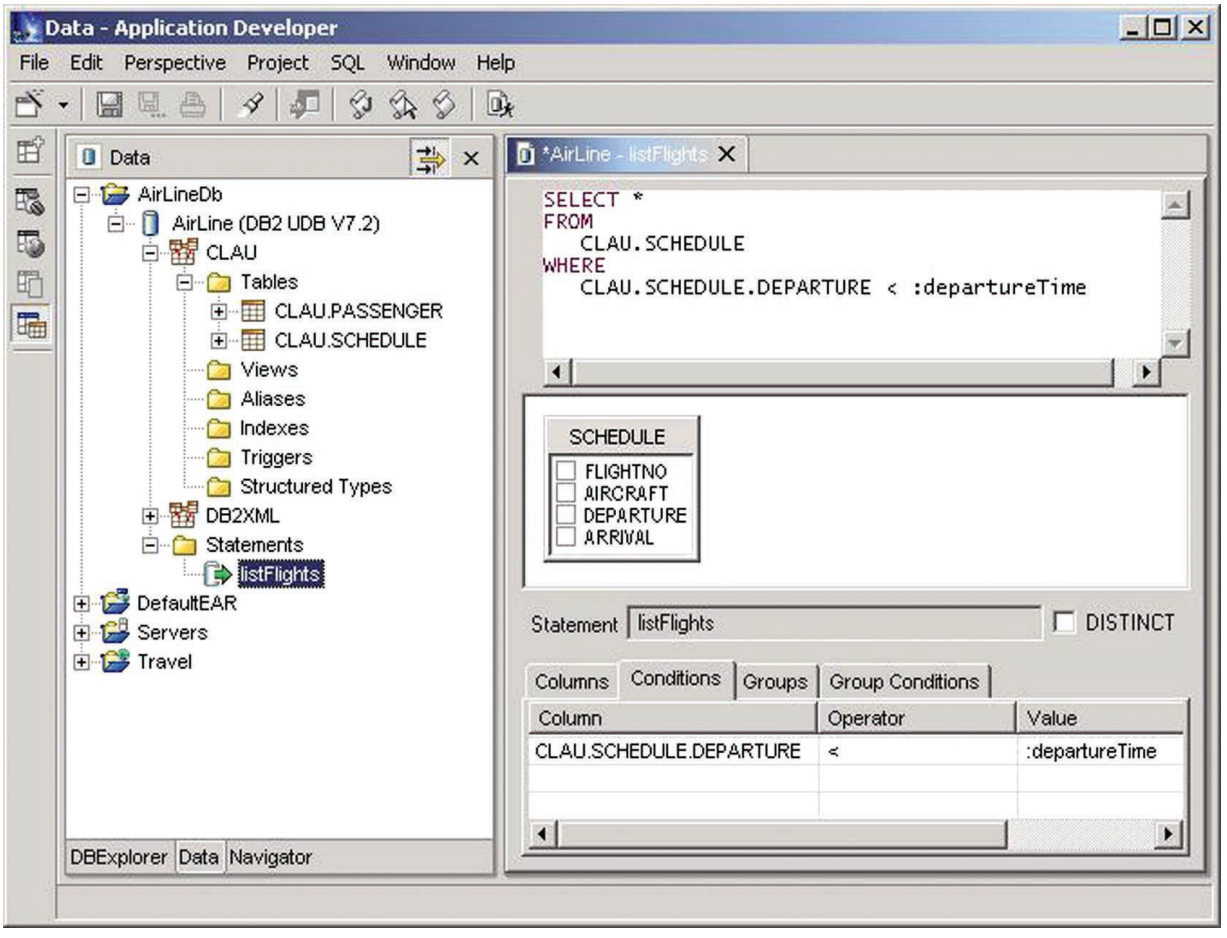


Updating a registry normally requires that the developer register with the node operator and obtain a user identifier and password to be used when logging in. Developers can use the UDDI Explorer to update any UDDI business registry that they have access to. Most node operators, including IBM, provide a public test registry for experimentation. IBM also provides a private UDDI registry that developers can use for local testing.

Scenario 1: Developing the Flights Web service

To illustrate the preceding steps, we now describe a development scenario in which we create a simple

Figure 4 The SQL Query Builder



Web service. The Flights Web service lists flights for an airline. Because it simply retrieves information, it is natural to implement it using a database query. If this Web service involved more complex business logic, we could have implemented it using a Java stored procedure, a Java class, or a session bean. Table 1 lists the main tasks involved, the input and output development artifacts, and the tool used to perform the task.

Set up the Web application. Let us assume that we have created a database named Airline to hold the flight information, and that we have created a Web project for developing the Web service. Our first step is to create a Web service group to hold the services that access the database. We run the *DADX Group Configuration Wizard*, as shown in Figure 3.

The wizard prompts us for the database connection information and other parameters, which are stored in the group.properties file in the directory created for the group. The wizard then updates the Web application deployment descriptor, web.xml, by adding an instance of the DXX invoker servlet to the Web service requests.

Create the operation. The Flights Web service will contain a single operation, listFlights, that lists all the flights that leave before a specified departure time. In general, a Web service contains many operations but we use a single operation here to simplify the example. We use the *SQL Query Builder* to create the SQL SELECT statement and save it to an SQL file, listFlights.sqx. SQL INSERT, UPDATE, and DELETE statements can also be created. Figure 4 shows the

SQL Query Builder, which allows tables to be added using a drag-and-drop style user interface.

