WebSphere Studio overview

by F. Budinsky
G. DeCandio
R. Earle
T. Francis
J. Jones
J. Li
M. Nally
C. Nelin
V. Popescu
S. Rich
A. Ryman
T. Wilson

In this paper we provide an overview of IBM WebSphere Studio, a family of tools for developing distributed applications for J2EE™ servers for state-of-the-art information technology systems. In today's business environment such systems are complex, comprise multiple platforms, and make use of a wide range of technologies and standards. Through a representative development scenario we illustrate the way WebSphere Studio satisfies the challenging requirements for a modern integrated development environment. The scenario covers a variety of technologies and standards, including database access, Web services standards, Enterprise JavaBeans™ implementation, integrated application testing, Web page design, and performance optimization. We also describe the Eclipse Modeling Framework, the open source technology base on which WebSphere Studio is built.

Modern n-tiered applications integrate function that is developed by using many different technologies. J2EE** technologies,[1] such as JavaServer Pages** (JSP**), servlets, Java** Database Connectivity (JDBC**) database access, and Enterprise JavaBeans** (EJBs**), are used to develop distributed applications that also use eXtensible Markup Language (XML)[2] for a technology-neutral representation of data, as well as Web services[3] for integrating various applications and platforms.

Application development tools are increasingly required to integrate with an overall application solution that includes tools and runtime environments. Such integration leads to synergy between tools and runtime environments and enhances developers' productivity. Not surprisingly, integrating a rich set of tools within a single user interface is a significant challenge.

To meet this challenge, major software tool vendors rely on a category of software known as an *integrated development environment*, or IDE. IBM's entry in this field is IBM WebSphere Studio,[4] a family of tools for developing applications for the IBM WebSphere Application Server[5] (the IBM middleware platform) and other J2EE servers, Web servers, and database servers. WebSphere* Studio tools run on an open-source extensible base, the Eclipse platform.[6] The tools are available on multiple operating-system platforms, are translated into many languages for worldwide distribution, and are accessible by persons with physical disabilities.

Integrated development environments. In order to support the development of sophisticated n-tiered applications in today's highly competitive application development tools market, IDEs must address a challenging set of requirements, including the following.

Productivity: Developers must be able to rapidly build and test applications. Repetitive processes should be

0018-8670/04/$5.00 © 2004 IBM

simplified. (Other requirements such as integration and reuse contribute toward the main goal of enhancing productivity.)

Integration: The tools must work together to create a seamless development workflow that encompasses design, implementation, testing, analysis, and Web development. They must be able to cope with multiple programming languages and heterogeneous runtime environments.

Flexible, user-friendly interface: The user interface (UI) must be able to mask the full complexity of the IDE, providing novice users with simple access to the core functionality. At the same time, the UI must also support the full range of functionality for expert users.

Shared development model: The tools should be able to share the same development model (sharing development artifacts and an integrated workflow) and should be able to use a different model when it would lead to greater productivity. The development model should support bottom-up or top-down application development as needed.

Reuse: The IDE should provide the ability to reuse complex application artifacts and Web resources (this boosts productivity, increases runtime performance, and reduces storage requirements).

Scalability: The tools must support the development of applications that operate within a wide range of environments. Specifically, for building large-scale J2EE applications, the tools must handle large numbers of application objects and the associated functionality.

Extensibility: The IDE must allow the incorporation of new function and components in a natural way, including tools that are custom-built to fulfill specific user requirements.

Testing: Given today's complex runtime environments, developers must be able to exercise their code without having to publish it, and to test their code on remote servers.

The major IDE products are Microsoft Corporation's Visual Studio** .NET, BEA WebLogic** Workshop, Borland Software Corporation's JBuilder**, Sun Microsystems, Inc.'s ONE Studio, and Oracle Corporation's JDeveloper.

Microsoft Visual Studio .NET is focused entirely on developing applications for the Microsoft .NET platform. Microsoft uses its own tools platform for development of Visual Studio .NET and has been successful in attracting third-party software developers to this platform to enhance and extend the Microsoft tools.[7]

BEA WebLogic Workshop is a development environment for J2EE applications that run on its BEA WebLogic server and is part of BEA WebLogic Platform 8.1. WebLogic Workshop is notable in that it has attempted to reduce the skill requirements for J2EE development by translating features found in Visual Studio. NET, such as application logic controls and source code annotations, into the world of Java. (Many of these features are proprietary.)[8]

Borland JBuilder is a popular IDE for Java and J2EE programming. JBuilder is not aligned with a particular middleware platform and offers support for several, including the market leaders, IBM and BEA. Because of this breadth of support, it does not cover the IBM middleware platform to the same depth or level of integration as WebSphere Studio. The product is implemented on a proprietary tools platform that includes support for third-party extensions.[9]

Sun ONE Studio is an IDE for Java and J2EE programming, focused primarily on support for the Sun ONE Application Server.[10] Sun ONE Studio is implemented on top of an the open-source tools platform NetBeans**,[11] which is offered as an alternative to the Eclipse open-source tools platform.

Oracle JDeveloper, another IDE for Java and J2EE development, is focused on development for the Oracle Application Server, but supports other platforms as well. It is built on a proprietary Oracle tools platform (older versions were built on a tools platform licensed from Borland).[12]

IBM has competed in the IDE marketplace with the VisualAge* family of products since the early 1990s. Since then the application development environment has become increasingly complex. IBM WebSphere Studio emphasizes breadth and depth in its support for the IBM middleware platform, while also providing support for competitive and open-source J2EE servers.

WebSphere Studio approach to IDE requirements. The IBM middleware platform is a rich software environment for the deployment and execution of n-tier applications, based on J2EE with significant IBM extensions. Because of the richness of this platform,

it requires a large and complex programming model. Middleware can be defined as the layer of enabling software that mediates between applications that either run on different platforms, or come from different vendors, or both. Generally, middleware mediates between applications and an operating system. IBM's middleware for Web applications includes the WebSphere Application Server and its extensions, WebSphere MQ messaging products, DB2* and associated database products, Lotus Domino*, and Tivoli* application management products. For the purposes of this article, the programming model includes the programming concepts, application programming interfaces, and file formats used for the development, deployment, and field support of applications running on this platform.

WebSphere Studio represents the result of a multiyear effort by IBM to reinvent its application development tools portfolio in order to address the requirements of its middleware platform. In the late 1990s, it became clear that the current extensive tools portfolio was too fragmented for modern n-tiered applications (which involve a range of diverse technologies). An ambitious project was started to simultaneously develop a new platform for integrating tools (later named Eclipse) and a new generation of IBM application development tools, implemented on top of that platform. The result of that effort is the WebSphere Studio family of products and the Eclipse workbench.

There are two main approaches to the design of application development tools: the *direct-to-middleware* approach and the *model-driven* approach. In the direct-to-middleware approach, the concepts of the underlying middleware execution platform are exposed to the application developer. IDE tools use the middleware APIs and create files in the formats required by the middleware. The direct-to-middleware tools operate almost exclusively on the file formats of artifacts defined by the middleware programming model. They give the application developer full access to the features of the underlying middleware programming model. Well-designed direct-to-middleware tools can significantly simplify programming tasks and reduce the number of concepts and detail a developer is required to deal with. WebSphere Studio takes the direct-to-middleware approach.

The tools that use the model-driven approach exploit more abstract, higher-level concepts than those of the target middleware environment, and then generate the code that runs in the target environment. The goal of tools using the model-driven approach is to allow applications to be defined using concepts closer to the business problem domain than to the information technology domain. Such tools use file formats different from those in the target middleware environment and do not give developers access to all the features of the target middleware. IBM Rational Software Corporation* produces many examples of model-driven development tools.

We believe that the direct-to-middleware approach and the model-driven approach are not competing but rather complementary approaches. Both approaches address the fundamental requirement of enhancing productivity, but for different development circumstances. Direct-to-middleware tools facilitate productivity by potentially reducing the number of new concepts and tasks a programmer is required to understand, especially for development in a familiar middleware environment. The model-driven approach allows an application to be specified at a higher level of abstraction. It can also enable the development of applications from the same specification for different versions of a target middleware platform, or even for different platforms. It is possible to use both kinds of tools together on the same project. In this paper we do not cover tools that follow the model-driven approach.

WebSphere Studio facilitates tight integration with other tools, including model-driven tools (especially, but not exclusively, the IBM Rational Software Corporation tools), by exploiting WebSphere Studio's platform APIs. WebSphere Studio also provides a rich platform for integrating third-party tools. In addition to the UI and source-code-management integration facilities offered by the Eclipse workbench, WebSphere Studio uses the Eclipse Modeling Framework to provide a common technology for the implementation of shared in-memory object models and API libraries. This technology enables the manipulation of all the concepts and file formats of the WebSphere platform.

In an IDE in which each tool is responsible for reading, writing and manipulating the objects and files required by the middleware platform, the integration of the various tools takes place at the file level, and is relatively loose and coarse-grained. In contrast, WebSphere Studio provides a rich set of API abstractions over a wide range of files, which simplify the implementation of associated tools and provide a more tightly integrated experience for the de-

veloper. Tools integrating with WebSphere Studio can share a single memory copy of a middleware artifact, and tools integration can occur at the finer-grained level of the individual objects stored within the files. Use of the Eclipse Modeling Framework provides a powerful approach to the fundamental requirements of enhancing productivity and enabling integration. A more detailed discussion of the framework is found near the end of this paper.

WebSphere Studio is not a single product, but rather a family of products. WebSphere Studio Site Developer (or Site Developer, for short) is intended for Web application developers and includes tools for relational database access, Web services, Web page design, performance profiling, and testing. WebSphere Studio Application Developer (or Application Developer, for short), is intended for J2EE developers and contains all the functions in Site Developer. In addition it supports EJB development. The development scenario that anchors this paper focuses on the functionality in these two core products. (See "A development scenario" later in this paper.)

The WebSphere Studio family of products also contains advanced extensions for specific categories of users. These extensions adhere to the same UI architecture as the core products and include much of the same functionality. WebSphere Studio Application Developer Integration Edition extends Application Developer with features aimed at developers of business integration applications. WebSphere Studio Enterprise Developer enables development for the entire range of IBM mainframe environments. WebSphere Development Studio Client addresses the needs of IBM eServer* iSeries* developers and those integrating iSeries components into distributed applications. These products are beyond the scope of this paper.

The rest of the paper is structured as follows. In the next section, we start with an extended overview of the user interface, which illustrates key mechanisms for the integration of tools and workflow. This leads into our chosen development scenario that exercises different aspects of WebSphere Studio functionality. Whereas many different scenarios are possible, the chosen scenario demonstrates many of the product's most useful functions in an integrated workflow. Next, we describe the Eclipse Modeling Framework, which provides the basis for WebSphere Studio's file sharing among tools. We conclude with a look at future challenges for IDEs, and WebSphere Studio in particular.

WebSphere Studio UI

In this section we describe some of the UI design features of the WebSphere Studio family. Because the UI is the mechanism that integrates WebSphere Studio's large number of tools, it provides the context for understanding the product's overall concepts and usage.

WebSphere Studio provides a user interface that is attractive and easy to use and that can be used in different combinations in a wide variety of development scenarios. This requires that a simple user experience be available for simple tasks while allowing expert users access to the full power of the underlying programming model. In order to satisfy this requirement, WebSphere Studio provides several nested levels of integration. We consider them in their logical order, building up from the simplest to the most general (note that the typical user experience might involve the higher level integration first).

Development of the WebSphere Studio products followed the User-Centered Design (UCD) methodology.[13] This methodology places the user at the center of all design decisions. It includes a range of specialized techniques to acquire feedback from representative samples of users during all stages of design and development. User testing sessions were conducted to understand the tasks that users typically perform and the problems they experience with their tasks, to iteratively evaluate the high-level and low-level design of the products, and to assess the products' design against user experience with the prime competitors' design.

The results from these activities indicated that users would like to see less clutter in the workbench UI and better UI support for task flows in the tools. This user feedback led us to further refine the design of perspectives and cheat sheets (described later) to increase user productivity and the task flow integration among our products.

The UI techniques described in this article provide WebSphere Studio with layered integration that pulls together plug-ins into a single tool suite and unifies user task flows. In order to encourage third-party plug-ins to exploit these various UI integration mech-

anisms in a consistent way, WebSphere Studio adheres to the Eclipse UI guidelines, produced for designers and developers of all Eclipse-based products.[6] The Eclipse platform that underlies WebSphere Studio is very flexible and extensible, but the platform framework can only enforce UI consistency among the registered components to a certain extent. The UI guidelines provide platform-specific guidance and best practices for plug-ins to consistently and tightly integrate into the workbench.

We describe below the various UI mechanisms used in WebSphere Studio that enable layered integration and provide an integrated user experience on top of the J2EE programming model.

Menus, toolbars, action sets, and extension points. In addition to the core functionality of its workbench platform, WebSphere Studio is designed to be extensible by *plug-ins*, which can contribute new views, editors, wizards, menus, and tool items to the platform. Plug-ins can be open-source tools or third-party vendor tools. Although each plug-in is implemented as a separate component in the platform, UI mechanisms such as merged menus, context menus, toolbars, and action sets (as well as perspectives and cheat sheets, to be discussed later) give the platform's UI the illusion of seamlessly integrated components, as shown in Figure 1.

Each workbench window contains a menu bar and a toolbar. These are prepopulated by the platform, but a plug-in can add items to each. This is accomplished by using an *action set*, a set of task-oriented actions that users can show or hide. The actions within an action set are often distributed throughout the window's menu bar and toolbar. For example, by using the Window → Customize Perspective dialog, users can enable the EJB action set to have EJB development-related actions available in the menu and toolbar.

In addition to supporting the standard content assistance in any of its several editors, the WebSphere Studio workbench also provides UI extension points, such as Quick Fixes and Quick Assist, to further enhance the user experience. For example, users can click on the light-bulb marker on the lefthand margin of an editor for suggestions on how to fix problems with the source code.

These kinds of UI extension points give users a consistent and context-sensitive way to perform tasks. The workbench has well-integrated support for the popular open-source build tool ANT,[14] which allows developers to automate repetitive tasks. Other external tools, such as word processors and spreadsheets, can also be integrated with the workbench. The UI extension points in WebSphere Studio pull together an integrated tool suite that works as a single entity and helps users be more efficient in accomplishing their J2EE application development tasks.

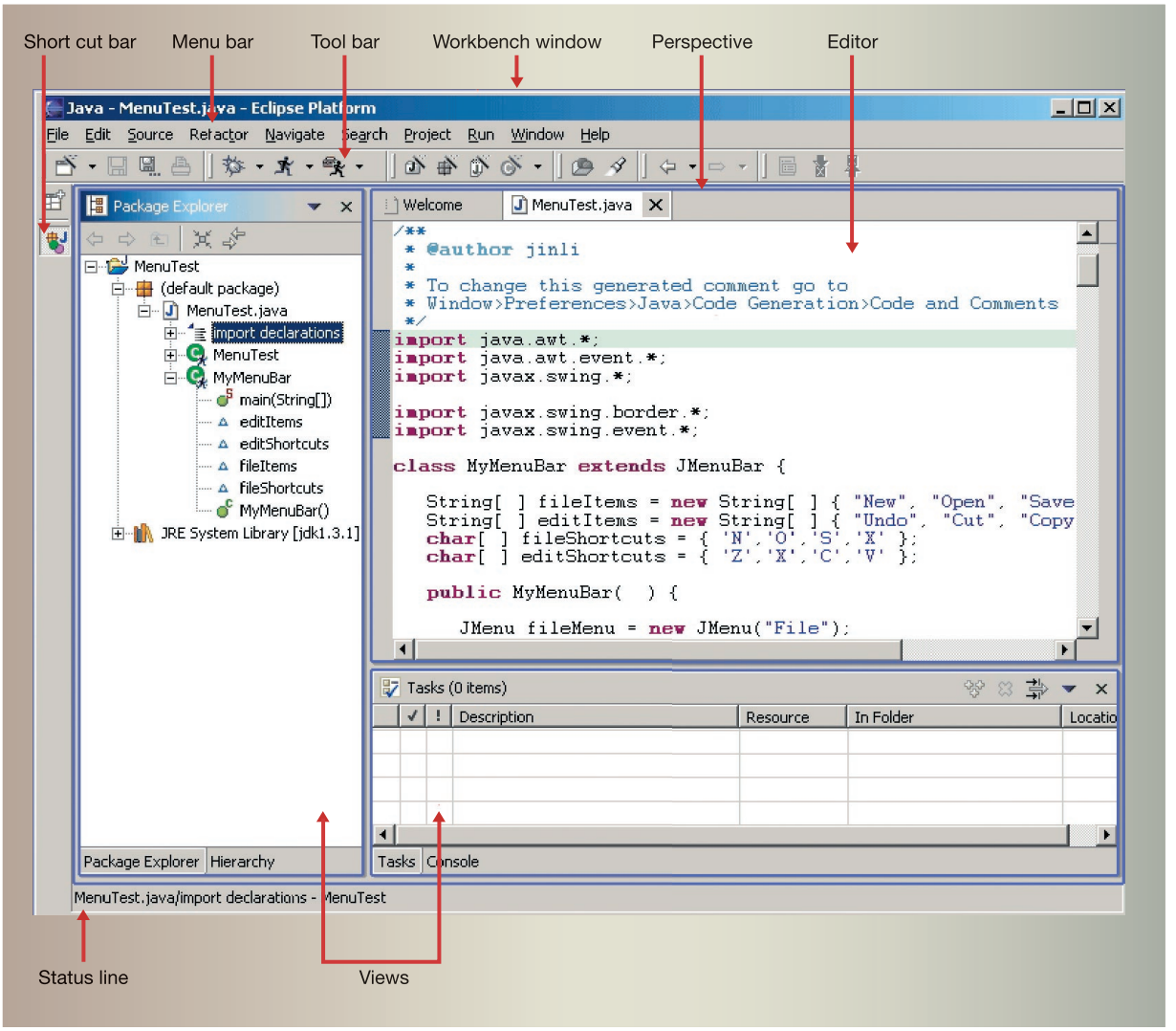
Editors and supporting views. In WebSphere Studio, text editors are used to interact with entities of interest, which may be documents or data objects. The focus of attention is a reflection of the task at hand. For example, when the task is to create, edit, and debug Java code, the focus is the Java code, and an editor is used to interact with that code.

WebSphere Studio's next level of UI integration comes in the relationship between the editors and the additional views provided to support the editors, primarily the Navigator, Outline, and Properties views. Whenever an editor is active, an outline of its content is displayed in the Outline view (Figure 1). The outline is used to navigate through the edit data or interact with the edit data at a higher level of abstraction. For example, if a user opens EJB implementation code in an editor, the structure of the class is displayed in the Outline view. Then, if a method or field is selected in the Outline view, the text declaration of that item is revealed and selected in the editor. If a user selects a method or field and invokes its context menu, the user can interact with the selected item as a conceptual unit, rather than just a sequence of characters.

