A WebSphere Studio Enterprise Developer solution
May 2002

WebSphere® software

IBM

Simplifying the development of enterprise-scale e-business applications

By Joseph Pesot, Stephen Hancock and Clifford Meyers
IBM Software Group

Contents

Introduction

To stay competitive in today's marketplace, your enterprise needs to develop an e-business solution that focuses on providing up-to-the-minute information that is accessible and accurate. This means your business applications and data need to be available across every level of the enterprise and on the Internet at any time. Your enterprise Web applications must also effectively leverage heterogeneous development teams, hardware and existing applications to optimize fundamental business processes that can differentiate your business from the competition. The right e-business solution can allow an enterprise to more effectively manage relationships with customers and suppliers, streamline business processes, speed decision making, reduce project cycle times and increase control over inventory.

However, creating an e-business solution and enterprise Web applications for your company can be challenging. Web applications must be flexible and able to grow with your business. They must be able to incorporate a variety of disparate core business applications and maximize the significant information technology (IT) investment of legacy core business applications. These factors create a significant IT challenge, which can slow Web application development, modernization and time to market.

This white paper discusses the critical Web application development needs of an enterprise and explains how an integrated development environment can simplify the development of e-business solutions and associated Web applications. It also describes key aspects of IBM WebSphere® Studio Enterprise Developer and how it can help your enterprise development team modernize existing applications for use with Web applications and e-business solutions. The intended audience includes software developers, system architects, IT professionals, technical managers and anyone interested in Web application development.

Enterprise Web applications

An enterprise using traditional applications faces several challenges as it strives to create Web applications that take advantage of an existing IT infrastructure. When new and diverse technologies are linked with existing enterprise applications, the integrity of business operations may be compromised if the new architecture is not extensible and compatible with the current one. Timing is also critical when an enterprise is developing an e-business solution. If a company moves too slowly in this area, it can mean losing customers to the competition.

The first step in creating an e-business solution is to develop appropriate enterprise Web applications. Consider the topology of a typical Web application, shown in Figure 1. Often called N-tiered applications, a Web application consists of a client tier (usually browsers and thin clients), a middle tier (Web and application servers) and an enterprise information system (EIS) tier of legacy applications and databases (like IBM System/390® and other host systems).

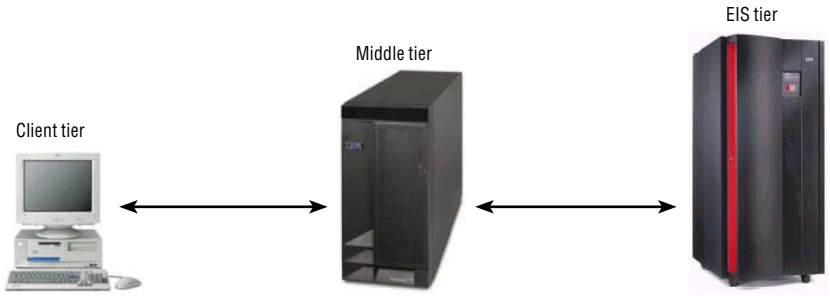

Figure 1. N-tier application topology

In the client tier, browsers generally remove the requirement to distribute code to each user of an application. This greatly simplifies the task of managing deployment of applications because enterprises have anywhere from hundreds to thousands of internal users and thousands to millions of external users. As new client hardware platforms emerge, such as handheld devices or data-enabled phones, efficient application designs avoid sending large amounts of code to the client.

The middle tier, which can have many levels, plays a critical role in a Web application because it holds everything together. For instance, the middle tier is responsible for managing communication with the client tier and the EIS tier. The middle tier often leverages JavaServer Pages (JSP) to manage input from the client and the delivery of information back to the client. The middle tier also manages communication with the EIS through various connector technologies. In addition, the middle tier is where much of the new application code is written to exploit component technologies and Internet enabling. The exploitation of such component technologies can result in the creation of JSP pages, and JavaBeans and Enterprise JavaBeans (EJB) components and servlets. The middle tier is responsible for a significant portion of new business logic that is required to handle today's industry processes dictated by emerging online business models.

The EIS tier is where the bulk of an enterprise's information technology investment resides. It often includes transaction-processing applications and databases, usually developed over a long time, that are critical to the daily operation of a business. Enterprises can leverage this core functionality as they build Web applications to save significant time, energy and money.