Create the Web service. The *XML from SQL Wizard* can generate a DADX file from a set of SQL statements. Figure 5 shows the wizard prompting for operation name, descriptions, and output file name, which in the example is Flights.dadx.

Figure 6 shows the generated DADX file in the *XML Editor*. The editor provides a tree-based design view and a text-based source view. DADX files can be modified in the XML editor or any other text editor.

Deploy the Web service. After the DADX file has been created, it must be deployed to the Web application server. The *Web Service Wizard* handles all deployment tasks. If the Web project is not currently associated with a Web application server, the wizard associates the default server with the project. The wizard installs the SOAP run-time component the first time a Web service is deployed, configures an in-

Figure 5 The XML from SQL Query Wizard

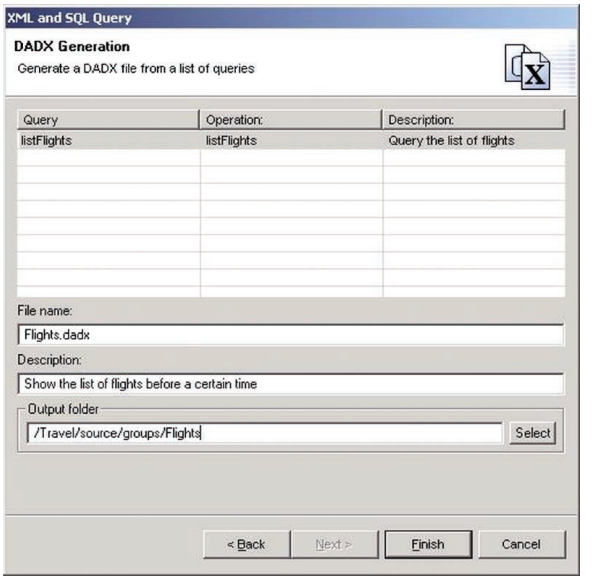


Figure 6 The XML Editor

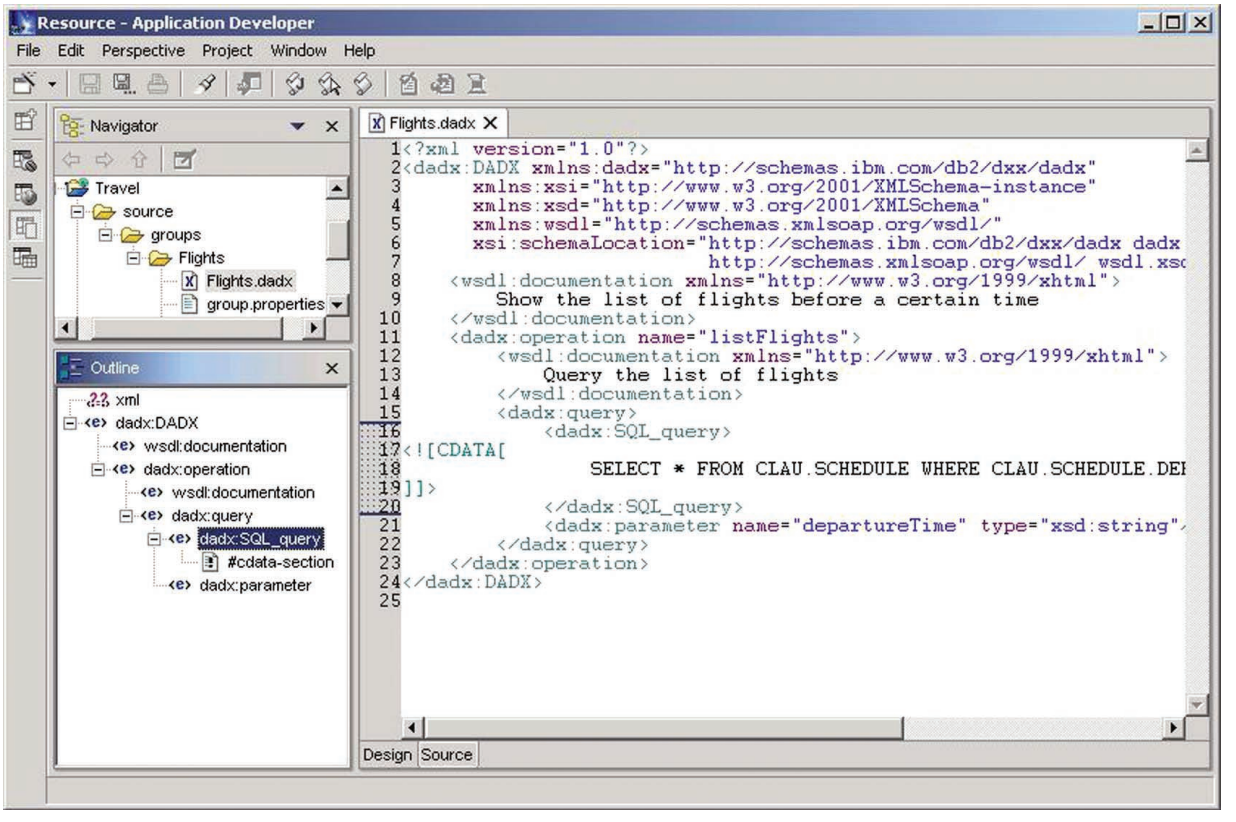


Figure 7 The Web Services Wizard

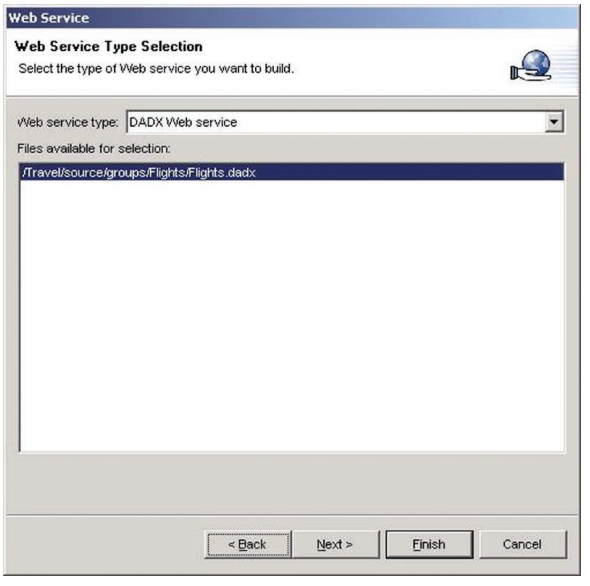
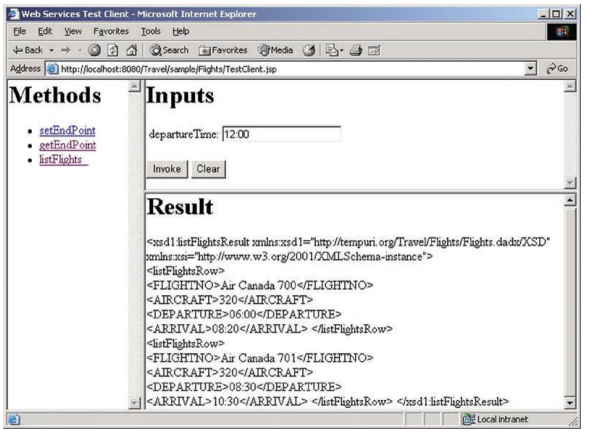


Figure 8 The sample application



stance of the SOAP RPC router servlet if necessary, generates the SOAP deployment descriptor for the service, and updates the deployed services configuration file. The wizard then starts the server, generates the WSDL file for the service, and adds it to the Web application. Figure 7 shows the first page of the wizard.

Test the Web service. The *Web Service Client Wizard* generates a Java client proxy from the WSDL for the service, and a JSP sample application that uses the proxy. Figure 8 shows the sample application that was generated for the Flights service.

The sample application has a methods pane that lists all the operations in the service and allows us to select an operation to test. We have selected the listFlights operation here. The inputs pane of the application prompts us for the input parameters of the operation. Here we have entered 12:00 as the departure time. The result pane shows the output of the operation.

XML development tools

XML plays two distinct roles in Web services development. The first role is that of infrastructure, in which XML is used for data interchange and description. For example, messages are encoded in XML using SOAP and described in XML using WSDL. In this infrastructure role, XML can be completely hidden from the developer. Web services can be developed in standard programming languages, such as Java, marshaling and unmarshaling can be handled by the run-time component, and the WSDL statements can be generated automatically and processed by tools.