The interaction between editor and view is two-way, with the editor driving the content of the supporting views. The editor and supporting views contribute user task-related actions to the menu bar, toolbar and context menu. The supporting views can then leverage these action contributions to further enhance the in-context user experience. For instance, in an EJB editing scenario, users can select methods in the Outline view and invoke an action to promote the methods to be remote interfaces.

Not all views are associated with editors. Some views provide information about the overall state of the workbench or its contents, for example, the Java package explorer view (see Figure 1), which shows a logical view of the Java packages and classes contained in files in the workbench. Other examples are the J2EE navigator and J2EE hierarchy views, which

Figure 1 Typical workbench configuration



provide a simplified, logical view of the J2EE file structures.

Dealing with J2EE file formats in an IDE like WebSphere Studio presents some design challenges. J2EE defines a rich, complex model for the formats of files that are executed on a J2EE application server. Files associated with Web user interfaces are structured in a particular directory structure inside an archive file called a *Web application archive* (WAR) file. Similarly, EJBs are defined by particular file and directory structures within EJB module archive files. WAR files and EJB modules are, in turn, nested inside an *enterprise application archive* (EAR) file. It is desirable to allow developers access to all of the detail of the files and directory structures in order to allow exploitation of all the available features of the application server. However, experience with previous development environments within IBM has shown that using an alternative, tools-based model for resources causes problems, particularly integration with other tools that use the application server's file structures directly and with the application server's own capabilities, such as the administrative console

and problem determination tools. On the other hand, exposing the details of these files and their structures directly, as in WebSphere Studio's standard Navigator view, makes for an intimidating experience, especially for simple scenarios and novice users.

For this reason, WebSphere Studio offers the J2EE hierarchy view as an alternative to the J2EE navigator. The J2EE hierarchy view shows a simplified, logical view of the project, directory, and file structure. It also shows logical J2EE information that is encoded in the contents of the files themselves, such as EJB and servlet definitions. Together, the J2EE navigator and hierarchy views fulfill the requirement of providing simplified views of J2EE while allowing developers access to the full detail of the J2EE artifacts.

Perspectives. The fundamental building blocks underlying the UI of any product based on the Eclipse architecture are editors, views, and action sets. Typically, editors allow the user to create and modify resources; views allow the user to navigate among a set of resources or see alternate logical presentations of those resources, and action sets allow the user to invoke groups of operations against these resources. Because Eclipse is an open system, it has no limit on the number of editors, views, and action sets that are integrated at any one time. Furthermore, no single component has control over the integrated UI, so that if left unchecked, it could easily become a chaotic collection of bit parts. To create an organizing principle that builds on the previous layers of UI integration, WebSphere Studio introduces the concept of *perspectives*. Perspectives provide the UI with order and integration by governing the set of editors, views, and action sets visible at any one time.

Three of the more commonly used perspectives are the J2EE perspective, the Web perspective, and the server perspective. At any given time, only one perspective can be visible in a workbench window (although the user may have several windows open in a workspace). The selected perspective determines the views that are visible, the spatial arrangement of these views, the default location of the editors when invoked, as well as the actions available on the window's menu and toolbar. Of the more than 80 views that exist in WebSphere Studio Application Developer, only eight are initially presented in the J2EE perspective; and of the more than 50 action sets that can be added to the menu and toolbar, only 16 are added by default.

The product itself determines the set of views and action sets to pull together within a perspective, as well as the number of perspectives to provide. The overriding principle is that WebSphere Studio perspectives encapsulate sets of views and actions that allow users to complete major application development tasks without having to switch their working context. Consequently, WebSphere Studio provides perspectives targeted to particular user roles (e.g., Web developer), to distinct phases of the development life cycle (e.g., profiling and logging), and to specialized resource types (e.g., data). The individual user is able to customize most aspects of a perspective, as well as create entirely new perspectives, but the design point is to provide a default collection that is immediately useful "out of the box."

Perspectives are also responsible for one final layer of order and integration, control over tool selection; for example, controlling the list of perspectives to which users might switch from their current one, controlling the list of views users might load into their current perspective, and controlling the list of wizards users might launch from the current perspective. Of all the possible entries (for wizards, that number runs into the hundreds), WebSphere Studio is designed to select those ten or so that are most useful to the task at hand (or, in the case of perspectives, the set to which users are most likely to switch as they proceed to their next major task). As with views, editors, and action sets, these lists are customizable.

Cheat sheets. WebSphere Studio includes many instances of long-running macro-tasks in which numerous smaller tasks must be completed. These macro-tasks often span multiple component areas, requiring the use of any number of wizards, editors, and views. Often, the smaller tasks that make up a macro-task need to be completed in a specific order, require repetitive data entry, and require context-related information to be maintained as the user traverses the various wizards, editors, and views. In the past, such macro-tasks have been extremely difficult for new users to complete or understand. To help new users through these long-running tasks, especially when the task sequence is non-intuitive, WebSphere Studio introduces a form of user assistance known as *cheat sheets* (Figure 2).

Cheat sheets list a linear sequence of steps where each step contains a step title, a short textual overview of the step, and an action link that invokes a wizard, an editor, or a view. Each step additionally

includes a visual indication of step completion, a link to the help system for further details, and, optionally, the ability to repeat or skip a step if appropriate. In other words, the cheat sheet is a form of user assistance similar to a traditional help system, except that it is interactive and highly integrated with the rest of the UI. The state of the cheat sheet is preserved if the user needs to close it (or even the product) midway through a task. The cheat sheet capabilities support macro-tasks that might take many minutes, if not hours, to complete and enable WebSphere Studio to support many more substantive task flows than can be done with wizards alone. Wizards are ideally suited to integrating a series of micro-tasks under a single unifying umbrella UI, but their strength is in supporting tasks that typically take no more than a couple of minutes to perform. Cheat sheets take task support well beyond those bounds. They offer a way of scaling up traditional user assistance to match the increased complexity of the IDE itself.

A development scenario

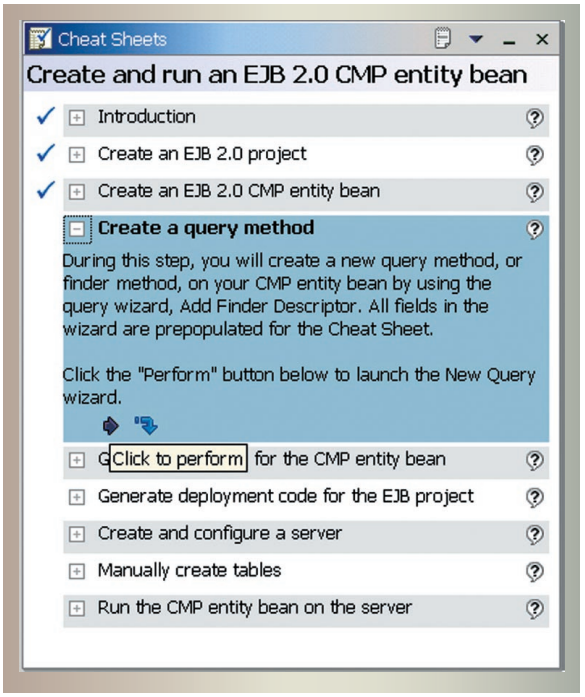
Now that we have introduced the WebSphere Studio UI, we will examine a development scenario that makes use of many of the UI features and the associated tools. The scenario is designed to introduce some of the functions of WebSphere Studio Application Developer for creating J2EE applications. The scenario has been greatly simplified to focus on representative capabilities and techniques. More detailed information is available in the extensive tutorial and reference information provided with the WebSphere Studio products.

In this scenario, a fictitious manufacturing company, Global Widgets, has decided to deploy an enterprise application based on Web services so that client applications running on application servers, personal computers, personal digital assistants (PDAs), and cell phones can access information stored in the corporate Human Resources (HR) database. Web services have been chosen so that a single programming interface to the HR database can support this large variety of client applications, thereby reducing costs and speeding development. The scenario also develops a Web user interface application using Struts[15] and JSP technologies.

The scenario covers the following application development phases:

1. Developing database reporting functions and publishing them as Web services.

Figure 2 Cheat sheet for creating an EJB



   This phase introduces the WebSphere Studio SQL query building and debugging tools, and shows how SQL queries and stored procedures can be published as Web services.

2. Defining a Web service that performs updates to the data.

   This phase shows how to define a more complex database interaction as an EJB implementation that can also be published as a Web service.

3. Creating the EJB implementation.

   This phase uses WebSphere Studio's UML** (Unified Modeling Language) Visualizer and Java Editor to create the code for the EJB session bean defined in the previous phase.

4. Testing and debugging the application.

   This phase explores the features of the Server Test Facility and shows how to use them to test the EJB session bean.

5. Testing for WS-I conformance and publishing to a UDDI (Universal Description, Discovery, and Integration) registry.

   This phase tests the application for conformance to the Web Services Interoperability (WS-I) organization's Basic Profile and shows how to publish the completed Web service to a company's internal Web service registry (UDDI).

6. Creating the Web application.

   This phase creates a robust user interface to access the newly developed Web service.

7. Analyzing and optimizing performance.

   The final phase uses the Performance Profiling tool to identify any performance bottlenecks in the completed application.

Note that the WebSphere middleware platform and the corresponding WebSphere Studio tools offer many options for application architecture and application deployment topologies, most of which we cannot cover in this paper. In this scenario, we choose to follow a single representative topology that provides some idea of the range of WebSphere Studio's capabilities. We develop a set of Web services using a combination of WebSphere and IBM DB2 technologies, and use the Web service interfaces we create in the development of our Web UI (as alternatives in our UI implementation, we could have used DB2 access, stored procedures, or EJBs directly). The best choice of architecture for a particular application depends on many factors, and the interested reader can consult Reference 16.

Creating data access reporting functions and publishing them as Web services. The initial phases of the Global Widgets scenario highlight WebSphere Studio support for data access functions. Database support covers a wide range of products, including IBM DB2, IBM Informix*, IDS Server, Oracle Database, Microsoft SQL Server, and others. This support includes, among other things, creating tables and views and sampling their data, accessing DB2 federated data, graphically building and executing SQL statements, and developing and testing DB2 Java and SQL stored procedures and deploying them in J2EE applications. The scenario is based on the DB2 SAMPLE database that contains sample data for a number of fictitious departments and their employees. Associated with each employee there is an ID number, a name, a salary, an education level, and a department. Departments have an ID number, a name, and a manager.

This phase of the scenario uses WebSphere Studio tools to develop data access reporting functions and then publishes those functions as Web services, thus making them easily available through the wide variety of devices and applications that Global Widgets supports. The reporting functions produce the list of all departments, the list of all employees in a given department, and the management chain for a given employee. These reporting functions are implemented bottom-up, using SQL queries and a DB2 SQL procedure, and then deployed as DB2 Web services.

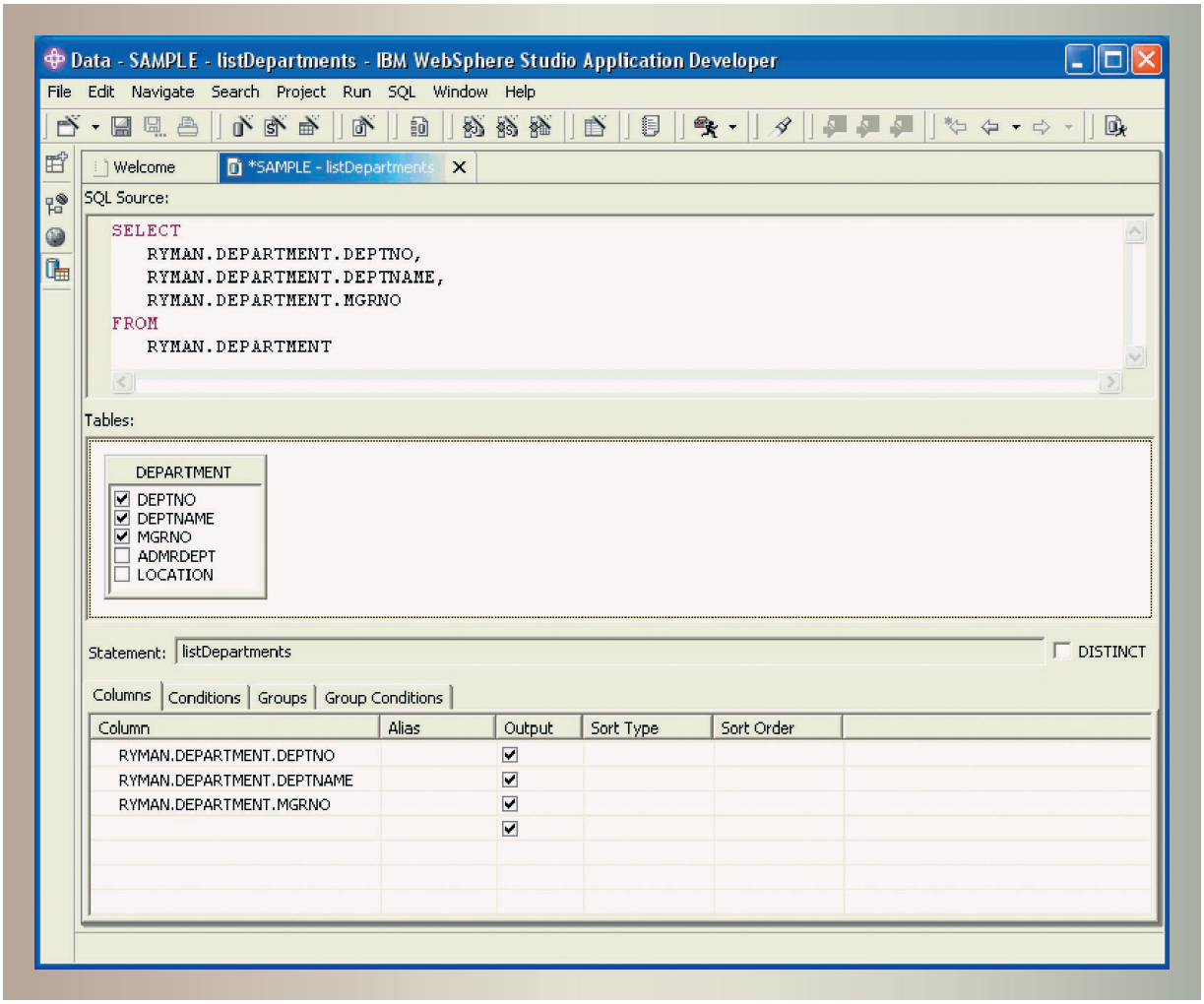
The reporting functions are implemented by directly accessing the database. Two of the functions, list all departments and list all employees in a department, are simple queries implemented as SQL SELECT statements. The third function, list the management chain for an employee, is a more complex query and is implemented as a DB2 SQL stored procedure. These functions are then deployed as a Web service. This illustrates a bottom-up approach to Web service creation.

A few simple steps with minimal coding are all that is required to begin development of these functions. After launching Application Developer and creating a new dynamic Web project, named HR, the developer opens the data perspective, creates a connection to the DB2 Universal Database* V8.1 SAMPLE database, and imports the database into the project. The next step is to open the SQL Query Builder, using the pop-up menu in the Statement folder. The query builder enables users to build SQL statements graphically by selecting tables and columns from the database, linking them to specify JOIN operations, and building expressions to qualify the data to be retrieved.

Figure 3 shows a query defined against the SAMPLE database, which itself was built with only a few mouse clicks. The data returned by this query will be displayed in the Department List page of the client application. While this example shows a relatively simple query, the graphical query builder can be used to build extremely sophisticated SQL statements. The query can also be executed directly within the editor to verify that it returns the intended data.

The statement that returns the list of employees would be defined in a similar manner. Because the

Figure 3 Query defined against SAMPLE database



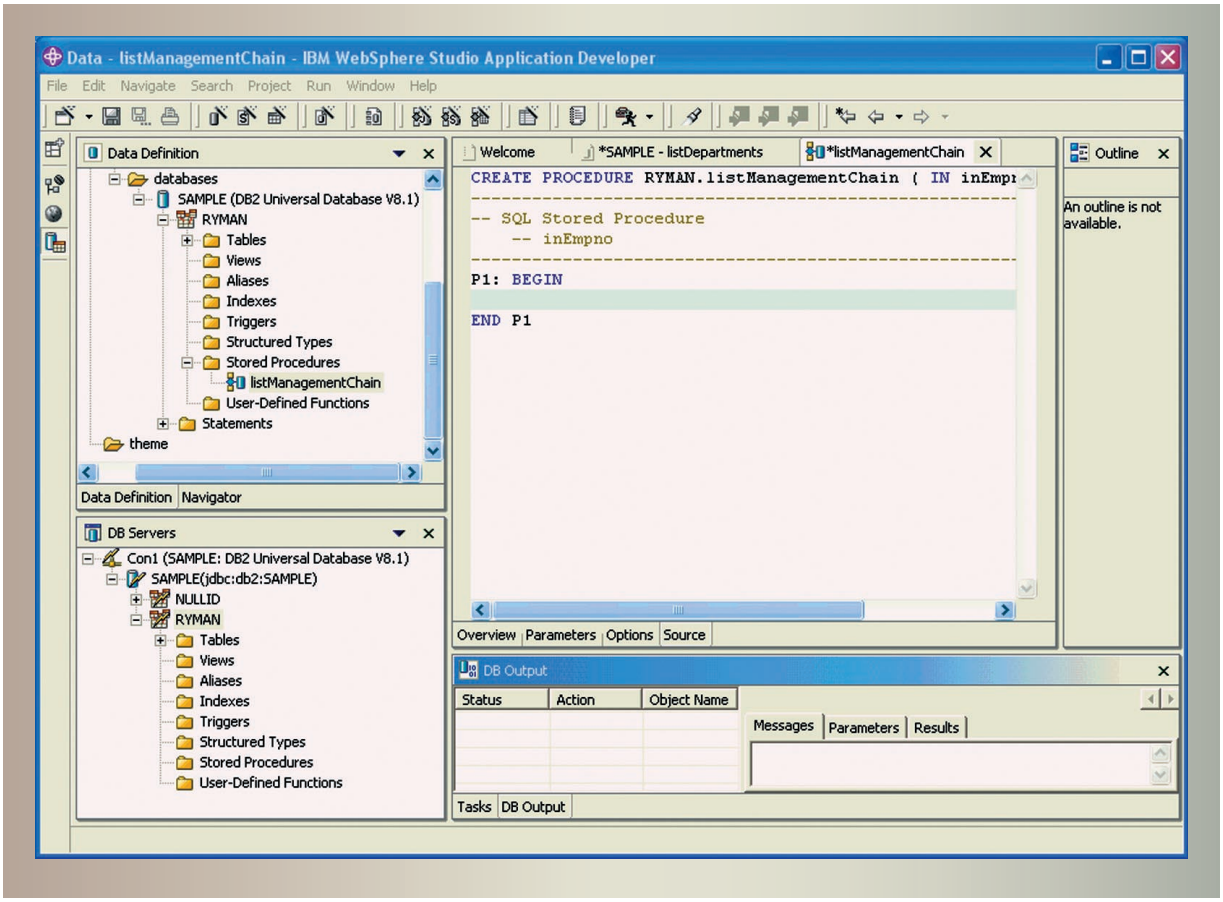
department whose employees will be listed must be passed as a parameter to the query, this statement would illustrate the tools support for defining queries that contain *host variables* (the standard SQL term for a formal parameter in a query, the value of which is set at execution time). When this type of query is executed by using the query builder, the user is prompted for values for any required host variables.