The middle-tier problem

A critical issue for many enterprises is a mismatch between existing development skills and the role of the middle tier in Web application development. The skills necessary for effectively developing the middle-tier components of a Web application are often difficult to learn, expensive, in short supply or not available at all. Typically, programmers from multiple skill groups are required when developing e-business solutions. Each brings important capabilities to the Web application development team, yet no one group is capable of handling it all. For instance, traditional programmers with extensive experience on host systems are familiar with the high-performance, transactional approach of EIS applications. Often they are not generally experienced in the more object-oriented, class-based programming models necessary in the middle tier. Conversely, programmers with experience in Java™, Extensible Markup Language (XML) and other Internet programming technologies are usually not familiar with the business logic that created existing host systems. As a result, it is critical that teams made up of individuals from diverse skill groups are brought together to solve the middle-tier problem.

The Web application development environment

Insufficient Web application development environments have further complicated the problems facing enterprise development teams. The Web application development environment has been fractured and disjointed, and so have been the results. Many early Web applications consist of chaotic strings of HTML and scattered programs written in common gateway interface (CGI) or Practical Extraction and Report Language (PERL) scripts. This should not be the foundation of a robust, 24x7 enterprise application. Security, reliability and maintainability have often been below acceptable standards. In addition, developers have faced a lack of adequate development tooling for building complex Web applications. Web applications and their middle-tier components have often been created by what can be called point tools — development tools that were extremely narrow in terms of function and focus. No single tool has been created to provide a complete e-business solution. Early development teams entering the emerging Internet space have been forced to cobble together development environments as best they could.

To make matters worse, much of recent Web application development has been haphazard. Some developers in emerging Web spaces initially set aside — or at times abandoned — standard approaches and processes to build applications as fast as possible. This rushed development process is sometimes referred to as *Web time*. The reality is that standard development processes that have helped ensure quality code are just as important as ever. Today's point tools do not always incorporate the knowledge and best practices of the last thirty years of application development.

The Web application development team

The process for planning and building software applications has been evolving for many years. Historically, requirements were gathered, project leads were put in place and a team was enlisted. Next, the work was distributed, milestones were set and project-tracking mechanisms were put in place. An entire software project might be built in COBOL, PL/1, C, C++ or some other programming language. In the past, software development teams were fairly homogeneous. Generally, the entire application would run on a single platform. Within the project everyone basically spoke the same language, learned the same processes and set similar milestones. This is not the case for some teams currently creating e-business solutions.

Today's Web application development teams often include business analysts, managers, host programmers, application programmers, Web page designers, graphics designers, Java programmers and component developers. One person might fill each role, or any one person might be required to play multiple roles. Planning a Web application has become complex because of the varied skills and numerous roles required. For example, on the EIS tier, you may have COBOL or PL/1 programmers with experience building IBM CICS® transactions or other applications and databases associated with current business logic. Their approach to development is probably based on models of structural programming (and most likely do not separate the user interface from the business logic). The processes for building such systems are specific to the host environment.

Your development team needs professionals with experience in your existing business applications – perhaps host programmers – working on aspects of the middle tier. On the middle tier, you can have Java programmers building servlets, classes or JSP code. Their development and architectural model is likely to be more object-oriented. It is crucial to get the business knowledge that is embedded in your existing applications and leverage it as you develop in the middle tier. Graphic designers and HTML programmers develop for presentation on client systems (graphics, JSP pages, HTML). Today this means browsers, but other client platforms, such as hand-held devices and data-enabled phones, are becoming popular. Team members with these skills tend to have backgrounds in building user interfaces. Of course, managers of teams developing Web applications might come from any of these programming disciplines or perhaps a technical business role, or some other technical lead position. Teams need development tooling that not only allows diverse roles to interact, but also integrates the members as a team.

Elements of e-business development tools

For enterprises to fully engage and implement Web applications, development tools are needed to directly leverage existing assets, help solve the middle-tier skill and complexity problems, and support proven development practices. The need for an integrated development environment in which critical aspects of Web application development and the associated modernization of existing applications must be addressed. Such tools would also include capability that simplifies the combination of the complex technologies involved – within and between the various tiers – in Web application development.

Development tools for the enterprise need to take the best capability of the point tools and combine them into a single coherent environment where team members with various technology backgrounds and experience can bring their particular expertise to bear. These tools also need to support the enterprise's requirement to create reusable components that can be leveraged throughout the e-business environment.