As Web services technology matures and becomes widely adopted, it is likely that XML will play a second role, namely that of a programming medium. In this programming role, developers will first be exposed to XML when designing Web services interfaces in a top-down approach. Once developers begin to define data in terms of XML, it will be natural to also specify processing in terms of XML. Current examples of XML programming approaches include XSLT, XML Query, and DB2 XML Extender. To use these new technologies, developers will require new tools.

WebSphere Studio Application Developer includes an integrated suite of tools for many aspects of XML development, such as:

- XML editing and validation
- XML Schema and document type definition (DTD) editing and validation
- Extensible Stylesheet Language (XSL) transformation and tracing
- Generators for creating Java beans from a DTD or XML Schema
- Utilities to generate XML Schema from DTD and *vice versa*
- Utilities to generate XML from DTD or XML Schema and *vice versa*

Table 2 Developing the Passenger List Web Service

Task	Input	Output	Tool
Design the output message format Create an XML Schema to describe the report format. Convert the schema to a DTD for use with the DB2 XML Extender.		passengerList.xsd, passengerList.dtd	XML Schema Editor
Create the operation Map the database tables to the passenger list schema and generate a DAD file for the DB2 XML Extender.	Airline database schema, passengerList.dtd	passengerList.rmx, passengerList.dad	RDB to XML Mapper
Create the Web service Create the DADX file that contains the passengerList operation.	passengerList.dad	passengerList.dadx	XML from SQL Query Wizard
Design the user interface Create a sample HTML instance document and generate a DTD from it. Create a stylesheet to transform the passenger list XML into HTML.	passengerList.xsd	reporthtml.xml, html.dtd, Reportmap.xmx, report.xsl	DTD Editor, XML to XML Mapper
Test the user interface Create a sample input passenger list. Trace the execution of the stylesheet and generate the HTML.	report.xsl	passenger.xml, report.html	XSL Trace Editor

- Tools to integrate XML and relational data, such as generating XML from SQL queries
- Utilities to generate XML schema from table definitions and *vice versa*