Our third statement, which lists the management chain, is implemented as a stored procedure that runs directly in the database. This provides an important security benefit: the database administrator can authorize users to execute the stored procedure and get results without having to authorize the same users to view all of the data accessed by the stored procedure. Users can be granted authority to call the listManagementChain procedure and get management chain information from the EMPLOYEE table, without being able to view the sensitive salary data stored in the same table.

WebSphere Studio provides sophisticated tools to assist with the development of stored procedures, including wizards, editors, and debugging capability. Figure 4 shows the WebSphere Studio desktop with the source code editor for creating SQL procedures.

Stored procedures need to be uploaded and installed into DB2 in order to run. WebSphere Studio takes

Figure 4 Source code editor for SQL procedures



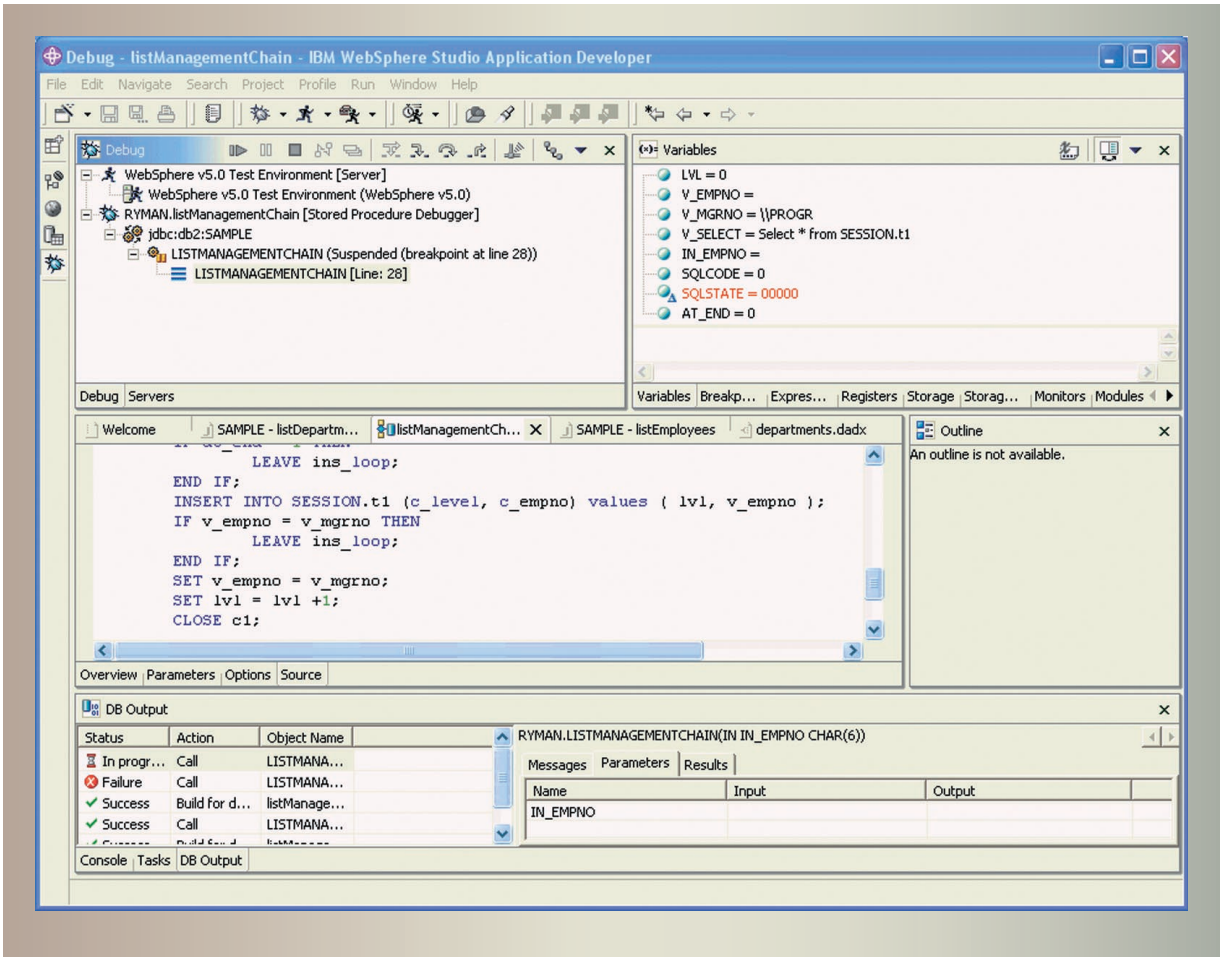
care of these mechanics with a single click. In addition, WebSphere Studio offers source-level debugging capability for SQL procedures. Figure 5 shows our SQL procedure being debugged.

An important architectural aspect of WebSphere Studio is that many components are shared across different features of the product. For example, the model of the database catalog that was imported in this scenario, as well as the SQL editor and other user interface components, are also used in JSP and EJB development scenarios.

WebSphere and DB2 provide the ability to publish SQL statements and SQL and Java stored procedures as Web services in a simple, declarative manner. WebSphere Studio wizards simplify the task of creating the configuration files necessary for publishing DB2 Web services. A single WebSphere Studio wizard captures the specification of the database connection and maps the Web service operation to the database operation. The wizard then deploys the service to a test server and can optionally start the test server. The built-in test page enables us to test our Web service without needing to develop a custom Web application, as illustrated in Figure 6.

Defining a Web service that performs updates to the data. Now that the data reporting functions have been created and published as Web services, the next phase of the scenario is the development of an admin Web service that allows applications to perform administrative functions, such as updating the name of an employee or a department, updating salaries and education levels, transferring an employee from one department to another, and creating or delet-

Figure 5 Debugging an SQL procedure

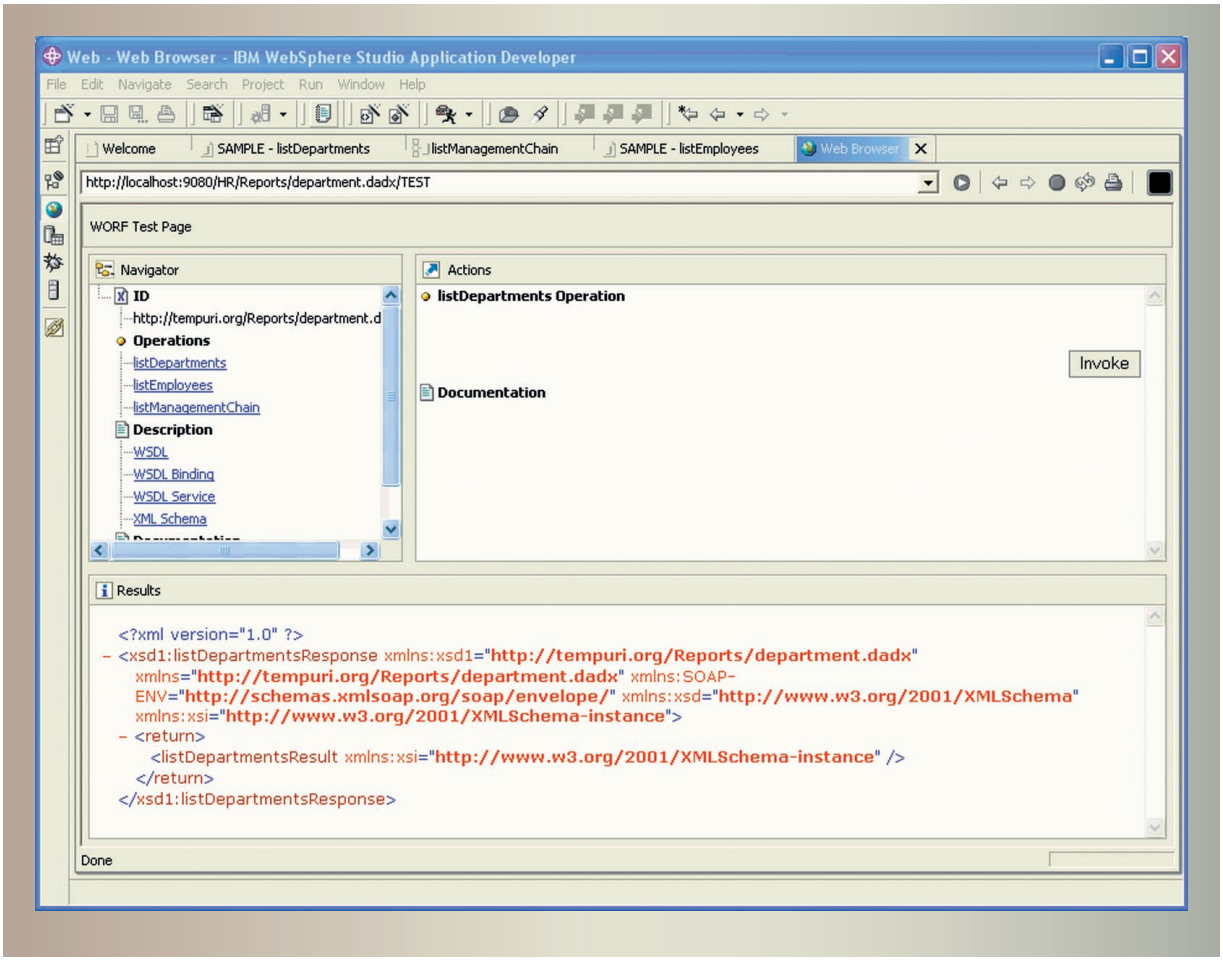


ing an employee or department. The admin Web service is more complex than the reporting Web services in that it must enforce specified authorization rules. For example, only a department manager may perform a salary change or employee transfer. In addition, the admin Web service must respond to invalid input parameters by returning detailed fault messages. These considerations lead to the choice of EJBs as the implementation technology, instead of SQL UPDATE, INSERT and DELETE statements. The Web service will be implemented as a stateless session EJB that will access the database using Department and Employee entity EJBs.

EJB-based Web services may be developed using either a bottom-up or a top-down approach. In the top-down approach a Web Services Description Language (WSDL)[3] document is developed first and a skeleton session EJB is generated from it. The methods of the generated session EJB are then coded. In the bottom-up approach, a session EJB is developed first and is then deployed as a Web service, with the WSDL document being generated from the session EJB. For a further discussion of these development approaches, see Reference 17.

The bottom-up approach is the fastest way to get started with Web services because it builds on traditional programming skills. However, the top-down approach has several important benefits, including improved interoperability. In Web services, WSDL acts as the programmatic contract between clients

Figure 6 Web service output



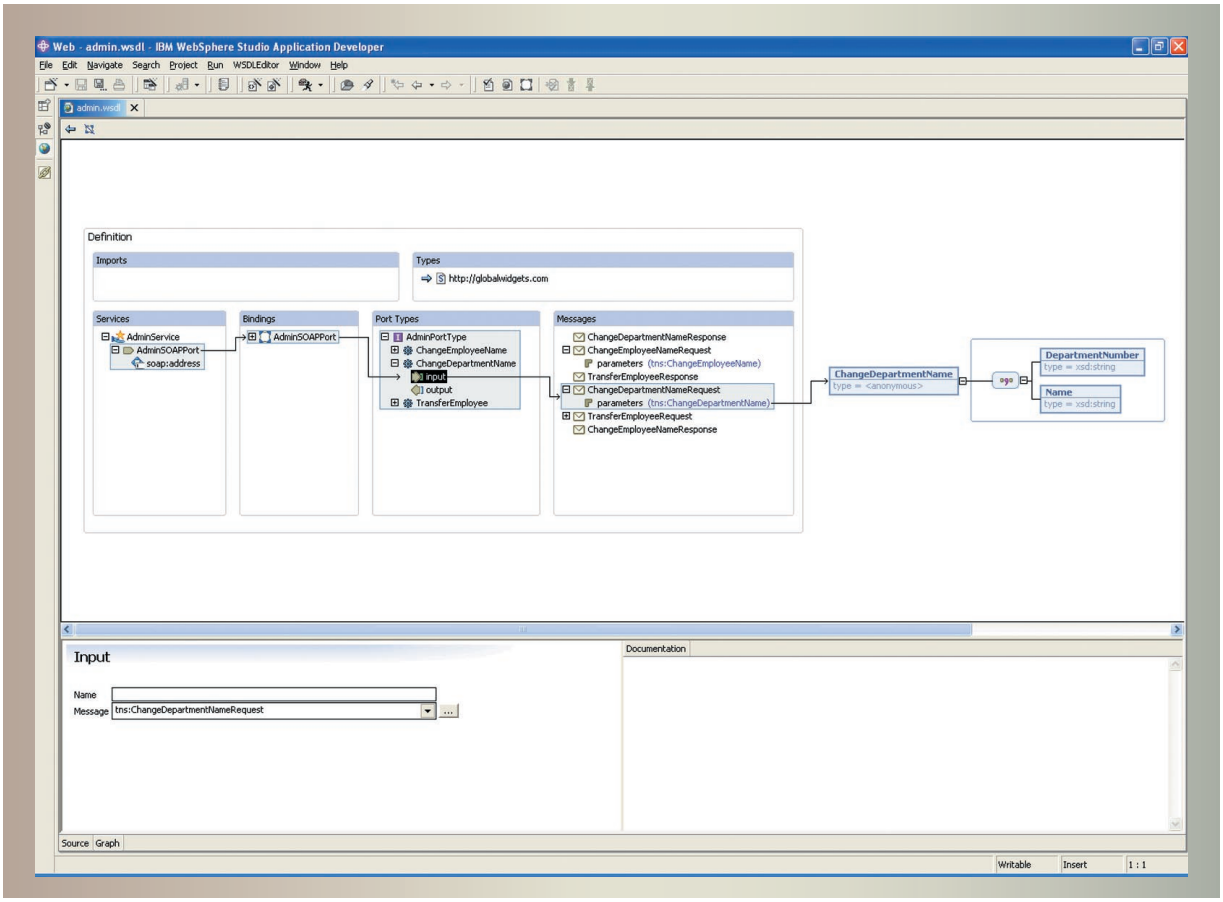
and services. In the top-down approach, WSDL is a central development artifact, being the source for generation of both service skeletons and client proxies.

Industry experience over the last few years has shown that the best way to achieve interoperability is to describe Web services as an exchange of XML documents, which, when using SOAP (Simple Object Access Protocol), is referred to as the *document/literal style*. WebSphere Studio includes many new features that directly support document/literal style Web services, such as top-down creation of WSDL, validation of WSDL, generation of HTML (HyperText Markup Language) documentation from WSDL, and conformance-checking of both WSDL documents and SOAP messages against the Web Services Interoperability Organization Basic Profile (WS-I BP). The Basic Profile specifies versions, clarifications, and constraints for SOAP, WSDL, and UDDI. For additional information on these Web services standards see References 18 and 19.

For the Global Widgets scenario, we start top-down development of the admin Web service by creating a new WSDL document in project HR, admin.wsdl, and then edit it with the WSDL editor as follows:

- We create a new port type for the service and add a new operation for each of the administrative functions in the service, for example, ChangeDepart-

Figure 7 Graph view of WSDL editor



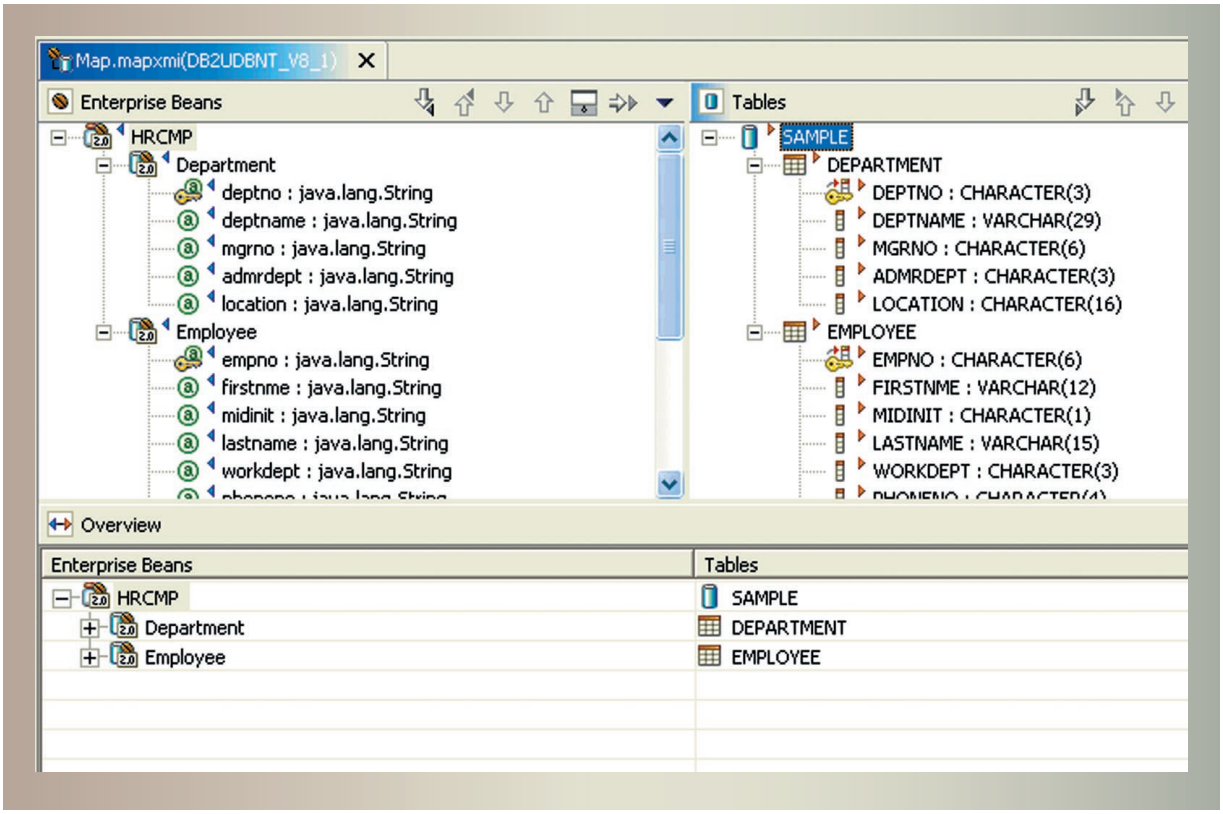
mentName, ChangeEmployeeName, and TransferEmployee.

- We create an input and an output message for each operation and, optionally, some faults.
- We specify the format of the messages and faults using the XML Schema Description Language (XSD) editor integrated into the WSDL editor.
- We complete the WSDL document by creating a service element that contains a single port, binding the port type using document/literal style SOAP over HTTP.