Using component technologies

A key aspect of today's Web application development effort revolves around understanding and leveraging of components and component-based architectures. Component models allow teams to reuse proven and reliable runtimes and capabilities instead of duplicating development effort. A component is defined as any piece of code that provides a service to another aspect of an application. It can be built in any language and may ultimately run in any environment. Components may also be harvested from existing capability within the enterprise. The use of component technology provides significant benefits in areas, such as reuse, reliability, maintenance, scalability and ultimate time to market of business applications.

The modernization of enterprise applications relies heavily on harvesting candidates for becoming components. When a particular capability becomes a component, that capability can be used again and again by many different applications. Emerging component models also allow for distributed application topologies that can be deployed across multitiered environments, scaling as business needs grow. Finally, component models isolate aspects of an application so that, as technology changes, components can be updated independently, limiting the impact of changes and updates to a manageable scope. Managing change is an important benefit of leveraging component architectures and models. If not managed well, change can result in the degradation of a modern Web application into unwanted complexity. A comprehensive design pattern can bring much needed structure to Web application development.

Managing change with model view controller

Model view controller (MVC) is a design pattern for creating components that was originally developed to help manage change in software application development. MVC separates a user interface from business logic and data. The key aspects of MVC are:

Model. The model contains the core of the application function. The model captures the state of the application. It does not include knowledge of the view or controller.

View. The view is the look of the application. The view presents, gathers and submits information, but it does not include knowledge of the model or controller.

Controller. The controller manages the execution flow of the application, passing appropriate state information between the model and the view.

The MVC approach allows the key aspects of a Web application to be isolated and maintained independently. For Web applications, the classic form of MVC needs modification because the Web brings unique challenges to software developers. Most important is the stateless connection between the client and server. This stateless behavior makes it difficult for the model to notify the view of changes. On the Web, the browser needs to query the server again to discover modification to the state of the data within the application. Another change is that the view is implemented using different technologies, such as Java, PERL, C/C++, from the model or controller. This creates the need to separate key development roles. For example:

- *Business programmers should focus on developing services, not HTML.*
- *The Web page designer does not need to have direct involvement in—or awareness of—service development.*
- *Changes to Web page layout by a page designer should not require changes to code of a service developer.*
- *Customers of the service should be able to create views to meet their specific needs.*

MVC, modified for the Web, is called MVC2. Figure 2 presents the MVC2 approach as it might appear in an N-tier environment. Input from the user is taken at the client and passed to a controller (servlet) that examines the input and the current state of the model. The model (or business objects) may reside on application servers, host servers or both. The controller then populates the appropriate response view with data obtained from the model. The view (display page) is then presented to the user. Using a design pattern, such as MVC2, helps to separate code responsibility and associated roles within the development team. Such separation helps ensure that changes (regardless of where they occur) are isolated.

Visual assembly tools

The Web application builder would prefer to simply leverage components, taking advantage of these services as needed without having to understand exactly how they are actually implemented. Once candidates for componentization have been identified, the focus shifts to assembling components in the most efficient way to provide end-to-end capability that meets business needs quickly. How does the application developer assemble components based on disparate technologies? Clearly, to solve this problem visual assembly tools are required to help organize and arrange components and their operations in meaningful ways. They automatically define the order of events and how data is passed from tier to tier and from component to component. Visual assembly tools should function to allow the developer to focus on the business logic required, while complex details of integrating disparate technologies are handled automatically.

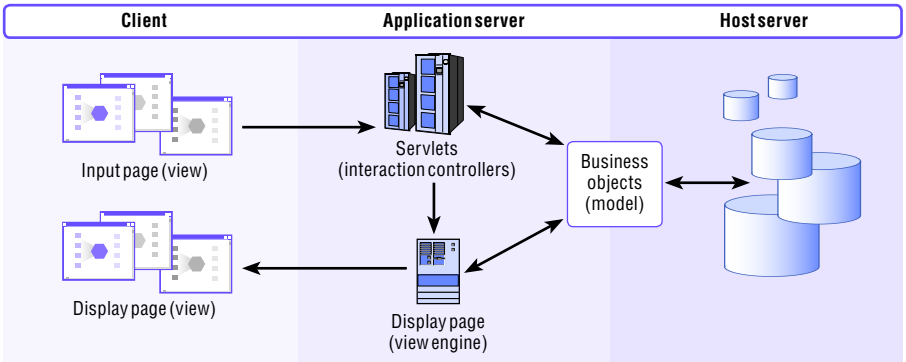

Figure 2. The MVC approach in an N-tiered environment

Visual assembly tools simplify the task of designing, coding, testing, debugging and maintaining the controller. The key role of visual assembly tools within enterprise Web application development is to build and manage the controller in an MVC2 implementation. The model and the view elements are the components driven by the controller and require business logic and Web page layout skills, respectively. A combination of visual assembly and componentization tools can help development teams maintain the MVC separation.