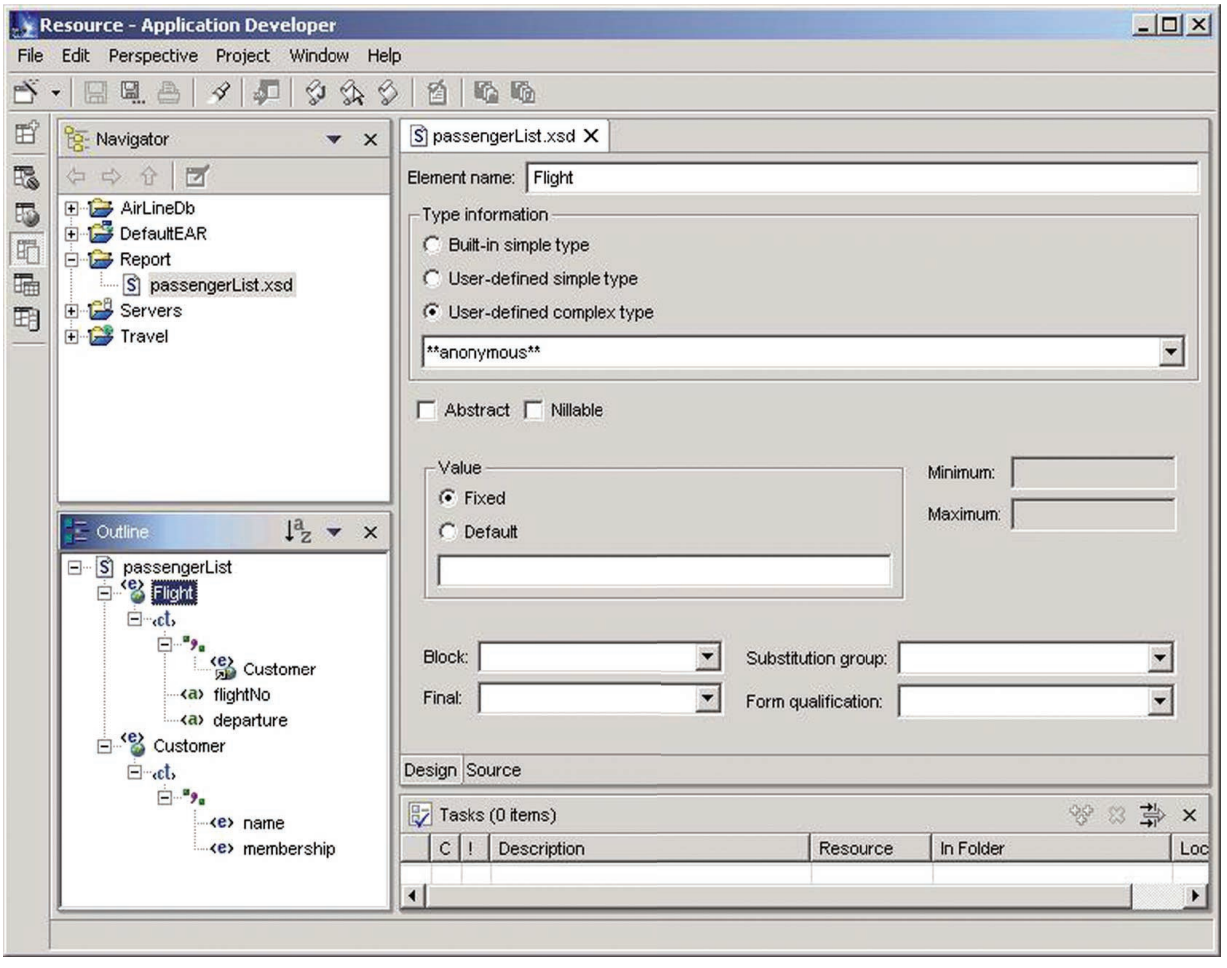
To illustrate how these tools can be used to create a new breed of Web applications based on XML Web services, we extend the scenario described earlier.

Scenario 2: Developing the Passenger List Web service

In this example we create the Passenger List Web service, which lists all the passengers that have tickets for a flight. This service has an operation that returns the passenger list for a specified flight. Here we take a more top-down approach by first designing the format of the output message as an XML Schema. We then map this output schema back to the database and express the mapping as a DAD file that can be executed by the DB2 XML Extender. The DB2 XML Extender allows us to handle XML documents that have complex hierarchical structures. We then work on a user interface to display the output as HyperText Markup Language (HTML) statements. We create an XSL file that transforms the output into HTML and test its behavior using a trace tool. Table 2 summarizes the main tasks in this scenario and lists the input and output development artifacts and the tools used to create them.

Design the output message format. We begin by using the *XML Schema Editor* to create an XSD file that describes the passenger list format. We want the passenger list to include the number and departure time of the flight and to list the name and frequent flyer number for each passenger on the flight. Fig-