Figure 7 shows an example of creating a WSDL document. Our document, admin.wsdl, is open in the graph view of the WSDL editor. The input message of the ChangeDepartmentName operation is selected and its XSD definition is expanded to show the input parameters, DepartmentNumber and Name. In general, a message format may be an arbitrarily complex XSD document. In this example, the message is merely a wrapper for some simple parameters, and the message type is referred to as the *wrapped document/literal* style.

Now that the WSDL document has been created, the next step is to generate a skeleton EJB Web service based on the WSDL document. At the same time we will generate a Java client proxy and sample JSP test client in order to exercise the service. The Web Service wizard can perform all the required tasks including creating the EJB project, HR EJB; generating the skeleton stateless session EJB, deployment code, and all deployment descriptors; starting the service in the WebSphere test environment; and generating the client proxy and JSP test client.

Figure 8 Map browser



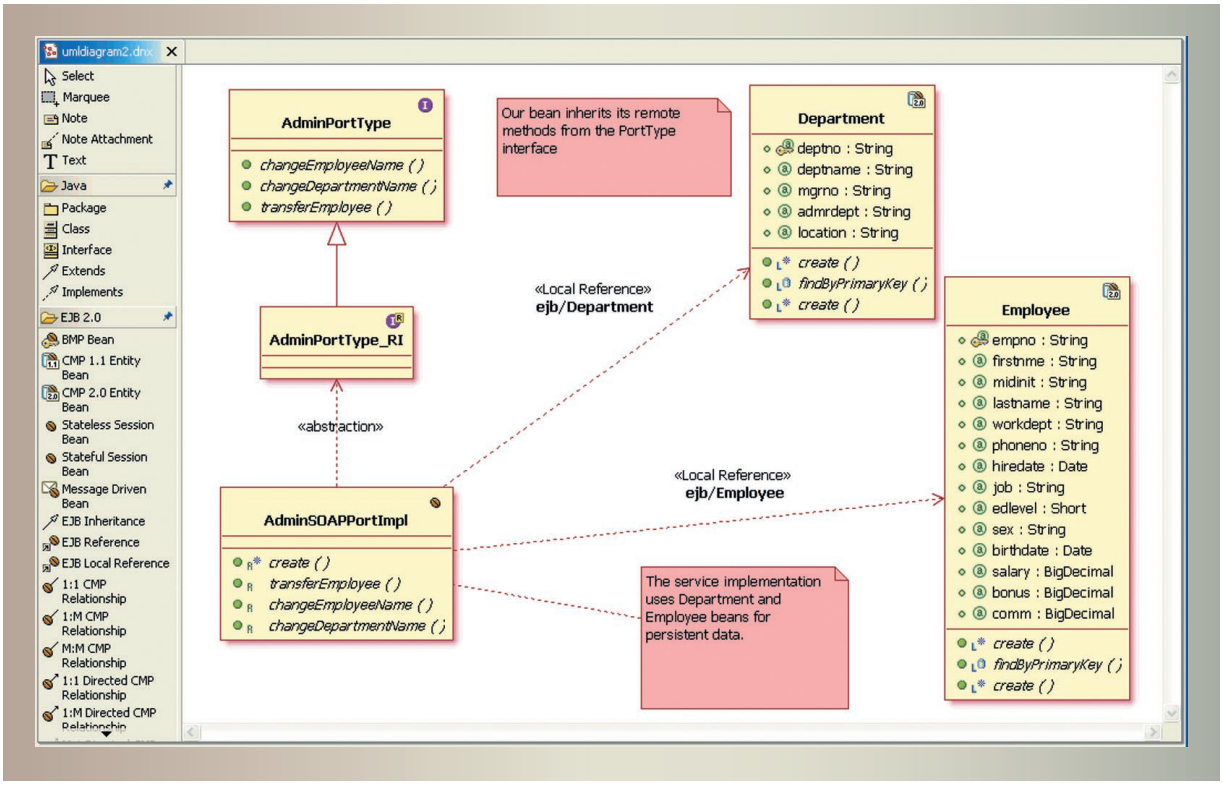
After this is completed, we have a running Web service and can test it using the JSP client. However, the service does nothing because it does not yet include any business logic. Implementing the business logic calls for WebSphere Studio's Java development tools and normal EJB programming. The following phases of the scenario introduce some of the tools that are available to simplify the job of developing and testing the EJBs. See Reference 20 for more information on Web services tools in WebSphere Studio.

Creating the EJB implementation. Now that we have created a session bean skeleton to implement our admin Web service, we can use the EJB tools to complete the service implementation. Using a session bean for the service implementation gives access to the services provided by the EJB container, such as management of transactions to ensure data integrity and security to allow control over which users have access to HR functions.

The admin service will work with the data from the EMPLOYEE and DEPARTMENT tables. In the EJB programming model, entity beans provide access to persistent data of this kind. WebSphere Studio provides tools to generate container-managed persistence (CMP) entity beans from table definitions. In particular, the EJB-to-RDB mapping wizard creates entities using a *bottom-up map*. This generates CMPs that provide a direct representation of the underlying tables to our HR EJB project. The wizard generates the two CMPs required by the admin service and automatically maps them to the EMPLOYEE and DEPARTMENT tables. We can view the detailed results using the map browser, as shown in Figure 8. Now that the two CMPs have been generated, we can implement the admin bean by using simple EJB/Java APIs on the CMPs to manipulate the data.

The mapping wizard automatically uses local-only entities (a best-practice behavior). Because only the

Figure 9 UML visualizer with EJB class diagram



session bean can be remote, we are able to maintain encapsulation of the persistent data. Because the CMPs are local entities, we must define local references in order for the session bean to work with them. These references can be created graphically using WebSphere Studio's UML Visualizer. The only steps required are to add the three beans to a new class diagram, and then use the Visualizer palette to draw the two local references. The resulting EJB class diagram is shown in Figure 9.

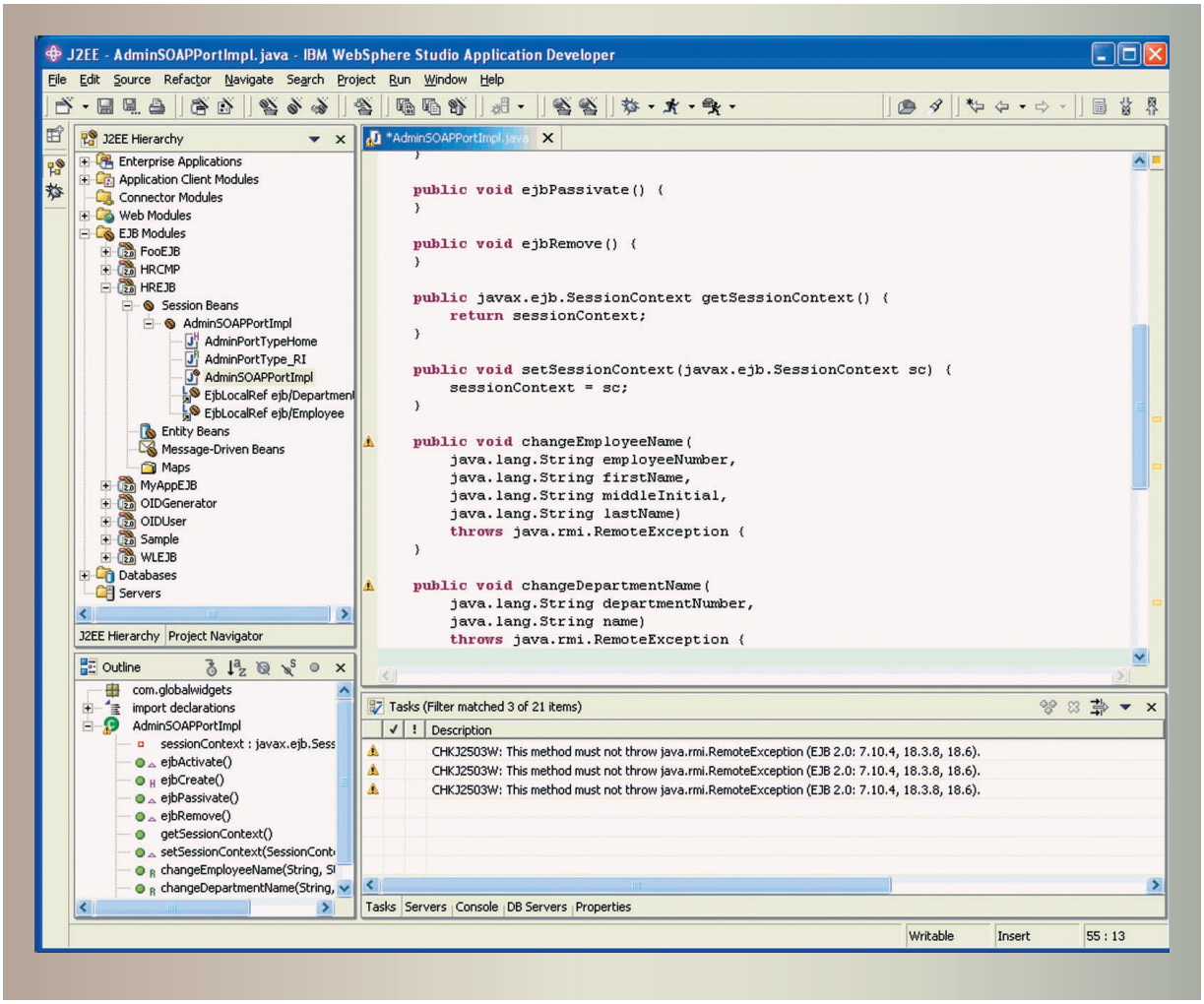
The UML Visualizer provides powerful graphical editing capabilities for EJBs. For example, Figure 9 also shows annotations that we added to show the inheritance among the generated Java interfaces. The UML Visualizer uses UML object-modeling notation that has been standardized by the Object Management Group, Inc.[21]

The diagram created by the UML Visualizer is saved in the admin EJB project and can be shared with the development team along with the other development artifacts. The Visualizer has no additional model data that needs to be synchronized with the EJB deployment descriptor and supporting Java source. Because its visualization is derived dynamically from those sources, it is always in synchronization. Changes made in the EJB deployment descriptor editor or Java editor are reflected immediately in any class diagram, and vice versa.

Once the local references have been created for the entity beans in the admin session bean implementation, the developer can switch to the Java editor to complete the implementation. The body of the changeDepartmentName method can use a standard coding pattern for JNDI (Java Naming and Directory Interface) lookup of the Department home via the ejb/Department local reference. Figure 10 shows how the developer would complete the method using the Java editor.

The WebSphere Studio Java editor is extended with features to assist EJB developers. For example, the

Figure 10 Session bean in Java editor

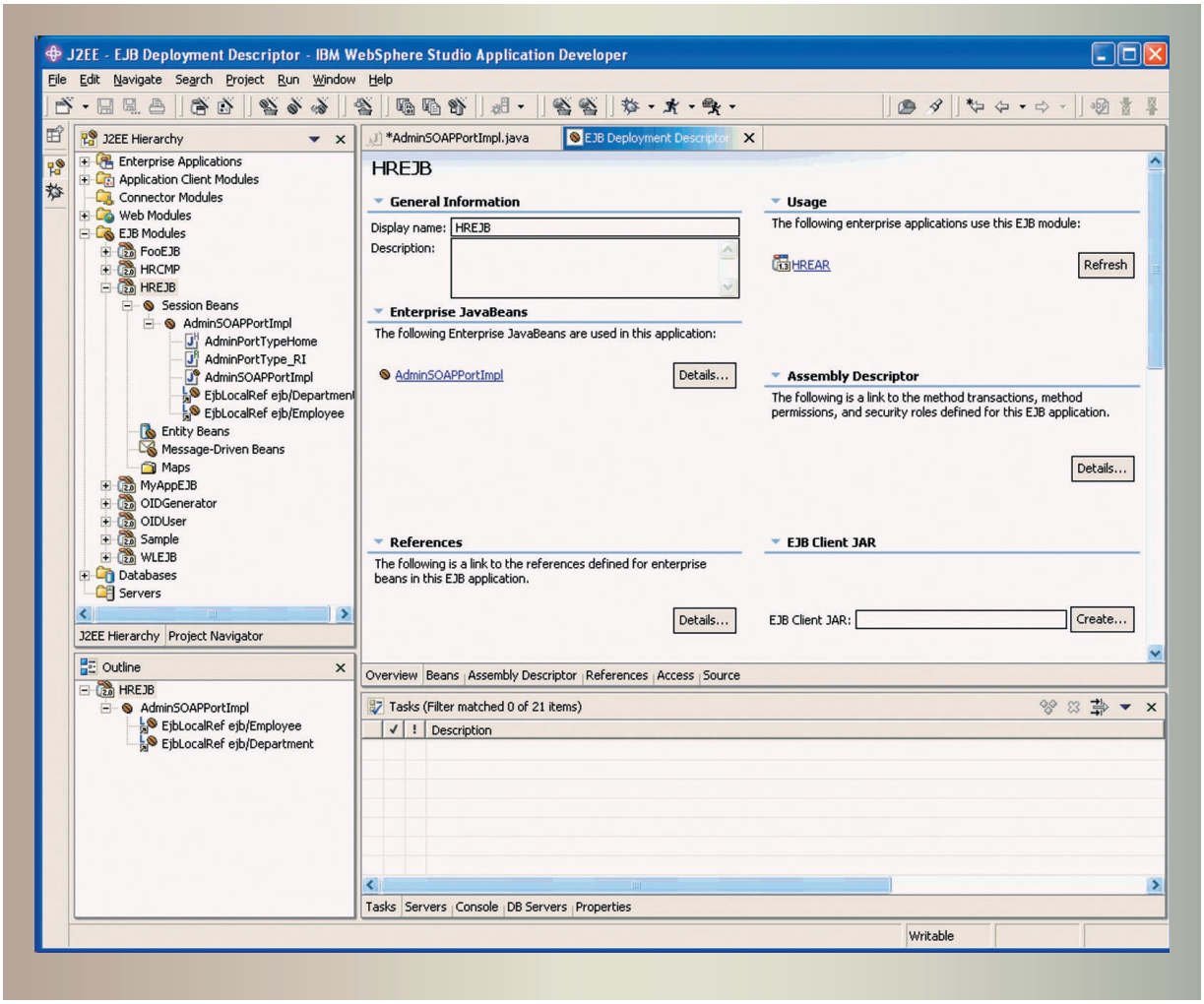


method outline adds decoration to distinguish home and remote/local interface methods, and provides an Enterprise Bean submenu to facilitate adding and removing implementation methods from the appropriate interfaces. The task list is extended to provide feedback on the correctness of EJB and other J2EE artifacts along with feedback on the Java compilation process.

This scenario shows a basic path through the WebSphere Studio EJB tools and illustrates how WebSphere Studio simplifies the task of building and developing applications with EJBs. The capability for using a scenario of this kind is important for initial developer experience with the products, as well as for programmer productivity. It is equally important, even for experienced developers, that WebSphere Studio provides deep and detailed control over every aspect of EJB development defined in the J2EE specification and over the application server extensions to these capabilities. One example of this is the multipage custom editor provided to assist with viewing and modifying EJB deployment descriptors, illustrated in Figure 11.

In the EJB phase of the scenario, we completed the implementation of the admin bean (the AdminSOAPPortImpl session bean) and connected it to

Figure 11 Multipage custom editor



the database using entity beans. The next step is testing and debugging the Web service application.

Testing and debugging the application. We have already made references to the ability to simply and immediately test a function by executing it on a WebSphere Application Server. In each case, we have done this by selecting a simple menu item or checkbox in a wizard, without installing, configuring or manually starting the application server, and without packaging and installing the application containing the function. This capability is provided by the Server Test Facility. Testing the admin bean for the Global Widgets scenario requires the use of the Universal Test Client (UTC), which is the most complex component of the Server Test Facility. To understand its usage, we will take an extended look at the Server Test Facility as a whole and at the test environments that it supports.

The Server Test Facility. This facility allows the user to explicitly manage and edit server configurations as part of the development environment and also automatically creates default server configurations for applications under development. This allows for immediate testing without the need for an explicit configuration step. The Server Test Facility supports multiple application servers including various ver-

sions of WebSphere Application Server, BEA WebLogic, and Apache Tomcat.

In addition to being able to create a single default test server configuration that supports the simple test experience described above, the test environment allows the developer to explicitly create and manage multiple test server configurations and server instance definitions. Because the files that form the server configuration are saved in the source code management system, complex configurations can be shared between team members to help with consistent, repeatable testing.

Server configurations and server instances provide a very flexible test environment, but with this support comes the inherent complexity of having multiple servers, server configurations, and associated projects. Rather than requiring developers to learn the entire structure just to test a simple application, WebSphere Studio provides a staged introduction to the test environment that starts with simple menu selections and wizards. If a project is not already associated with a server configuration, a new configuration is created for it. If the configuration is not associated with a server instance, a new instance is created. Finally, the server is started with the configuration and displayed in a browser integrated into the workbench. Other server configuration details, such as the definition of data sources required for CMP beans, the generation of EJB deployment code, and even relational database table creation, are handled as part of this process.

In addition to the support for this simple experience, WebSphere Studio Server Test Facility also provides extensive tooling for expert developers who need to control each and every detail of their server configurations, such as the Server Test Facility multipage custom editor. Further discussion and additional details on the WebSphere Studio Server Test Facility are available in Reference 22.

WebSphere Unit Test Environment. This environment is a key implementation of the Server Test Facility. It consists of a complete instance of the WebSphere Application Server, which is shipped and installed with the Studio development environment. Because the server is tightly coupled with the desktop, the applications under development can be easily executed and tested without the overhead of publishing the application or installing it into a separate server. Studio includes multiple versions of the application server, in order to allow testing of applications that target an older version of the runtime. It is even possible to apply WebSphere Application Server fixes to the test environment.