WebSphere Studio Enterprise Developer

An integral part of an e-business solution is leveraging multiple skill sets, modernizing existing applications, identifying components and assembling them into Web applications. Yet implementing an effective Web application requires a robust component architecture and development tooling. WebSphere Studio Enterprise Developer offers tools for building and managing complex, component-based, N-tier Web applications to development teams with heterogeneous skill sets. It also leverages component architecture models and development tooling that can enable the successful creation, deployment and maintenance of enterprise Web applications. With WebSphere Studio Enterprise Developer, an enterprise can create Web applications by integrating diverse employee skill sets and extending existing systems. This approach can allow enterprises to leverage proven runtime environments while helping to reduce deployment risks.

WebSphere Studio Enterprise Developer also supports accepted practices and emerging Web application technologies to help ensure that development teams build robust component-based applications. In particular, you can leverage an MVC architectural model and an open-source MVC2 implementation design. WebSphere Studio Enterprise Developer also helps extend emerging Web application component-oriented technologies and projects, such as XML, Struts, Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). (See the appendix for a brief description and resources for each.)

Designing a Web application

The focus on modernizing enterprise applications and creating e-business solutions dictates specific development processes. Existing enterprise applications must act as a resource pool for the Web applications under construction. You will need new application code and business logic. You need to create reusable components so that, as your enterprise builds its Web applications, its capability can be leveraged as components in future Web applications.

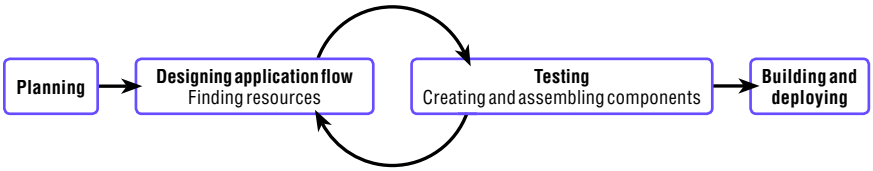

Figure 3. A Web development process

Figure 3 shows a development process where existing components are identified, new components are created and application flow is defined, and components are linked together in an efficient way. Once you complete initial planning, simultaneous development begins that defines application flow, finds existing components, creates new components and engages in various levels of testing. As this development concludes, your teams can engage in build and deploy activities. The following information highlights basic development activities and how WebSphere Studio Enterprise Developer can provide development teams with the support they need to be effective.

Designing the application flow

Connecting components that comprise disparate technologies is the biggest challenge facing Web application developers. WebSphere Studio Enterprise Developer visual assembly environment is used to define the basic flow of a Web application graphically connecting JSP pages with component services or actions. This approach simplifies the creation of an MVC controller by masking the complexity of the disparate technologies involved. The actions defined in the visual assembly environment can be implemented in whatever technology is most appropriate for your specific needs (COBOL, PL/1, Java or IBM Enterprise Generation Language).

WebSphere Studio Enterprise Developer visual assembly environment leverages Struts, an emerging open standard for constructing MVC-based Web applications. Struts provides an action servlet that manages the runtime relationship between JSP pages and actions. The use of Struts helps to ensure an effective separation of code responsibilities and developer roles. (See the appendix for a brief description and additional references for Struts.)