Figure 9 The XML Schema Editor



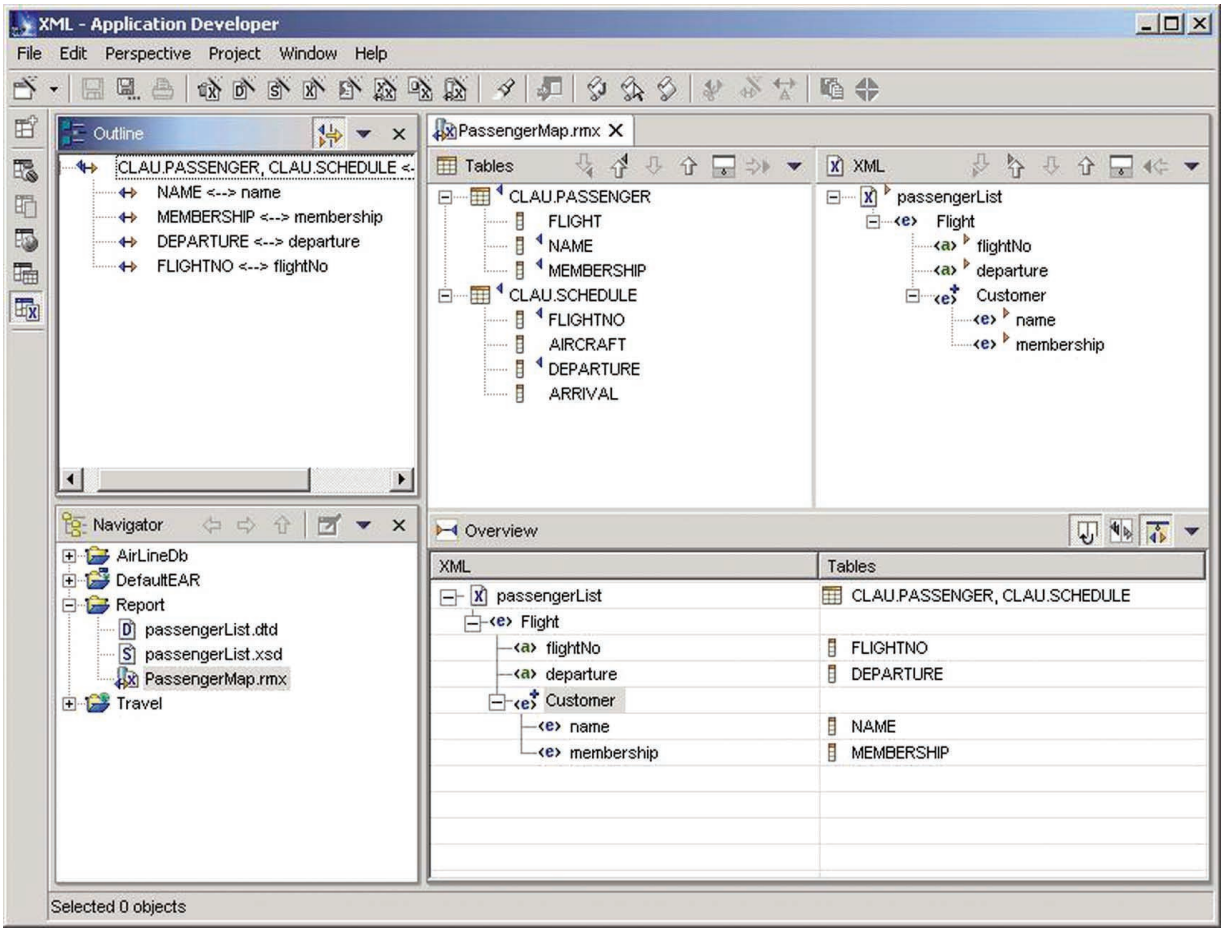
ure 9 shows the XML Schema Editor creating passengerList.xsd.

The XML Schema Editor has a design view and a source view. The design view presents a form-based user interface that is synchronized with the outline view. When a part of the schema is selected in the outline view, a form for modifying it appears in the design view. For example, in Figure 9 the Flight element is selected in the outline view and a form for modifying the element is presented in the design view. Parts of the schema can be added and deleted in the outline view. The outline view and design view eliminate the need for a knowledge of XSD syntax, but developers who prefer to directly edit the file can work with the source view.

In addition to the XSD Editor, WebSphere Studio Application Developer also has the DTD Editor. We refer to both XSD and DTD as schemas. WebSphere Studio Application Developer has extensive support for schemas, including tools for validating schemas, converting between XSD and DTD, generating schemas from sample XML instance documents, generating sample XML instance documents from schemas, and generating Java classes and relational database schemas from XML Schema. In addition, the XML Editor uses schemas to validate instance documents and to provide editing assistance, such as code completion.

Create the operation. Now that we have defined the schema for the output message, we can define the

Figure 10 The RDB-to-XML Mapper



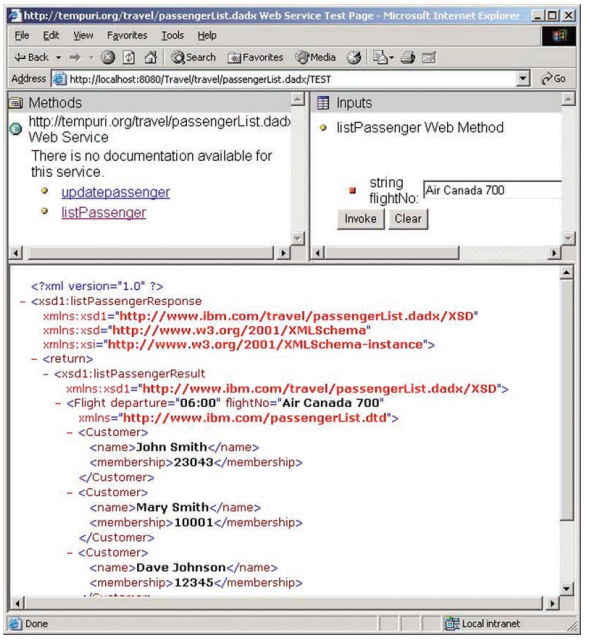
listPassenger operation that retrieves the information from the database. Here we plan to use the DB2 XML Extender run-time component to execute the query and generate the XML result. We must therefore define a DAD file for the retrieval operation. The *RDB-to-XML Mapper* tool, shown in Figure 10, helps us define the mapping from the database to the XML result.