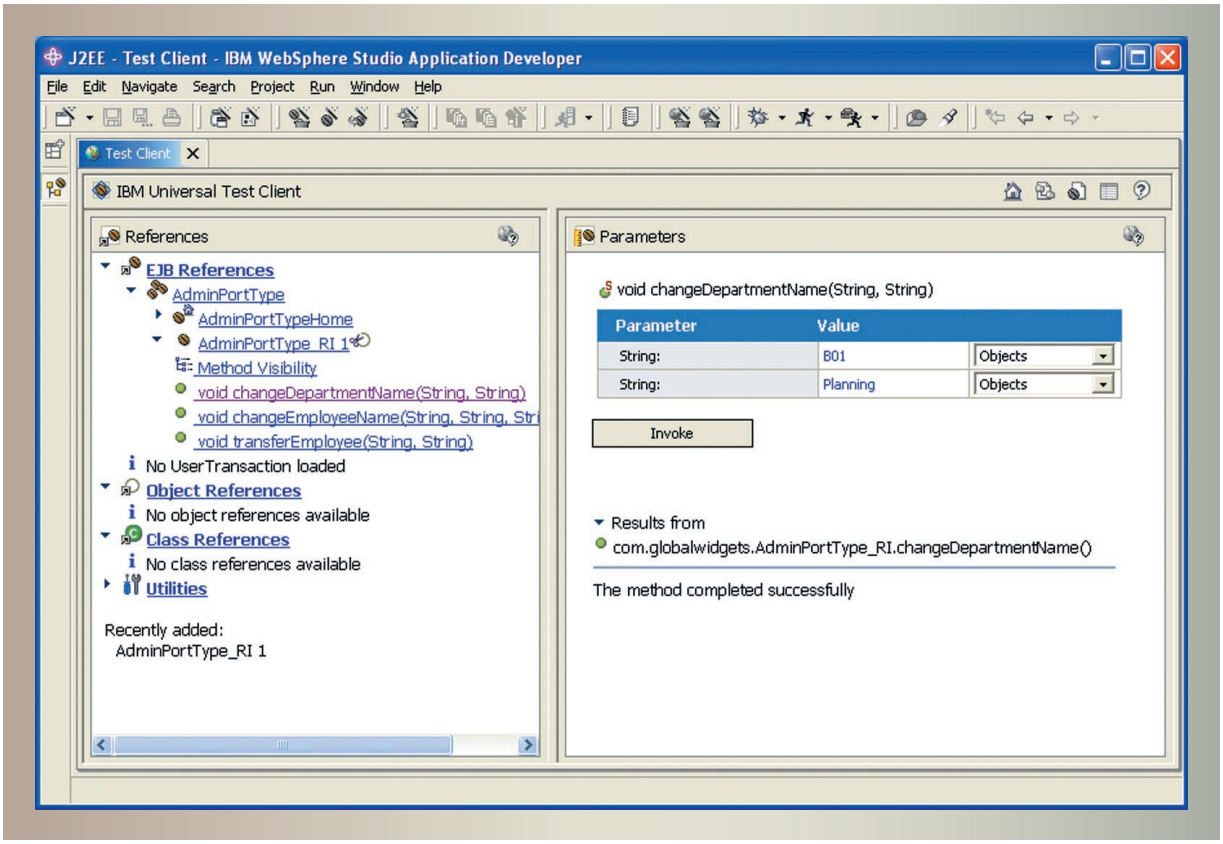
The critical distinguishing feature of the WebSphere Unit Test Environment is its locality; because it is installed as part of the development environment, it is possible to bypass the file distribution and publishing step otherwise required to install an application. Because the WebSphere Studio organizes files according to the J2EE specification, the Unit Test Environment can generate a server configuration that points to the application as it exists in the workspace. This means that even after the application has started running, the developer can make changes to the files. The server will detect that the files have been updated and immediately reflect the changes. This tight coupling provides an extremely productive edit/compile/debug experience, in which the overhead of republishing and redeploying changes made while debugging is almost zero. For further information on configuring the WebSphere Unit Test Environment, refer to Reference 23.

WebSphere Studio provides a related feature called *hot method replacement*, which has the capability to replace, in memory, the byte-codes for a previously loaded class. This means that the developer can make code changes to an application that is running on the WebSphere Application Server and has stopped at a breakpoint. When the source file is saved, the incremental Java compiler updates the JVM** with the new code. The current execution point resets to the start of the current method, but no other application state is lost.

WebSphere Remote Test Environment. The Remote Test Environment refers to the capability of Studio to publish to, configure, start, and interact with a server defined on a remote system. This capability should not be confused with the runtime system management function. Rather, it is intended to allow testing on a server other than that included as part of WebSphere Studio.

As with the Unit Test Environment, the Remote Test Environment can use different versions of the WebSphere Application Server, but each server instance must be installed separately from WebSphere Studio. The installation can be on the same machine on which WebSphere Studio is running, on another machine running on the same operating system, or even on a different operating system. This model can also be used to mitigate the performance impact of run-

Figure 12 Bean page in Universal Test Client



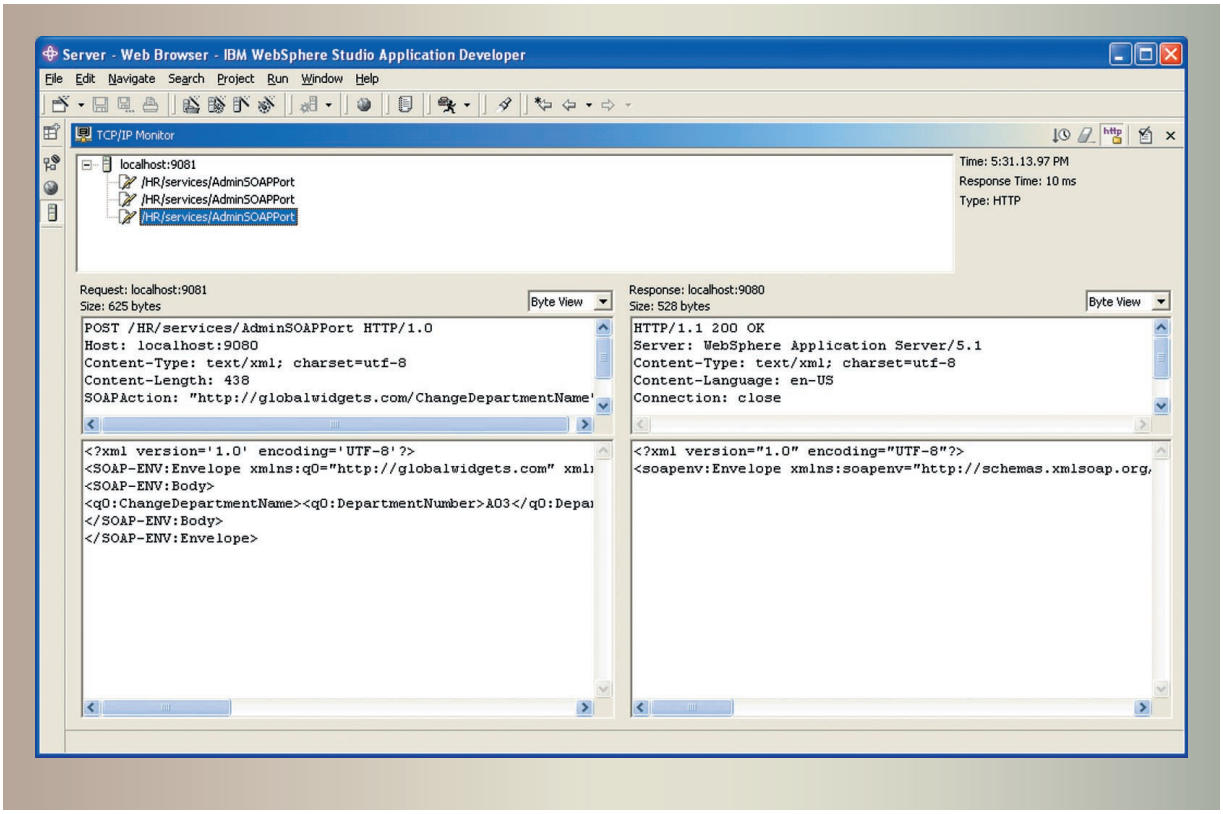
ning the application server on the development environment machine—a single dedicated test machine can provide the execution capabilities for a complete department. After a remote server has been completely defined, it is used in exactly the same way as the Unit Test Environment.

Testing an EJB in the Universal Test Client. When creating a Web application, the unit test mechanism is relatively straightforward—we display the application pages in a browser. However, when creating an EJB or Web-service-based application, it is much more difficult to unit-test the constituent parts of the application. The UTC is a Web application included with WebSphere Studio that provides an easy mechanism for testing a server-side component such as an EJB without the need to develop a custom application to drive the EJB. The UTC is enabled in the server configuration, which results in the additional application being included in the server when the configuration is published.

Interaction with the UTC occurs via a browser—either the one integrated into WebSphere Studio or an external browser. The UTC includes a number of pages, all accessible from the application home page: the JNDI explorer and properties pages allow us to view and modify the JNDI namespace, and the bean page allows us to explore active Java beans and EJBs. The typical test experience occurs in the bean page, but the first step is to load the EJB. The JNDI name of the EJB in question can be located via the JNDI explorer, and the EJB home can be added to the bean page. Alternatively, the bean page can load a Java class and then create a new instance of that type, as in Figure 12.

The interaction within the bean page mirrors the J2EE programming model. In the case of our admin bean, the first step is to select the home bean. The various methods are dynamically determined and displayed in the browser: we can then select a method to create or find the bean, and that method returns an in-

Figure 13 TCP/IP Monitor view with Web service operation invocations



stance of the admin bean (which is displayed in the browser). The next step is to click the Work with Object button and invoke methods on the bean: select the method, define the parameters to pass, and invoke it. This provides an indication of whether the admin bean is working as designed.

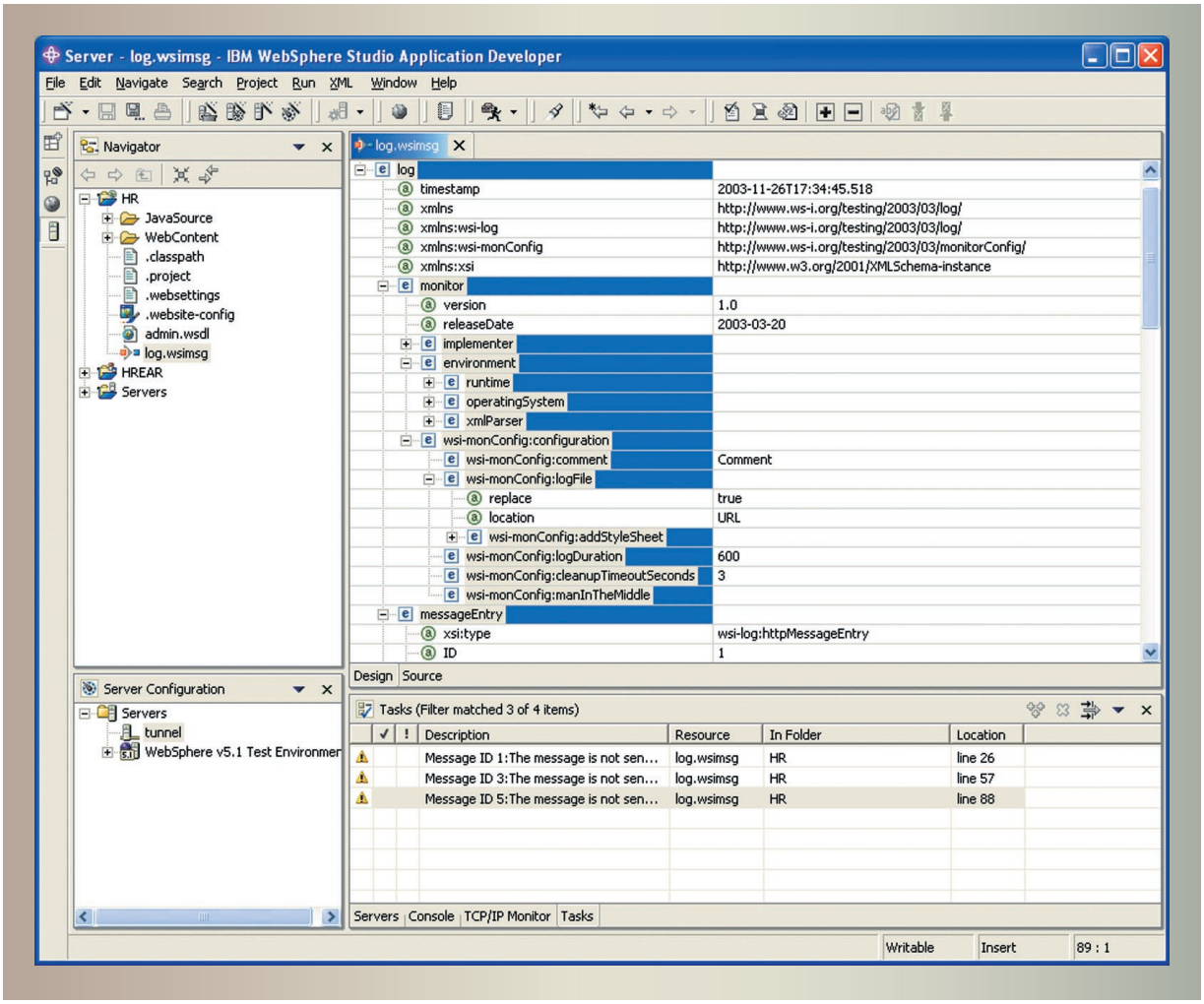
The result of the method invocation can be further inspected and invoked, or just displayed as necessary. Equally importantly, we can set breakpoints at any point in the application and use the UTC as the trigger to run to that point. The UTC usage model is very consistent with the best practice usage of EJBs—the UTC is a Web application that invokes the EJB.

It is important to note that EJBs are not the only artifact type that can be displayed and invoked. In fact, any Java class can be loaded via the UTC, making it an ideal mechanism to invoke multiple application paths. This includes the Java proxy that is generated for Web services or any user-provided class file.

Testing for WS-I conformance and publishing to a UDDI registry. After the business logic of the admin session bean has been unit-tested in the UTC, the scenario shifts to a focus on the Web service aspect of the implementation. The implementation is validated for conformance with the WS-I Basic Profile and then published to the Global Widget's internal UDDI Web service registry.

The TCP/IP Monitoring Server is the WebSphere Studio tool used to test for WS-I conformance. This server acts like a tunnel between the client and the Web service. It captures, displays, and records all message traffic. In addition to being a valuable debugging tool, it can save the recorded messages to a log file, which can be validated against the Basic Profile. Figure 13 shows the TCP/IP Monitor after it

Figure 14 Log file and task view

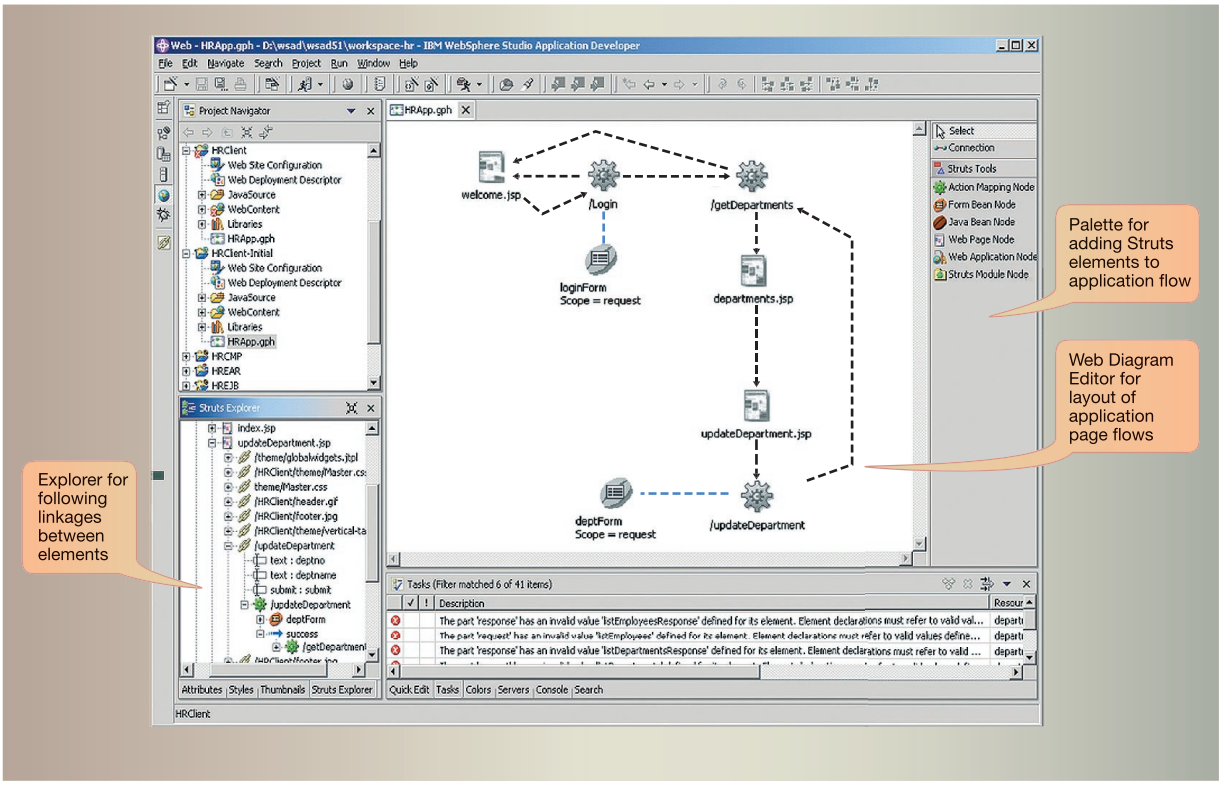


has captured three Web service operation invocations. The third invocation is selected, and its input and output messages are shown in the bottom left and right panes.

To validate the captured SOAP messages, simply click the checklist icon at the top of the TCP/IP Monitor view. This action saves the messages to a log file in the standard XML format specified by the WS-I Test Tools and runs the validator on the log file. Any errors or warnings detected by the validator are associated with the log file and appear in the Tasks view. In our example, the service is valid, but the validator detects a warning in the client: the Basic Profile specifies that messages should be sent using HTTP 1.1, but here the client is using HTTP 1.0. The log file and Task view are shown in Figure 14.

The final task in developing the admin Web service is to publish the service description to a UDDI registry. WebSphere Studio's Web Services Explorer acts as a universal client to any compliant UDDI registry. The Web Services Explorer lets us easily export a WSDL service description from a development project and publish it in a UDDI registry. The first step is to export the service description into WebSphere Studio's private UDDI test registry, which can be installed into the WebSphere test environment.

Figure 15 Tools for Struts



After the service description has been tested and validated, it can be published to the private UDDI registry used by Global Widgets by using either the registry's standard programmatic SOAP interface or its Web interface.