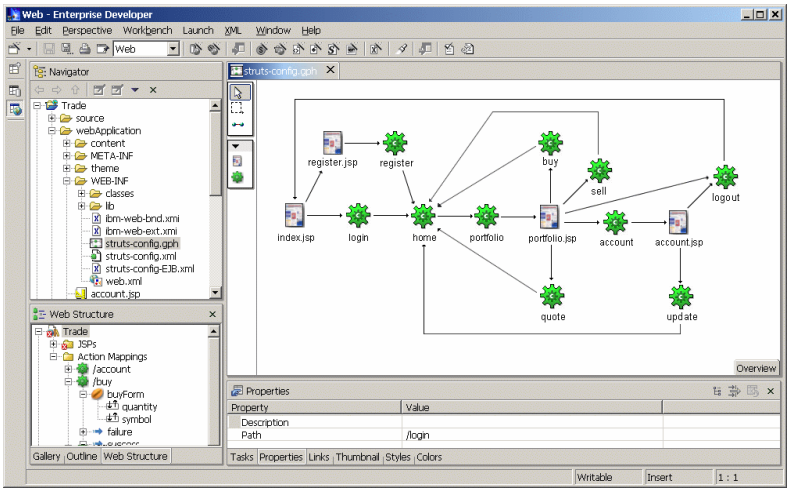

Figure 4. Struts lets you quickly see layout and view of Web applications.

The visual assembly environment is used initially as part of the design process, helping a development team quickly lay out view (JSP) and model (action) components (see Figure 4) without having to consider the technical issues of combining disparate technologies that have yet to be created or harvested from existing capability. The visual assembly environment is then used throughout the development process to add, extend and test a Web application's capability.

Creating components from existing capability

A key aspect of creating a Web application involves leveraging existing applications, and harvesting components from within the enterprise. However, developers may encounter significant difficulties when they try to create components based on traditional applications. WebSphere Studio Enterprise Developer simplifies the process by providing powerful componentization tooling that helps development teams turn existing applications into reusable components. WebSphere Studio Enterprise Developer adapter tooling, provides a wizard-based user interface, which helps a developer identify important aspects of an EIS component.

The tooling automatically creates a Java 2 Connectivity (J2C) interface to the host component. This component interface is a Java class that runs on the Web server but can communicate with transactions or other capability on the host. This component interface can then be incorporated into the visual design tool, making it available as an action within the Web application. These connectors are automatically designed to be complete Web services, allowing the enterprise to directly engage Web service business paradigms.

Developing new components

In addition to leveraging existing capability, your development team will often need to create new components to be part of a Web application. WebSphere Studio Enterprise Developer provides powerful source editors integrated as a single development tool for building many component types required, including HTML, JSP pages, EJB components, JavaBeans, COBOL, PL/1, Assembler and IBM Enterprise Generation Language (EGL). Many combinations of these technologies can be leveraged when implementing actions described in the visual assembly environment. This approach can allow WebSphere Studio Enterprise Developer to provide a fully integrated development environment in which the various Web application development roles can be performed effectively by a heterogeneous team.

Creating Web pages with HTML source editor

Once the general layout of Web pages has been defined in the visual assembly environment, the actual Web pages can be created. WebSphere Studio Enterprise Developer HTML source editor is an advanced-function HTML editor that can allow your Web page designer to quickly build complex dynamic Web pages, visually and textually. The dynamic element support enables you to include form elements, Java applets, embedded scripts, dynamic controls and JSP tags. You can toggle among three modes to visually design pages, work with HTML text and preview your pages (see Figure 5). Content help provides guided editing as you insert new tags and is available in the HTML source view. To help you create the visual impact you want on your Web sites, the editor includes its own library of reusable graphics and two graphic programs that create, edit and animate image files.

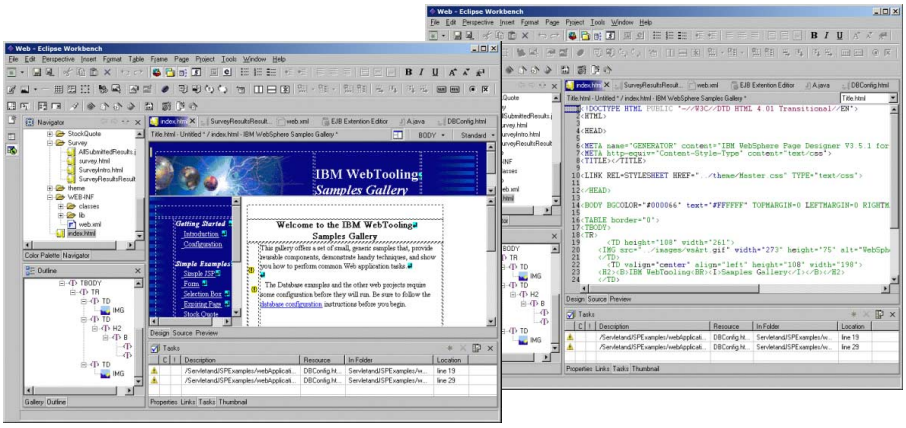

Figure 5. Web page designer preview screens allow you to see your work before it's published.