To define the mappings we select the column of the database in the tables view and its corresponding XML element or attribute in the XML view, and then add the mapping to our definition. The complete definition is stored in PassengerMap.rmx, which is an abstract representation of the mapping. The mapper generates a concrete mapping in a DAD file, passengerList.dad, for the DB2 XML Extender.

The RDB-to-XML Mapper is based on a general-purpose mapping framework. Mapping between different data descriptions is a common application development task. For example, the XML-to-XML Mapper is also based on this framework. Other examples include mapping between programming language data structures, for example, from COBOL to Java, and from XML to Java.

Create the Web service. The DB2 XML Extender provides stored procedures that can execute DAD files. Here we want to create a Web service so, as before, we run the XML-from-SQL Wizard to generate a DADX file, passengerList.dadx, but instead of referencing SQL statements we reference the passengerList.dad file. In general, a DADX file can be built from a collection of many SQL statements

Figure 11 The Web service test page



and DAD files. The wizard generates both a retrieval operation, listPassenger, and a storage operation, updatePassenger. The storage operation takes an XML passenger list document as input and updates the database tables. To complete the Web service, we manually edit the DADX file to include a flight number input parameter for the retrieval operation.

We then deploy the DADX file as before using the Web Services Wizard, but instead of generating a Java client proxy and JSP sample application to test the new Web service, we use the *WORF* run-time component to dynamically generate a test and documentation page. Figure 11 shows the dynamically generated test page for the passengerList service.

As in the sample JSP test application, the dynamically generated test page is also divided into three panes. The methods pane lists the operations in the Web service and lets us select one. The inputs pane is a form for the input parameters of the selected operation, and the results pane shows the XML result of the operation.

Design the user interface. Although the output of a Web service is normally consumed by an application for further processing, in some cases it is useful to format the output for presentation to users. In this example, we want to present the passenger list in a Web browser. We begin by manually creating a sample output HTML file, reporthtml.xml, and use the *DTD Editor* to generate a DTD, html.dtd, from reporthtml.xml. Then we use the *XML-to-XML Mapper* tool to define the mapping from the XML passenger list schema, passengerList.xsd, to the HTML page, reporthtml.xml. Figure 12 shows the XML-to-XML Mapper.

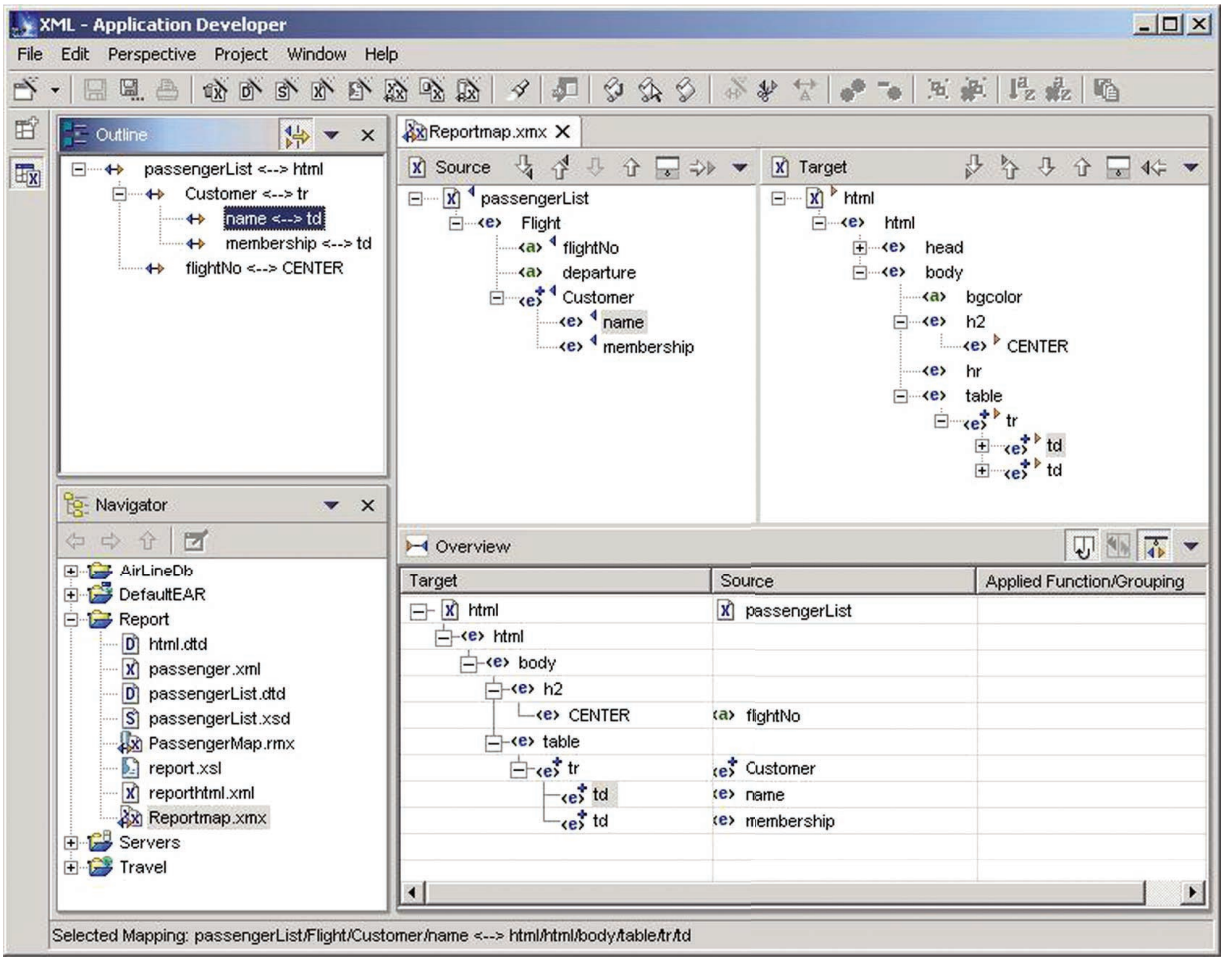
Here we use the mapper to define the correspondence between a source schema and a target sample instance document. We can also use a schema as the target of the mapping. The reason for using a sample document as the target, rather than a schema, is that the full schema for HTML is very complex. It is easier to create a sample of the desired HTML output than to work with the complete HTML schema. Note, however, that mapper requires that the sample HTML document be a valid XML document.