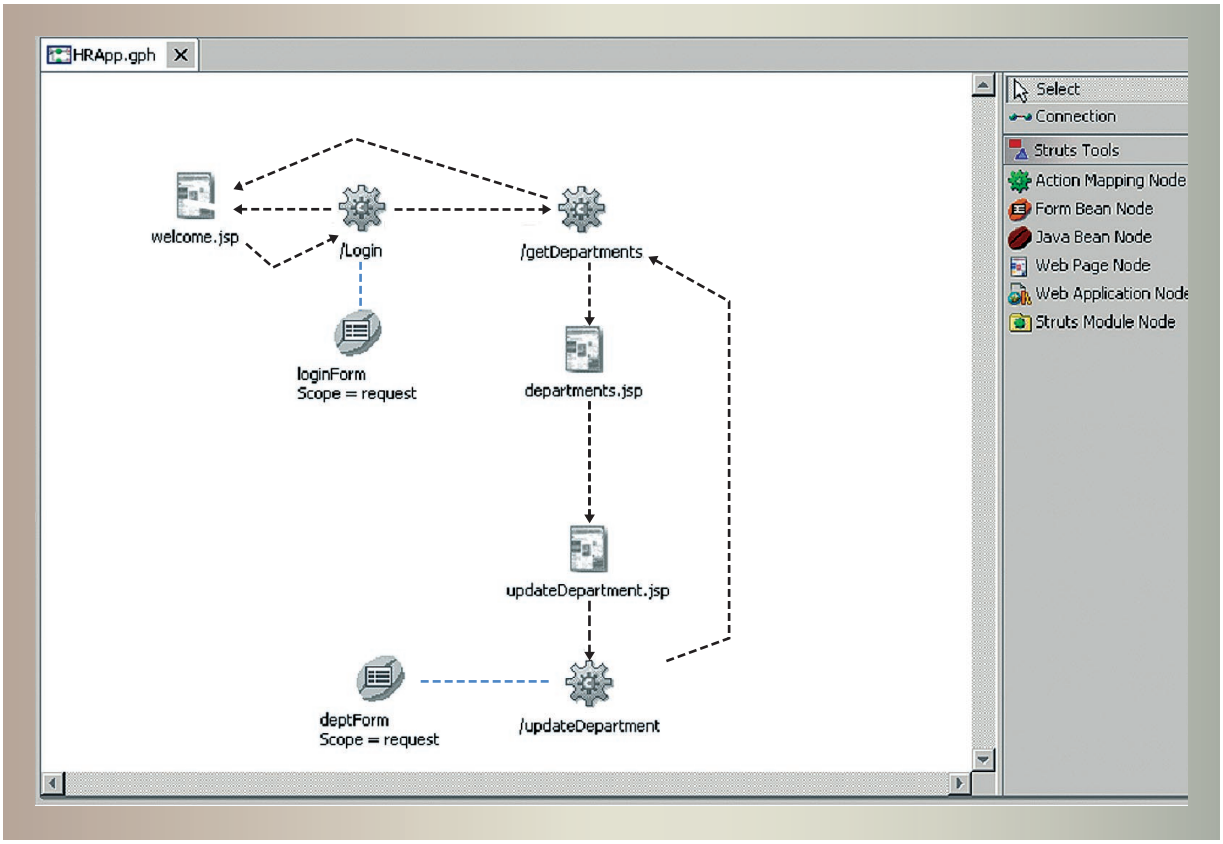
At this point, the admin Web service has been designed, implemented as a session bean, tested, validated, and published. It is now ready for use in client applications.

Creating the Global Widgets Web application. So far, the Global Widgets scenario has used a variety of WebSphere Studio tools to implement the logic and data access components for the Global Widgets Web application. The next phase focuses on the set of tools used for developing a Web user interface. The tight integration between the Java tools and Web development tools supports a task flow that had previously been difficult because the tools may have been based on different models and may have not worked well together.

The user interface for the Global Widgets application will support login, reporting, and updates. The reporting will include a master-detail data grid showing the list of departments and allowing users to drill down into the employees in a department. The client will be developed as a Struts application. (Struts is a popular open-source framework that supports development of Web user interfaces using the widely accepted model-view-controller architecture. For a detailed discussion of all the elements in the Struts programming model see References 15, 24, and 25.) WebSphere Studio supports the building of Struts-based applications through a set of tools illustrated in Figure 15.

The Web Diagram Editor is used to lay out the flow of a Web application graphically. The other important tool here is the Struts Explorer. It shows how all the elements of the application relate to each other and shows these relationships from different perspectives in the form of a hierarchy. From any

Figure 16 Web Diagram Editor



given node in the hierarchy, one can traverse links between the elements. The underlying infrastructure understands how these indirect references would resolve at runtime and shows this relationship directly in the hierarchy at development time. In this way the tool can tell the user at development time whether all the linkages are correct in much the same way that compilers show how references to named elements resolve.

Application developers often start designing a Web application by drawing diagrams of how the application pages relate to each other. WebSphere Studio provides this capability. Using the Web Diagram Editor (Figure 16), users can draw diagrams in which the nodes represent pages and actions and the arrows represent transitions to other pages or actions.

Figure 16 shows a diagram of the Global Widgets application at some point in the development. The rectangular icons represent JSP pages, and the "gears" represent code (actions) that will be invoked when each page is submitted. Struts concepts, such as form beans, Java beans, modules, and Web applications, can also be represented in the Web Diagram Editor.

At this point, all the nodes are gray, and the arrows are dotted. Nodes and arrows in this state are said to be *unrealized*. This is because there are no runtime artifacts that implement any of the diagram elements. We now go through a process of *realization* to create runtime artifacts. When artifacts become realized, the nodes representing them on the diagram will become colored. Arrows emanating from realized nodes become solid if the reference represented by the arrow is valid. When realized, the diagram represents the actual deployment artifacts: the tool does not maintain a separate abstract model from which the artifacts have been generated. Be-

cause of this, even Struts applications that were not created with this tool can be rendered and modified. This also means that the diagram will stay completely in synchronization with the deployed code if further modifications are made outside the tool.

A developer can realize any node by simply double-clicking on it and completing the resulting wizard. The wizard uses relevant information from the diagram to provide default values. In this way the developer can work on successive nodes and realize the entire diagram very quickly. After each wizard is completed, an appropriate editor for the given artifact opens. In the case of JSP pages, the Page Designer tool opens. (Details of this tool will be covered when we realize the Global Widgets JSPs.)

The logic elements created earlier are integrated into the Web application by implementing a Struts action (gear icon). This is done by double-clicking on the Struts action to invoke the New Struts Action wizard, completing the wizard, and inserting a few lines of Java code in the Java editor. For instance, for the GetDepartments action, this means invoking the Web service that returns a list of departments and then forwarding the list to Departments.jsp.

Designing the Web pages. The Web development tools include a full-function Web page editor that makes creating, editing, and maintaining Web pages simple and easy. This editor, named Page Designer, uses a drag-and-drop metaphor for adding elements to a page and includes sophisticated page layout functions to create pages with a professional look and function. The Page Designer specializes in handling typically hard-to-edit dynamic JSPs, as well as static HTML pages (for more information on JSPs, see Reference 26). The editor contains both a WYSIWYG (what you see is what you get) design page and a source page. The two pages are completely synchronized, so that developers can switch between them depending on their task. An important feature of Page Designer is that it preserves the exact format of surrounding HTML and JSP source while the design page is being used to modify an element.

The Page Designer Editor (Figure 17) opens whenever a JSP or HTML page is realized in the Struts editor. The wizard creates default page content to help jump-start the page design. If the page is connected to a form bean in the Web Diagram Editor, the wizard automatically creates page content that contains a Struts form connected to the form bean.

Page templates. Page templates are used to create a common look and feel for an application. A page template can factor out common visual elements of all pages in a site into a single file. This permits changing the look of the entire site quickly and easily by changing only one or more page templates. WebSphere Studio automatically propagates changes made to a page template to all pages based on the template. One way to think about a page template is as a Web page with "holes" in it. These holes, or *content areas*, are the only areas that Web developers can modify in a page instantiated from a page template. All other areas are read-only.

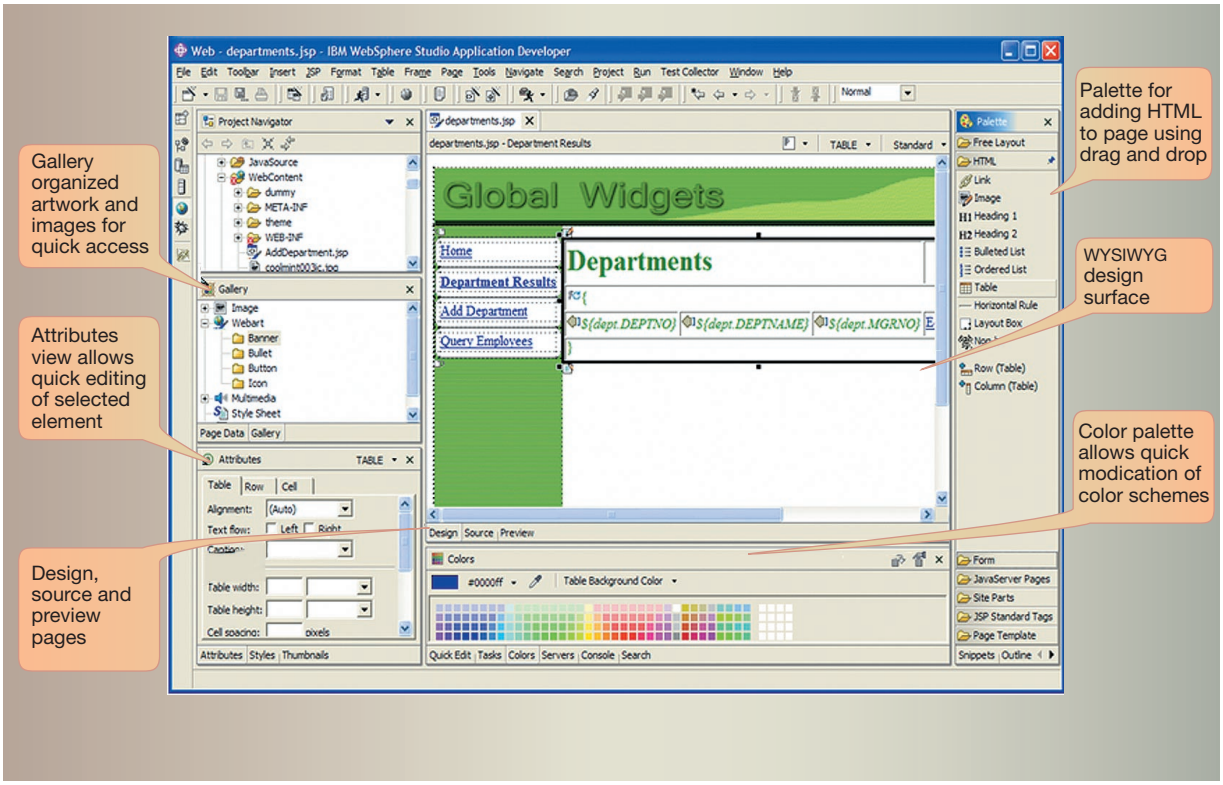
Page templates can be created quickly with the same process used to create regular Web pages. For our scenario, we can easily create a page template by using drag-and-drop techniques in the Page Designer Editor. First, we add a common header and a left-side navigation area. Then we drag and drop a Content Area from the Page Template category of the palette. When pages are created using this page template, page editors will only be able to modify content within the BODY content area. Our completed page template is displayed in Figure 18.

We can apply page templates to Web pages when they are created using the JSP wizard, or apply them to existing pages while editing them in Page Designer. In the scenario, we apply the template to the existing pages in the Global Widgets Web site.

The Global Widgets page template uses Cascading Style Sheets (CSS) to specify default text style and background colors. Style sheets are useful in abstracting fonts, color, and other design elements of a Web site and can be referenced within HTML/JSP pages. You can change the look and feel of the page template by using the WebSphere Studio CSS editor, which is called CSS Designer (Figure 19). CSS Designer is integrated with Page Designer so that you can change styles for a page and see the results immediately. By linking to the CSS style sheet from within the page template, the style sheet is applied to all pages instantiated from the template.

Adding navigation bars with Web Site Designer. The next step is to add navigation bars to the Global Widgets Web pages using WebSphere Studio's Web Site Designer tool. Managing navigation structure can be difficult and time-consuming because sites often require different navigation bars on each page, for example, if each page's navigation bar only showed its parent page and its direct child pages. Web Site De-

Figure 17 Page Designer Editor



signer automates the process of creating, organizing, and maintaining the navigation structure.

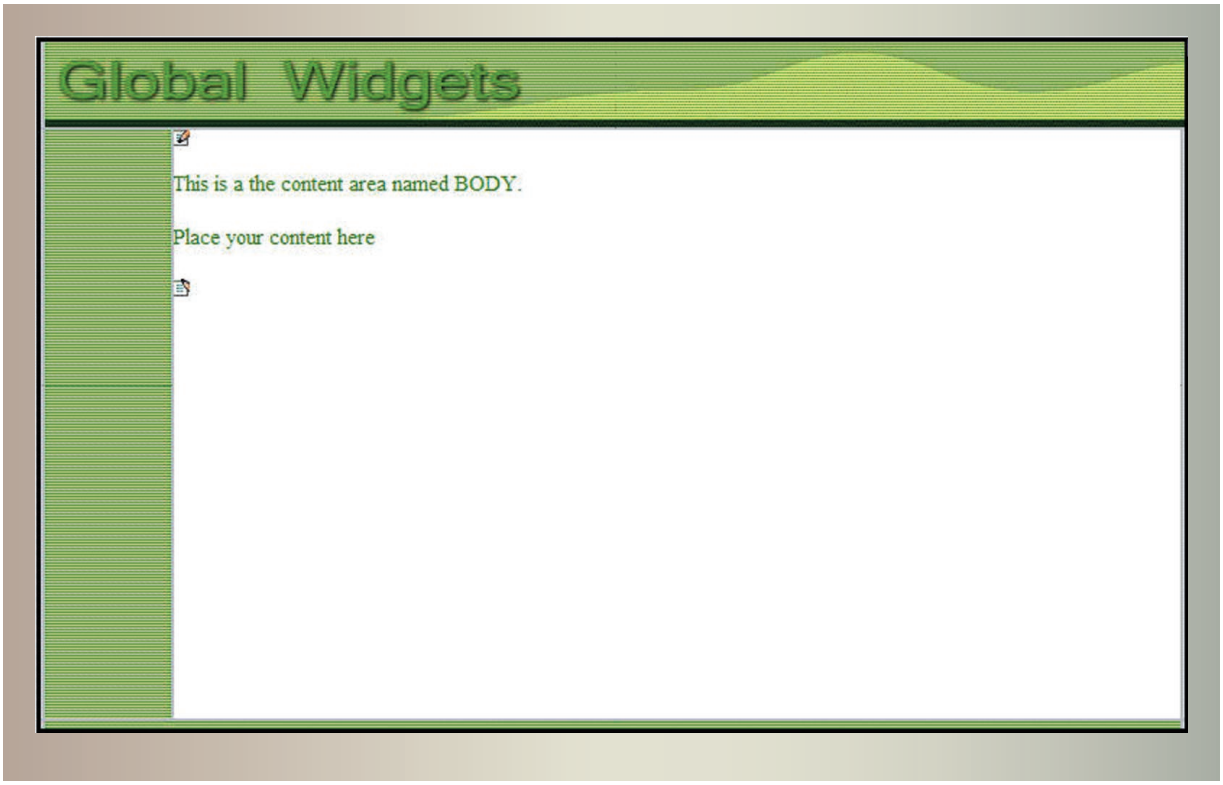
To add a navigation bar to the Global Widgets pages, we use Web Site Designer to build up a "map" of the site pages that we want to include in the navigation. This "map" can include existing pages as well as new pages that will be realized later, in the manner of the Struts editor. Figure 20 shows an example of a hierarchy that could be built for the Global Widgets site.

After we have created the hierarchy, we can add the navigation bars described by this hierarchy by using the navigation tags in the site parts drawer of the palette, which is displayed when a JSP or HTML file is open in Page Designer. For the Global Widgets pages, all that needs to be done is to open the page template in Page Designer and drag a vertical navigation bar into the left area of the page. When the template is saved, all the files that use the template are automatically updated with the appropriate navigation links. Furthermore, if Web Site Designer is used to reorganize the site, every navigation bar in the site will be updated accordingly. Figure 21 shows a sample page from the Global Widgets application with navigation bars added.

Performance analysis and optimization. Now that the Global Widgets Web application is more or less complete, the next step in the scenario is to tune the application performance. For this step, we use the WebSphere Studio Performance Profiling tool to identify and isolate performance bottlenecks, object leaks, and system resource limits.

The Performance Profiling tool. The Performance Profiling tool is capable of reconstructing the application execution path and displaying the interaction of business components such as servlets, EJBs, and JSPs. The tool can also collect execution information at the class instance and method invocation level.

Figure 18 Global Widgets page template



The tool targets applications at all levels of complexity, from simple, standalone Java applications to complex, multitiered enterprise applications running on multiple machines, and even on different platforms.

Two different types of data collectors, also known as profiling agents, are provided with the tool: the J2EE Request Profiler and the Java Profiling Agent. They allow us to gather profiling information used to analyze application execution from different angles.

The J2EE Request Profiler agent is used to visualize the application execution logic, without going into method invocation details, as it collects data from requests arriving on EJB and Web containers. This data collection mechanism enables the creation of sequence diagrams, which represent interactions among servlets, JSPs, and EJBs, while ignoring other artifacts of the application infrastructure that do not represent the business logic. As the application execution crosses the boundaries of a host, the remote discovery mechanism causes an attachment to other instances of the J2EE Request Profiler.

The Java Profiling Agent runs in the JVM process and receives notifications of JVM events, based on the JVMPI (Java Virtual Machine Profiler Interface). This agent is best used to identify performance details such as the classes or methods responsible for poor execution performance.

The WebSphere Studio Performance Profiling tool provides a powerful user interface for profiling an application and for analyzing the profile data. A set of statistical views helps us identify performance hot spots at the package, class, or method-invocation level. The tool also provides a set of graphical views that help us better understand the application execution at the node, process, thread, or method-invocation level.