Implementing EGL applications

As part of WebSphere Studio Enterprise Developer, IBM also includes EGL, a high-level programming language (based on the IBM VisualAge® Generator product family). EGL is a fourth-generation programming language (4GL) that enables traditional developers to code aspects of Web applications at a high level and then generate the appropriate source code for targeted runtime environments. From the visual assembly environment, a traditional developer can choose to implement a particular action using EGL. Then, using the EGL editor, seen in Figure 6, and the associated scripting language (optimized for rapid application development), your developers can create programs, functions, records and other structures. And then generate COBOL or Java source code as needed. The EGL editor also includes task wizards to help your developers quickly create the parts they need.

The EGL language is similar to COBOL and PL/1, and other 4GLs, providing developers who have those skill sets with the ability to quickly generate applications that can run in either the middle tier (Java) or EIS tier (COBOL or CICS) of a Web application. This capability provides a developer with options to write code once and then have it potentially run in multiple environments (using COBOL or Java generation as needed).

Specifically, EGL allows traditional developers to create business logic and then generate Java programs for the middle tier. Further, EGL provides traditional developers with an easy way to take that same business logic and create new EIS COBOL applications and the associated Java wrappers and connectors needed in the middle tier. This capability can help your development teams overcome the middle-tier problem. Because the Java wrappers and associated connectors are created automatically, Java programmers are not required to develop wrappers for COBOL applications they didn't create. The wrappers are created automatically and can be directly leveraged in the visual assembly environment.

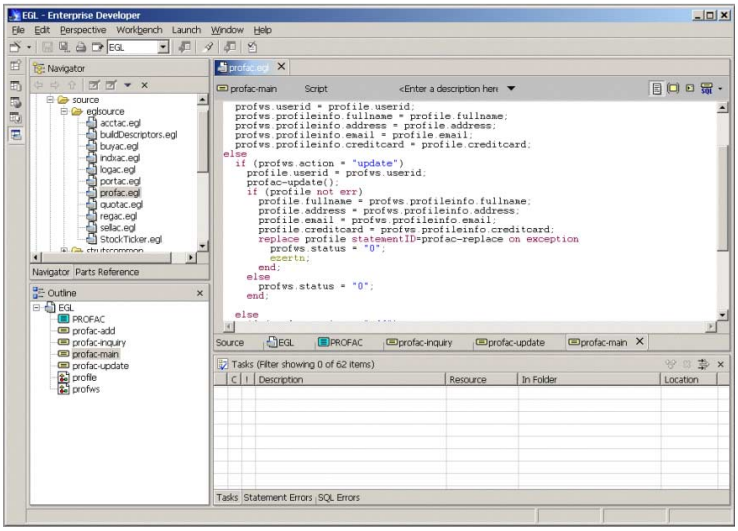

Figure 6. EGL editor

COBOL and PL/1 source editor

Actions defined in the visual assembly environment may also be implemented in COBOL or PL/1. Traditional developers can create and edit host-based resources using the WebSphere Studio Enterprise Developer COBOL, PL/1 source editor (see Figure 7). It includes the ability to connect to various host systems to locate your development resources, as well as syntax highlighting, a job monitor and the capability to enter TSO commands. Once a developer has prepared the capability, the developer can use the adapter tooling to connect this host logic to the appropriate part of the Web application through the visual assembly environment.