The user interface of this mapper is similar to that of the RDB-to-XML Mapper because they are based on the same mapper framework. The abstract mapping is stored in the Reportmap.xmx file. The mapper can generate the XSTL file, report.xsl, from the abstract mapping. The XSLT file can be applied to the XML passenger list either on the server, for example in a Java servlet, or in the client, using JavaScript, if the browser supports XML parsing and XSLT.

Test the user interface. We complete the testing of the user interface by running the XSL file against a sample passenger list file using the *XSL Trace Editor*, as shown in Figure 13.

We generate the sample input file, passenger.xml, by running the Web service and saving the result. To run the XSL Trace Editor, we select the sample file, passenger.xml, and the XSLT file, report.xsl, in the navigator view, then invoke the trace from the context menu. The trace editor presents the input XML, the input XSL, and the output HTML. We can replay the transform using stepping controls in the toolbar. As the transform is applied, the trace editor highlights each XSL statement and the XML input element it matches, allowing us to understand the transform and resolve any problems. We save the output as report.html. At this point we are confident that the XSLT file is performing as required. To complete development, we must incorporate the XSLT file into our application.

Figure 12 The XML-to-XML Mapper



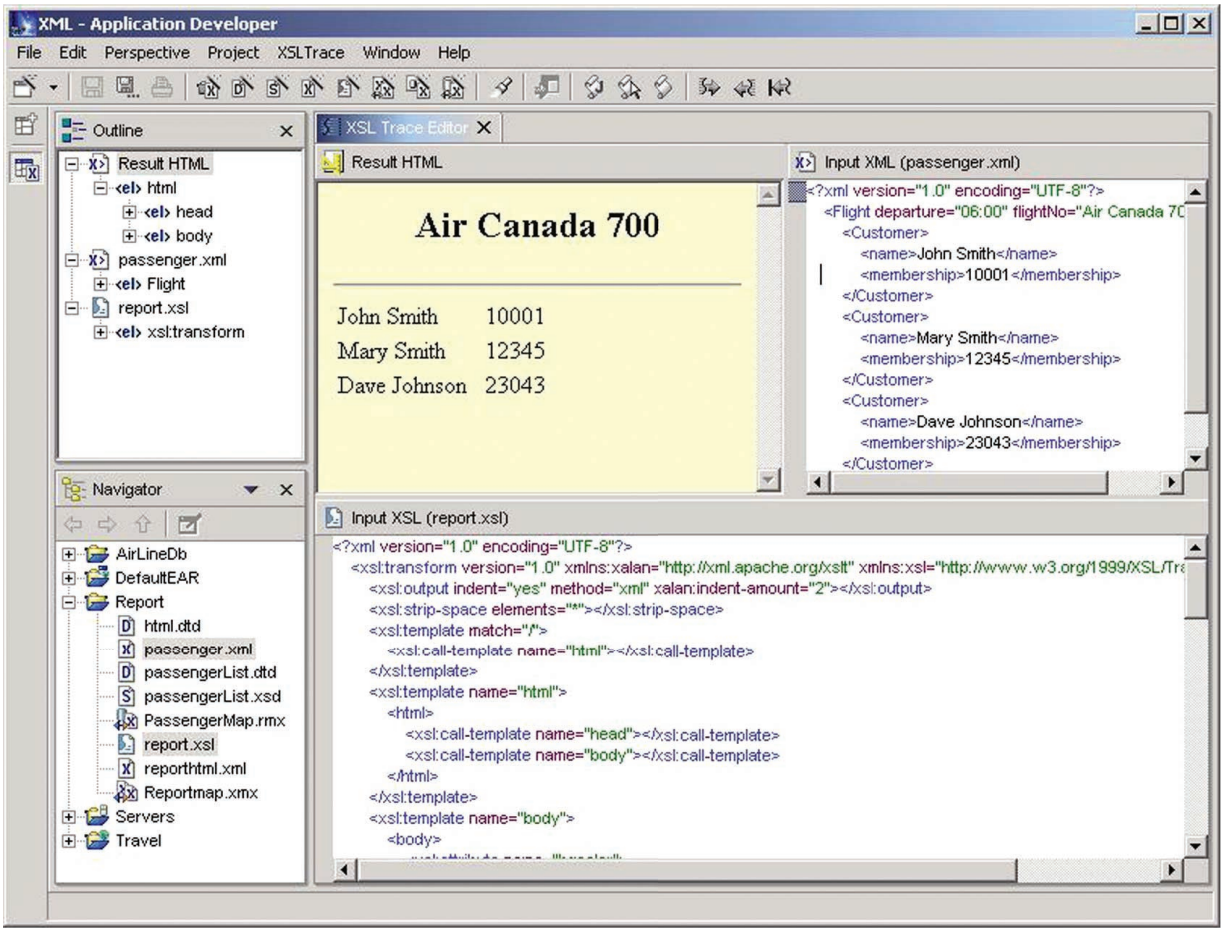
Conclusion

XML Web services provide a powerful new technology for integrating heterogeneous applications over the Internet. WebSphere 4.0 provides a fully supported production-ready deployment environment for Web services based on the Apache SOAP run-time environment. XML plays a central role by providing a data interchange format that is independent of programming languages, operating systems, and hardware. As Web services become more widely adopted, XML will also be used more frequently to specify processing using technologies such as XSLT, DB2 XML Extender, and XML Query.

WebSphere Studio Application Development is a new development environment that supports the full life cycle for Web services development with support for SOAP, WSDL, and UDDI, and includes a powerful suite of XML tools. Using this environment, developers can easily transform existing components, such as Java beans, EJB beans, and SQL statements, into Web services, and can incorporate Web services into new applications.

This paper has described an initial set of Web services and XML development tools. Armed with this information, developers should be able to begin Web services development using WebSphere Studio Application Developer. Over time, developers can expect to see wider coverage of the development process and better integration of the tool suite for this

Figure 13 The XSL Trace Editor



exciting and rapidly maturing distributed programming technology.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of the Object Management Group, Microsoft Corporation, or Sun Microsystems, Inc.

Cited references and notes

1. A. Ryman, *Understanding Web Services* (December 2000), see http://www7.software.ibm.com/vad.nsf/Data/Document4362.