Figure 19 CSS Designer

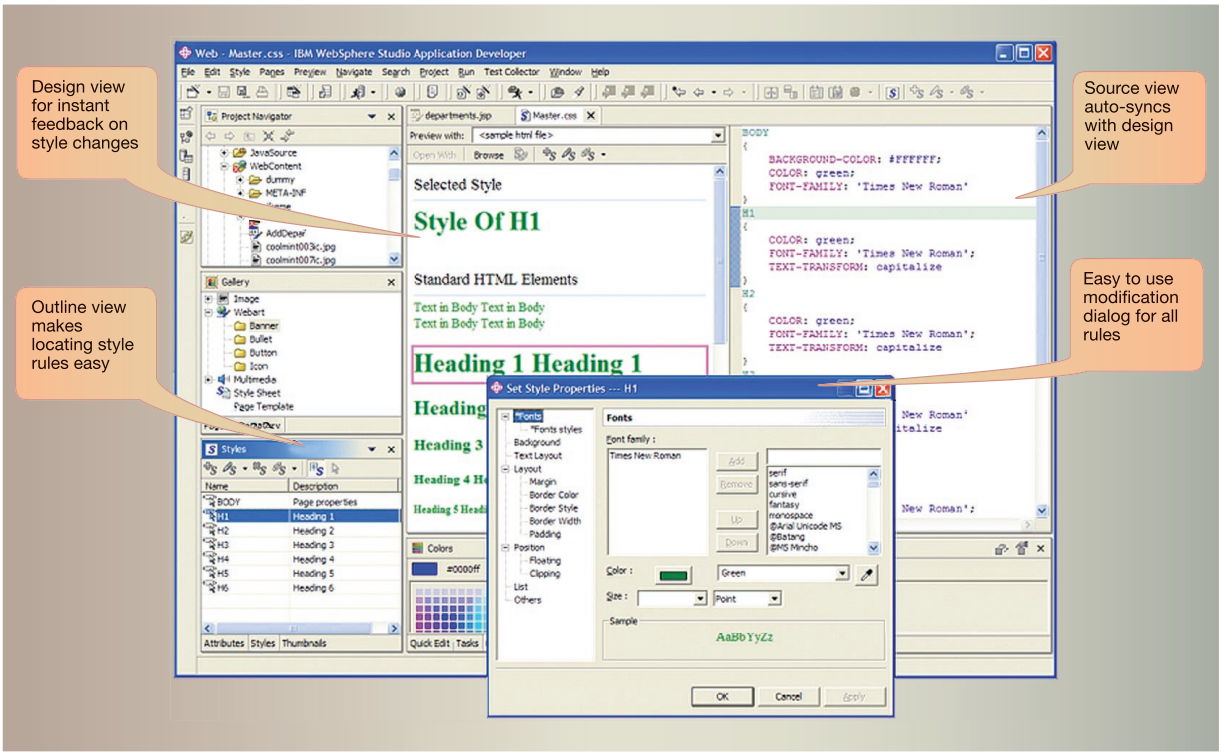
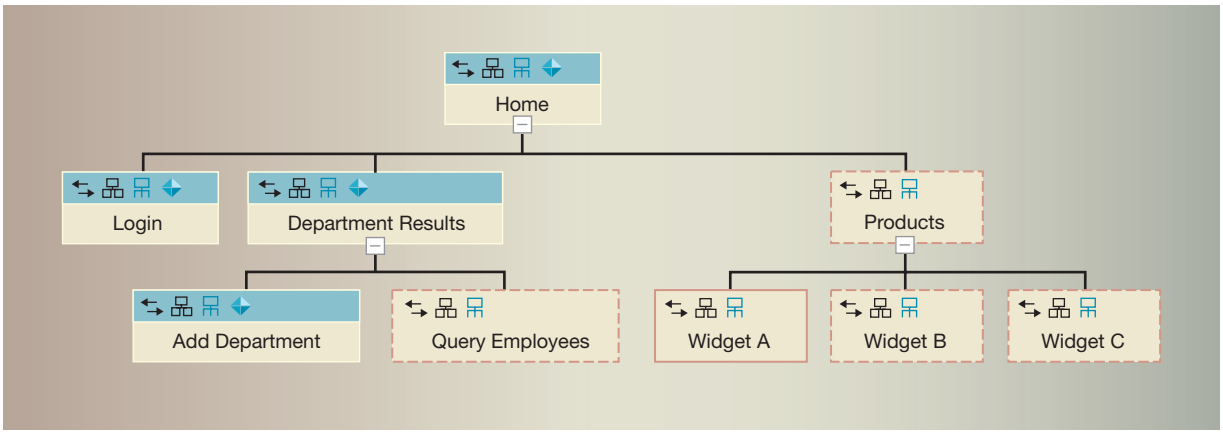
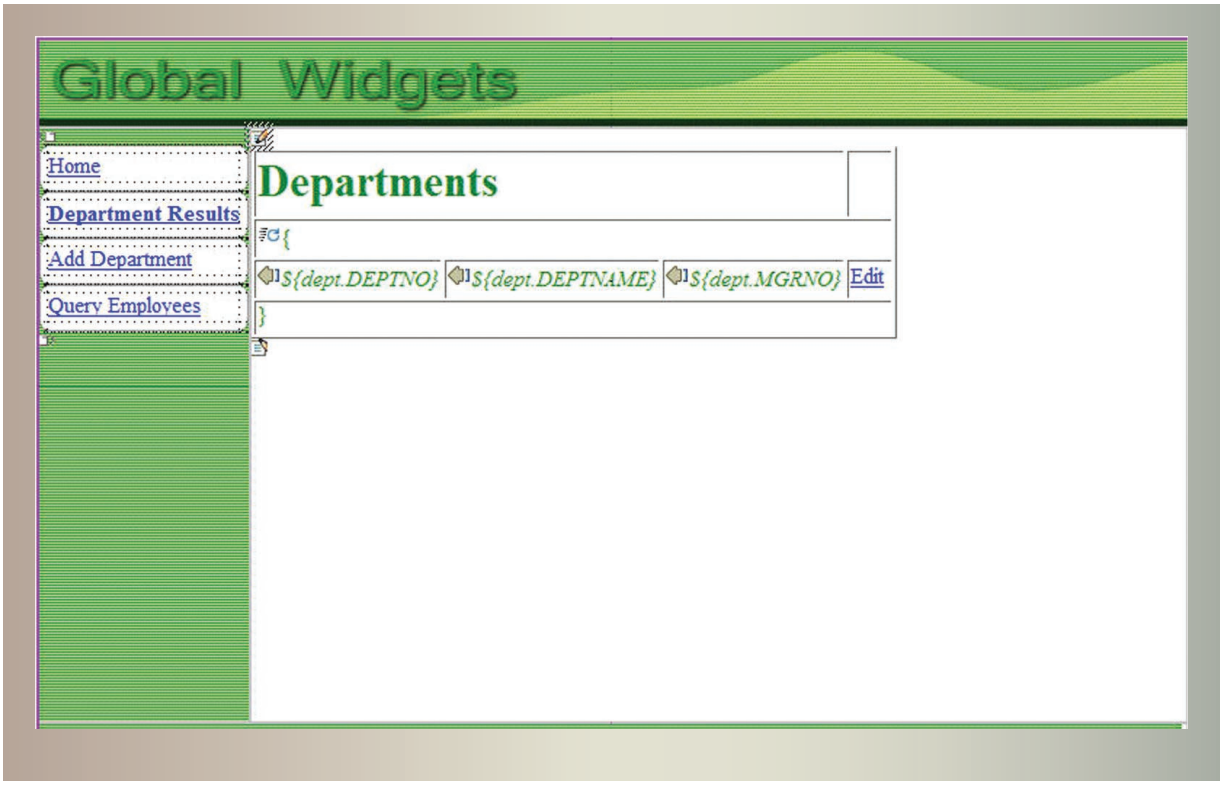


Figure 20 Global Widgets site hierarchy



Analyzing performance. This phase of the scenario focuses on solving a specific performance problem: the time taken to change the department name is very high the first time the operation is executed, while the next call to the same operation is visibly faster. The best way to start analyzing enterprise applications is to have an overall view of the application execution flow. The J2EE Request Profiler agent

Figure 21 Global Widgets page with navigation bar



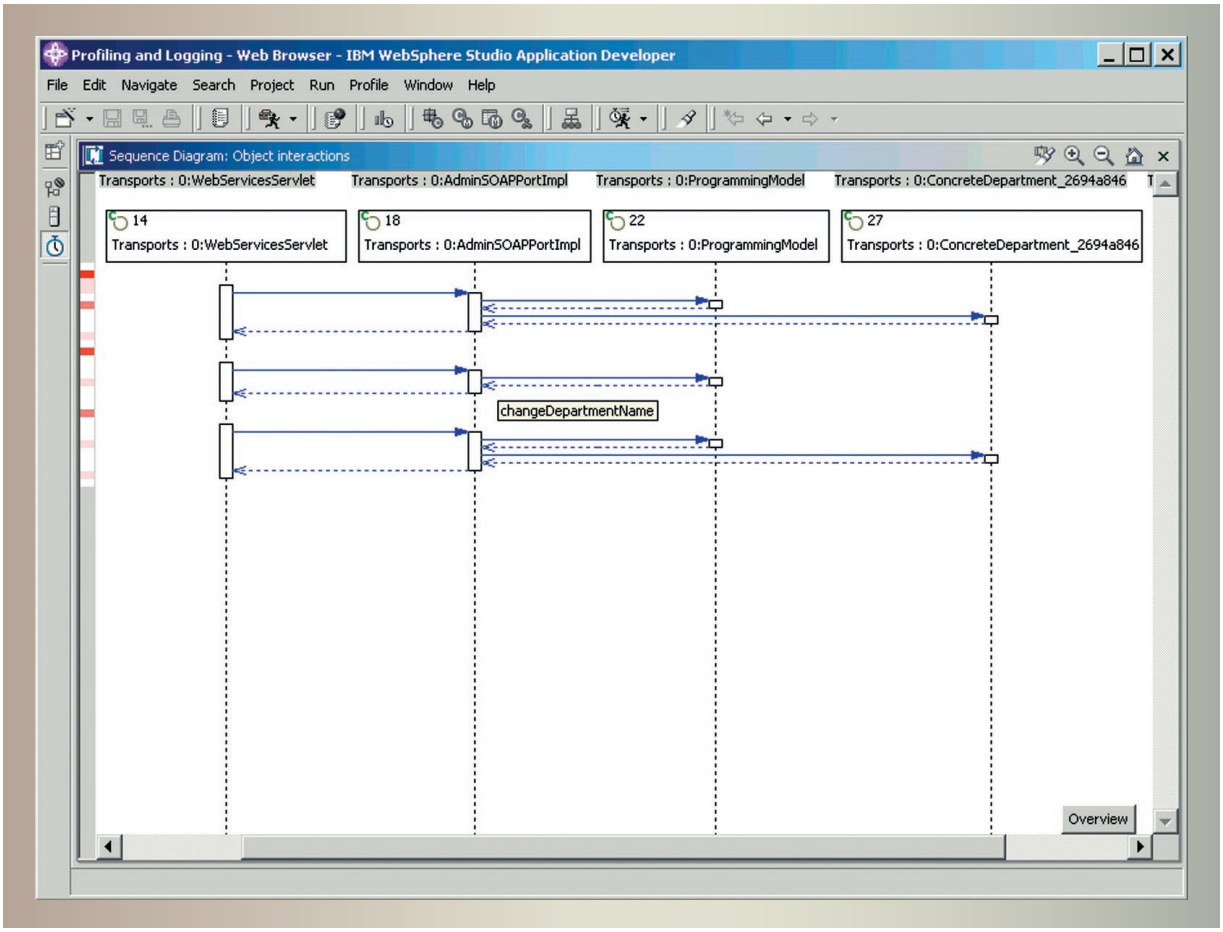
is used to collect this type of information. While the agent is monitoring the Global Widgets application, we will change a few department names and then return to the WebSphere Studio Performance Profiling tool and analyze the execution results.

In Figure 22, the Object Interactions view displays the application execution in a UML format. Notice the interaction among servlets and JSPs. In the figure, we can see that three department change operations were executed. The method AdminSOAPPortImpl.changeDepartmentName call results in a call to the ProgrammingModel.findByPrimaryKey method. For the first and last execution, the department was successfully found, and, as a result, there followed a call to the ConcreteDepartment_.setDepartment bean method. The red indicators on the left side of the diagram represent the elapsed time between consecutive events. Double-clicking on any of the indicators locates the sections with high execution time.

Next, the Java Profiling Agent can be used to get more details on the application execution, based on method invocation details. The Performance Profiling tool provides a set of statistical views to help analyze the execution performance at the package, class, or method level. The Package Statistics view (Figure 23) is a good starting point because it identifies the packages responsible for slow execution time. From here, we can drill down and look for the class or method responsible for poor execution performance.

In Figure 23, the Package Statistics view shows statistics of the different classes, grouped by their containing packages. The view can be customized to display any type of statistical data, such as memory allocation or time-based statistics. In this particular example, because the interesting information is the execution time, the packages are sorted using the execution Cumulative Time column. The figure identifies the com.globalwidgets package as the execution hot spot, and within this particular package, AdminSOAPPortStub is the class responsible for the slow execution time.

Figure 22 Object Interactions view



In a like manner, we can use the Class Statistics view to identify the method responsible for the AdminSOAPPortStub class's overall poor performance, which turns out to be the changeDepartmentName method. Next, we use the Method Invocation view to analyze the execution pattern for AdminSOAPPortStub.changeDepartmentName. The Method Invocation Table shown in Figure 24 displays the method invocations in a tabular format.

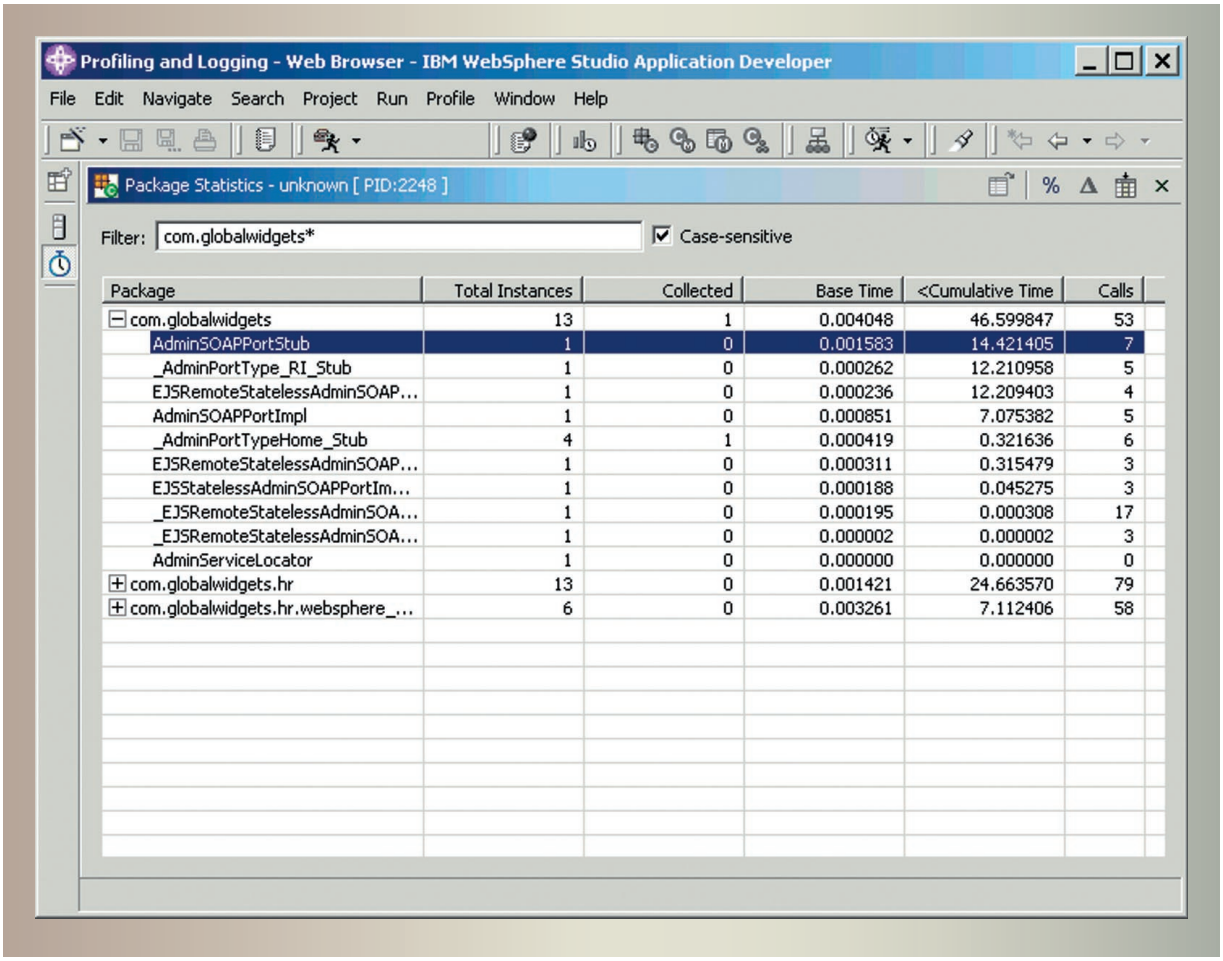
It is interesting to note that for the first method invocation the execution time (see the Cumulative Time column) is high compared with the second getAdminSOAPPort invocation. Analyzing the method invocations stack, we can finally determine that the invocation of the Stub$Invoke.invoke method on the first execution of AdminSOAPPortStub.changeDepartmentName is responsible for the poor performance.

This example introduces only one of the capabilities of the Performance Profiling tool. Additional capabilities include finding memory leaks, performing heap analysis, and tracing application execution (log analysis and correlation).

The Global Widgets scenario and developer productivity. Now that the Global Widgets application development scenario is complete, we can summarize its general tasks and the WebSphere Studio tools that were used.

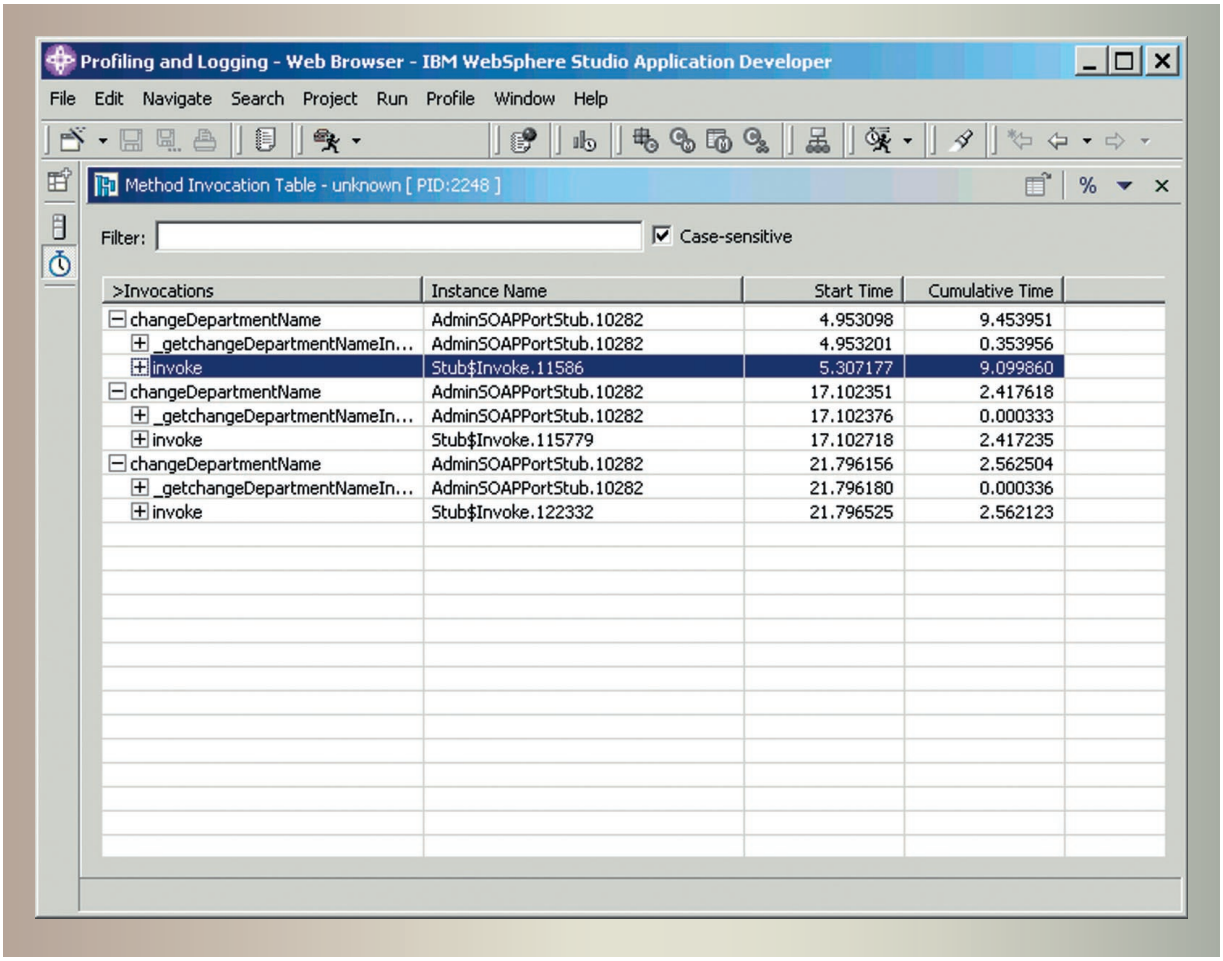
1. Develop database reporting functions.
   a. Create the reporting functions, using SQL Query Builder and Source Code editor and debugger for SQL stored procedures.
   b. Publish the database reporting functions as a Web service, using WebSphere Studio wizards.