2. Web Services Zone, IBM DeveloperWorks™, see http://www.ibm.com/developerworks/webservices/.
3. *Extensible Markup Language (XML) 1.0*, Second Edition, W3C Recommendation (October 6, 2000), see http://www.w3.org/TR/REC-xml.
4. *XML Schema, W3C Recommendation* (May 2, 2001), Part 1: Structures, see http://www.w3.org/TR/xmlschema-1/, Part 2: Datatypes, see http://www.w3.org/TR/xmlschema-2/.
5. E. Christenson, F. Curbera, G. Meridith, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, W3C Note (March 15, 2001), see http://www.w3.org/TR/wsdl.
6. *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation (November 16, 1999), see http://www.w3.org/TR/xslt.
7. DB2 XML Extender, see http://www.ibm.com/software/data/db2/extenders/xmlext/.
8. XML Query, see http://www.w3.org/XML/Query.
9. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note (May 8, 2000), see http://www.w3.org/TR/SOAP/.
10. Apache SOAP, The Apache XML Project, see http://xml.apache.org/soap/index.html.
11. Tomcat, The Apache Jakarta Project, see http://jakarta.apache.org/tomcat/index.html.
12. *Document Object Model (DOM) Level 2 Core Specification Version 1.0*, W3C Recommendation (November 13, 2000), see http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.
13. Eclipse, see http://eclipse.org/.

14. IBM XML and Web Services Development Environment, see http://www.alphaworks.ibm.com/tech/WSDE.
15. Universal Description, Discovery, and Integration (UDDI), see http://uddi.org/.
16. "Operator" means the organization that operates a UDDI Business Registry node. For example, both IBM and Microsoft operate UDDI nodes.
17. F. Curbera, D. Ehnebuske, and D. Rogers, *Using WSDL in a UDDI Registry 1.05*, UDDI Working Draft Best Practices Document (June 25, 2001), see http://uddi.org/pubs/wsdlbestpractices-V1.05-Open-20010625.pdf.
18. Web Services Listing, XMethods, see http://xmethods.com.
19. WebService Behavior, Microsoft, see http://msdn.microsoft.com/library/default.asp?url=/workshop/author/webservice/overview.asp.

Accepted for publication January 18, 2002.

Christina Lau *IBM Canada, Toronto Laboratory, 8200 Warden Avenue, Markham, Ontario L6G 1C7, Canada (electronic mail: clau@ca.ibm.com).* Ms. Lau is a Senior Technical Staff Member at the IBM Toronto Laboratory and is the development manager for the XML tools in WebSphere Studio Application Developer. Prior to working on XML tools, she worked on Object-Builder for the IBM Component Broker. Ms. Lau is a recognized expert in distributed object development and is the author of *Object-Oriented Programming Using SOM and DSOM*, Van Nostrand Reinhold Publishing Company, New York (1994).

Arthur Ryman *IBM Canada, Toronto Laboratory, 8200 Warden Avenue, Markham, Ontario L6G 1C7, Canada (electronic mail: ryman@ca.ibm.com).* Dr. Ryman is a Senior Technical Staff Member at the IBM Toronto Laboratory and is the architect for the Web services tools in WebSphere Studio Application Developer. Before his work on Web services, he was the architect for VisualAge® for Java, focusing on servlets, JavaServer Pages, and the WebSphere Test Environment. Dr. Ryman is a member of the IBM Academy of Technology and an adjunct professor of computer science in the Faculty of Graduate Studies at York University in Toronto.