Figure 23 Package Statistics view



2. Develop an EJB implementation for data update.
   a. Define the data update (admin) Web service using the WSDL editor.
   b. Generate a skeleton EJB Web service from the WSDL definition by using a Web service wizard.
   c. Create CMPs to represent the data to the EJB implementation, again by using a wizard.
   d. Define references to the CMPs by using the UML Visualizer.
   e. Create the business logic for the Web service session bean by using the Java editor with EJB extensions.
3. Test and debug the implementation using the Server Test Facility's UTC.
4. Publish the Web service.
   a. Test the Web service for conformance to the Web Services Interoperability Basic Profile (WS-I BP).
   b. Publish to the Global Widgets Web services UDDI by using the Web Services Explorer.
5. Develop the Web application.
   a. Create the Global Widgets Web UI as a Struts application by using the Web Diagram Editor.
   b. Design the JSP Web pages by using the Page Designer editor, page templates, and CSS Designer.
   c. Create a Web navigation hierarchy by using Web Site Designer.
6. Analyze and optimize performance by using the Performance Profiling tool.

Figure 24 Method Invocation Table



Rather than attempting a comprehensive overview, this scenario has focused on a single representative workflow. Other scenarios just as easily could have been chosen that would have illustrated the same degree of integration and productivity while exercising different functionality. It is worthwhile to note the wide variety of tools utilized in this single, tightly integrated workflow, and the ease with which application artifacts can move from one tool to another. For many tasks, wizards automate most or all of the programming steps. The high level of tools integration contributes greatly to developer productivity.

Eclipse Modeling Framework

The underlying framework that enables the fine-grained data integration between the WebSphere Studio tools is known as the Eclipse Modeling Framework (EMF).[27] This section provides a technical overview of the EMF. In addition to having general technical interest, this technology is notable for the manner in which it enables WebSphere Studio to be extended in an extremely tightly integrated fashion.

One future goal of WebSphere Studio is to provide an extensible tools platform that tools vendors and even customers can extend as a platform for creating J2EE and WebSphere applications. Such a platform would leverage the EMF technology used to implement the internal model APIs of WebSphere Studio. EMF is also a powerful technology for implementing object models outside of WebSphere

Studio. In the future, we expect to expose many of the internal WebSphere Studio object model APIs developed with EMF, along with ancillary APIs that facilitate manipulation of these models, to allow tool developers outside of IBM to reuse this function within other tools that can be integrated with WebSphere Studio.

EMF is an Eclipse subproject that provides a Java runtime framework and code generation facility for building tools and other applications based on structured models. Unlike many tools of this type, EMF models are surprisingly simple—essentially the class diagram subset of UML—providing a large percentage of the benefits of modeling with a very low cost of entry. Most importantly, EMF modeling enables and supports data sharing between applications, a critical requirement for an open, extensible tool environment like WebSphere Studio.

The EMF framework provides a metamodel, called Ecore, for describing EMF models. Ecore is based on the Object Management Group's MOF** (Meta Object Facility) specification. Ecore started out as an implementation of MOF but evolved based on the experience gained while using it to implement the WebSphere Studio tools. The resulting EMF framework is a highly efficient Java implementation of a core subset of the MOF API. Ecore corresponds to the EMOF (Essential MOF) portion of the recently accepted MOF 2 specification.

The canonical form of an EMF model is an XMI (XML Metadata Interchange) serialization of an Ecore model. One of the main strengths of EMF is its flexibility with respect to the means of defining an Ecore model:

- *XMI*—we can create an Ecore XMI document directly by using an XML or text editor or by using EMF's simple tree-based sample Ecore editor.
- *UML*—We can define the model by using a commercial UML modeling tool such as Rational Rose*, or by using a free Eclipse plug-in such as Omondo's EclipseUML graphical editor.[28]
- *Java*—We can use basic Java interfaces with a few simple annotations to describe an Ecore model.
- *XML Schema*—We can convert an XML Schema defining the data structures for the model directly into an Ecore model.

The XMI document approach is the most direct but generally only appeals to XMI experts. The UML choice is desirable if we are already using modeling tools, while the Java approach provides the benefits of modeling in a pure Java development environment (for example Eclipse's JDT [Java Development Tools]). The XML Schema approach is most desirable when the tool or application is intended to manipulate XML data that is already defined using an XML Schema, such as with Web services. Regardless of which input form is used to define an EMF model, the benefits are the same.

From an Ecore model, EMF's generator can create a corresponding set of Java implementation classes. Every generated EMF class implements an interface, EObject, that provides an efficient reflective API for accessing the object's properties generically. In addition, change notification is an intrinsic property of every EObject, and an adapter framework can be used to support open-ended extension to the objects. The runtime framework also manages bidirectional reference handshaking, cross-document referencing including demand-load, and arbitrary persistent forms with a default generic XMI serialization that can be used for any EMF model. EMF even provides support for dynamic models; that is, Ecore models that are created in memory and then instantiated without generating code. All of the benefits of the EMF runtime framework apply equally to them.

Most of the tools in WebSphere Studio model their data using EMF. These include models for XSD (XML Schema), WSDL (Web services), RDB and SQL (relational databases and queries), Java and EJB (J2EE tooling), and many more. A generic EMF mapping model and framework is also used to implement WebSphere Studio's mapping tools, such as EJB-to-RDB mapping and XML-to-XML document transformation. This kind of reusable mapping framework is possible because of the consistent use of EMF to model the data being mapped.

Some models in WebSphere Studio simply use the default XMI serializer provided by EMF to persist their models. Others implement customized serialization, enabling the model to be persisted in its natural (native) format. For example, XML Schema models are persisted as .xsd files, Java models as .java files, and so on. Other models, for example, mappings, are typically persisted using XMI, although specific types of mappings may be serialized differently. For example, XML Schema mappings can also be serialized in XSLT (Extensible Stylesheet Language Transformations) format.

There are two fundamental benefits from WebSphere Studio's use of EMF. First, it results in a productivity gain in implementing the tools by providing a high-level method (UML) to represent the design for communication among teams and by generating part of the implementation code. Second, while Eclipse itself provides an effective platform for integration at the UI and file level, EMF allows the WebSphere Studio tools to integrate at a much finer granularity than would have been possible otherwise. EMF modeling provides the foundation for fine-grained data integration in WebSphere Studio.

A framework called EMF.Edit extends and builds on the core EMF framework, adding support for generating adapter classes that enable viewing and command-based (undoable) editing of a model, and even a basic working model editor. EMF.Edit is also used by several of the tools in WebSphere Studio. More information on EMF (and EMF.Edit) is available in Reference 27.

Future directions and challenges

Experience with WebSphere Studio has shown that as the functionality provided to the user grows, especially in higher-end versions of the product, it becomes a challenge to keep the user interface uncluttered and easy to understand. This is true even with skillful use of Eclipse UI structuring mechanisms such as perspectives and views. In response to this challenge, the Eclipse community is introducing new UI organizing concepts, namely activities and contexts. Future versions of WebSphere Studio will exploit these concepts to provide a simpler, more task-focused user experience in products that continue to have very rich function.

We will also explore techniques for implementing tools that can be made available either on an individual computer or over the Web. Classic development tasks, such as compiling source code, are almost always performed by individual developers working on dedicated computers. However, for some tools it would be desirable to use them not only in the context of an IDE running on a personal computer but also on a server with access through a Web browser UI. Tools for configuring and administrating systems are typical examples.

Finally, an important goal for the WebSphere middleware platform is simplifying the development process. We consider an approach based on further separation between tasks performed by application programmers focused on the implementation of business logic and tasks performed by specialists dealing with the IT infrastructure. This direction is a joint effort between IBM tool and runtime teams; an important example can be seen in the recently published SDO architecture,[29] which is supported in the latest release of the WebSphere Studio Web tools and WebSphere runtimes. We expect to see more of this kind of runtime and tool simplification to ease the development of applications that run on the WebSphere platform.

Cited References and notes

1. *Java 2 Platform, Enterprise Edition (J2EE)*, Sun Microsystems, Inc., http://java.sun.com/j2ee/.
2. *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium (W3C) (2000), http://www.w3.org/TR/REC-xml.
3. World Wide Web Consortium (W3C), http://www.w3.org/.
4. *IBM WebSphere Studio Family*, IBM Corporation, http://www.ibm.com/software/info1/websphere/index.jsp?tab=products/studio.
5. *WebSphere Application Server Version 5.1 Information Center*, IBM Corporation, http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp.
6. *Eclipse.org*, Eclipse Foundation, http://www.eclipse.org/.
7. *Product Overview for Visual Studio .NET 2003*, Microsoft Corporation, http://msdn.microsoft.com/vstudio/productinfo/overview/default.aspx.
8. *BEA WebLogic Workshop*, BEA Systems, Inc., http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/workshop.
9. *Borland JBuilder X*, Borland Software Corporation, http://borland.com/jbuilder/index.html.
10. *Sun Java Studio Standard*, Sun Microsystems, Inc., http://developers.sun.com/prodtech/javatools/index.html.
11. *NetBeans' Products*, NetBeans.org, http://www.netbeans.org/products/.
12. *Oracle JDeveloper 10g*, Oracle Corporation, http://otn.oracle.com/products/jdev/index.html.
13. D. A. Norman and S. W. Draper, Editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1986).
14. *Apache Ant*, Apache Software Foundation, http://ant.apache.org/index.html.
15. *The Apache Struts Web Application Framework*, Apache Software Foundation, http://jakarta.apache.org/struts/.
16. J. Adams, S. Koushik, G. Vasudeva, and G. Galambos, *Patterns for e-Business: a Strategy for Reuse*, IBM Press, Double Oak, TX (2001).
17. A. Ryman, "Tools for Building Web Services," in *Java Web Services Unleashed*, R. Brunner et al., Sams Publishing, Indianapolis, IN (2002), pp 641–674.
18. A. Ryman, "Understanding Web Services," developerWorks, IBM Corporation (July 2003), http://www.ibm.com/

developerworks/websphere/library/techarticles/0307_ryman/ryman.html.
19. *Basic Profile Version 1.0a, Final Specification*, Web Services Interoperability Organization (2003), http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.htm.
20. C. Lau and A. Ryman, "Developing XML Web Services with WebSphere Studio Application Developer," *IBM Systems Journal* **41,** No. 2, 178–197 (2002).
21. See G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language (UML): User Guide*, Addison-Wesley, Upper Saddle River, NJ (2001); and *OMG Unified Modeling Language Specification, v. 1.5*, UML Revision Task Force, Object Management Group (March 2003).
22. T. Francis, E. Herness, R. High Jr, J. Knutson, K. Rochat, and C. Vignola, *IBM WebSphere 5.0 Application Server*, Wrox Press, Birmingham, UK (2003).
23. H. Kushner, *Developing J2EE Applications with IBM WebSphere Studio*, IBM Press, Double Oak, TX (2003).
24. C. Cavaness, *Programming Jakarta Struts*, O'Reilly, Cambridge, MA (2003).
25. T. Husted, C. Dumoulin, G. Franciscus, D. Winterfeldt, and C. R. McClanahan, "*Struts in Action: Building Web Applications with the Leading Java Framework*," Manning, Greenwich, CT (2003).
26. M. Hall, *More Servlets and JavaServer Pages*, Prentice Hall, Upper Saddle River, NJ (2002).
27. *Eclipse Modeling Framework*, Eclipse Foundation, http://www.eclipse.org/emf/.
28. *Eclipse*, Omondo EclipseUML, http://www.omondo.com/index.jsp.
29. *IBM and BEA Joint Specifications Overview*, IBM Corporation and BEA Corporation (2003), http://www.ibm.com/developerworks/java/library/j-commonj-sdowmt/.

Accepted for publication January 30, 2004.

Frank Budinsky *IBM Software Group, IBM Toronto Lab, 8200 Warden Ave, Markham, ON L6G 1C7, Canada (frankb@ca.ibm.com).* Mr. Budinsky is leader of the Eclipse Modeling Framework (EMF) project at Eclipse.org, co-architect and implementor of the EMF framework, and EMF code generator. Frank has been involved in the design of several frameworks and generators, including the IBM/Taligent Compound Document Framework in VisualAge/C++, the Composed Business Object Builder in Component Broker, and a common framework for mapping tools in WebSphere Studio. He holds B.Sc. and M.Sc. degrees in electrical engineering from the University of Toronto and is lead author of *Eclipse Modeling Framework: A Developer's Guide* (Addison-Wesley, 2003).

George P. DeCandio *IBM Software Group, Research Triangle Park Lab, 3039 Cornwallis Road RTP, NC 27709 (decandio@us.ibm.com).* Mr. DeCandio is a Senior Technical Staff Member and senior manager of Web and portal tooling for WebSphere Studio. He is one of the original architects and developers of WebSphere Studio. Most recently he and his teams have been involved in defining the JavaServer Faces specification (JSR 127) and in architecting and developing the IBM tooling for this new technology. He has also worked as a programmer and team lead on the Visual Builder portion of IBM's VisualAge products. He received a B.S. degree at the Rochester Institute of Technology in 1989.

Ralph Earle *IBM Software Group, Research Triangle Park Lab, 3039 Cornwallis Road RTP, NC 27709 (ralphe@us.ibm.com).* Dr. Earle is a senior software engineer in the Rational Division, and manager of User Assistance Development for WebSphere Studio. From 1998–2003, he was the content architect for IBM's VisualAge and WebSphere Developer Domains. He is the co-author of *Enterprise Computing with Objects* (Addison-Wesley, 1998).

Tim Francis *IBM Software Group, IBM Toronto Lab, 8200 Warden Ave, Markham, ON L6G 1C7, Canada (francis@ca.ibm.com).* Mr. Francis, who joined IBM in 1990, is a Senior Technical Staff Member and development manager of the WebSphere Tools team in the IBM Toronto Lab. In 2001 he received an IBM Outstanding Technical Achievement Award for his work on WebSphere Studio. Tim is a senior member of the WebSphere Architecture Board, a core member of the Rational Desktop Tools Development Council, and is a co-author of *Professional IBM WebSphere 5.0 Application Server* (Wrox Press, 2003).

Julian Jones *IBM Software Group, IBM Toronto Lab, 8200 Warden Ave, Markham, ON L6G 1C7, Canada (julianj@ca.ibm.com).* Dr. Jones, a Senior Technical Staff Member at the IBM Toronto Lab, joined IBM in 1988 and worked on the usability of application development tools. He received a B.Sc. degree in occupational psychology from University of Wales in 1984 and a Ph.D. degree in human-computer interaction from the University of York in 1991. In 2002 he received an Outstanding Technical Achievement Award for his work on the Eclipse project.

Jin Li *IBM Software Group, IBM Toronto Lab, 8200 Warden Ave, Markham, ON L6G 1C7, Canada (jinli@ca.ibm.com).* Mr. Li is a User Experience lead for the IBM Rational tools products, with a focus on usability. He participated in many customer engagements, helping IBM customers build their business applications. He has an M.Sc. degree in computer science from the University of Toronto, is a certified Sun Java developer, and is an IBM solution developer.

Martin Nally *IBM Software Group, Research Triangle Park Lab, 3039 Cornwallis Road RTP, NC 27709 (nally@us.ibm.com).* Mr. Nally, who joined IBM in 1990, was lead architect for VisualAge/Smalltalk, one of the architects of VisualAge/Java, and the lead architect and development manager for WebSphere Studio. His current title is Chief Architect for Rational Desktop Products.

Connie Nelin *IBM Software Group, IBM Austin, 11501 Burnet Road, Austin, Texas 78758 (nelin@us.ibm.com).* Dr. Nelin is a Distinguished Engineer in the Data Management Architecture and Technology department. Since joining IBM in 1987 she has worked on database application development support and tooling. She currently has overall responsibility for application development tooling strategy, architecture, and development for the DB2 family of products.

Valentina Popescu *IBM Software Group, IBM Toronto Lab, 8200 Warden Ave, Markham, ON L6G 1C7, Canada (popescu@ca.ibm.com).* Valentina is an advisory software developer, leading a team that develops user interfaces for profiling, tracing, and logging tools for the Eclipse platform. She is currently interested in user interface issues including analysis, correlation, and visualization of problem determination data (such as profiling, tracing, and logging data). In 2003 she received an IBM Outstanding Technical Achievement Award for her work on autonomic computing.

Scott Rich *IBM Software Group, Research Triangle Park Lab, 3039 Cornwallis Road RTP, NC 27709 (srich@us.ibm.com).* Mr. Rich is a Senior Technical Staff Member, the development manager of WebSphere Studio Application Developer, and leader of the Rational Desktop Tools Development Council. He has been with IBM for 15 years, holding a number of technical positions involving VisualAge/Smalltalk, VisualAge/Java, and WebSphere Studio.

Arthur G. Ryman *IBM Software Group, IBM Toronto Lab, 8200 Warden Ave, Markham, ON L6G 1C7, Canada (ryman@ca.ibm.com).* Dr. Ryman is a Senior Technical Staff Member and development manager in the Rational Desktop Products group. In 2001 he received an IBM Outstanding Technical Achievement Award for his work on Web services. Dr. Ryman is a member of the IBM Academy of Technology, an Adjunct Professor of Computer Science at York University, a senior member of the Institute of Electrical and Electronic Engineers, and a member of the Association for Computing Machinery.

Timothy W. Wilson *IBM Software Group, Research Triangle Park Lab, 3039 Cornwallis Road RTP, NC 27709 (tww@us.ibm.com).* Mr. Wilson is a Senior Technical Staff Member in the WebSphere Studio Tools group. He has been working with integrated development environments and programming languages since 1987. He is currently chief architect of Enterprise Generation Language, a business language tool set based on Eclipse. In 2003 he received an Outstanding Technical Achievement award for the creation of the WebSphere Studio tool set surrounding the Struts open-source framework.