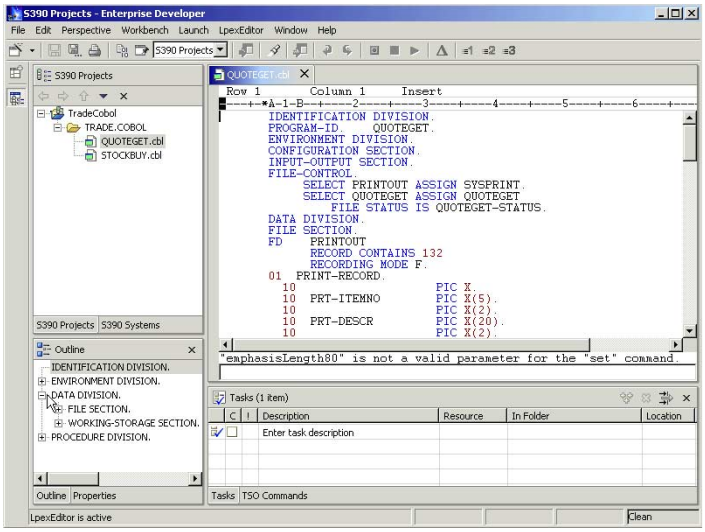

Figure 7. COBOL, PL/1 source editor

Java source editor

If your application includes middle-tier Java logic, your Java programmers can use the WebSphere Studio Enterprise Developer full-function Java editor to implement selected actions using Java technology-based software. The Java editor includes multiple views showing project structure, the class outlines and properties and their associated values. The Java editor, featured in Figure 8, also includes a class hierarchy browser. This editor can be used to create JavaBeans and EJB components for use in WebSphere Studio Enterprise Developer Web application development.

Testing and debugging

Testing Web applications can be just as complex as building them. As a team brings together the various technologies required when an enterprise modernizes its applications, it needs a development environment that provides end-to-end support. WebSphere Studio Enterprise Developer starts by providing breakpoint and monitor testing capability within the visual assembly environment. This capability can allow the team to debug the flow of the Web application, helping to ensure that each aspect of the flow and the associated connections perform as intended. WebSphere Studio Enterprise Developer includes a validation framework that identifies many coding errors while the code is being written. Programmers can select the error and immediately see the area of the code where the problem was identified.

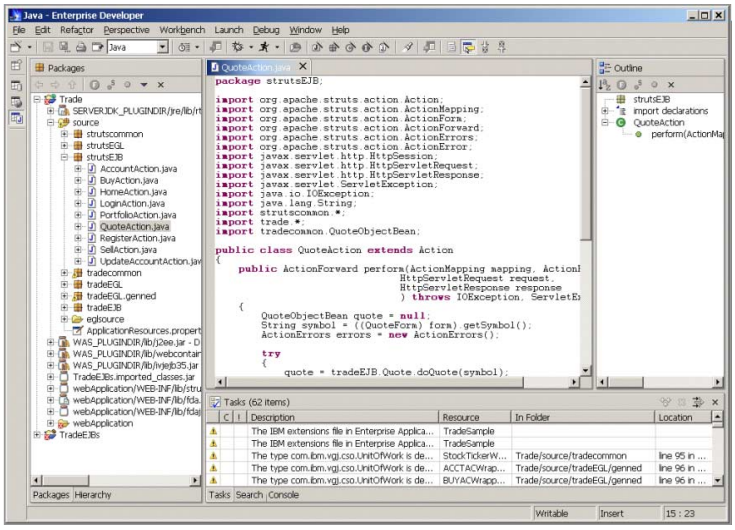

Figure 8. Java source editor

WebSphere Studio Enterprise Developer also includes problem determination capabilities such as traditional debugging, distributed code profiling and unit testing. WebSphere Studio Enterprise Developer problem-determination capabilities feature local debugging of Java source, including middle-tier and EIS debugging for Java, EGL, COBOL and PL/1 technology-based components. Traditional debugging support includes functions like setting code breakpoints and monitors and querying variable content. Distributed code profiling provides performance analysis tools that can locally or remotely monitor code as it is executing to better understand memory allocation and bottlenecks. The unit test environment can allow users to model their configurations for test or production. Users can update their environment version and release at any time.

Building and deploying

After a Web application is tested and approved, your developers can use WebSphere Studio Enterprise Developer to build and deploy the application to target environments on specific machines, as shown in Figure 9. WebSphere Studio Enterprise Developer also enables automated deployment of EGL applications based on instructions included in a build plan.

Build plans are created to specify such information as target machine, target directory, the list of files to move, procedures to invoke to build the outputs and so on. The build plan is an XML file, which is generated from a build descriptor specifying generation and output preparation options, as well as information the generators derive from program content. Build descriptors can often be shared by developers working on the same project. When the build process is initiated and outputs are to be built on a runtime platform, the WebSphere Studio Enterprise Developer build command processor transfers files to the appropriate machine and executes any necessary commands on that machine as specified in the build plan. WebSphere Studio Enterprise Developer build deployment uses TCP/IP sockets to transfer data between the client and target server machines.

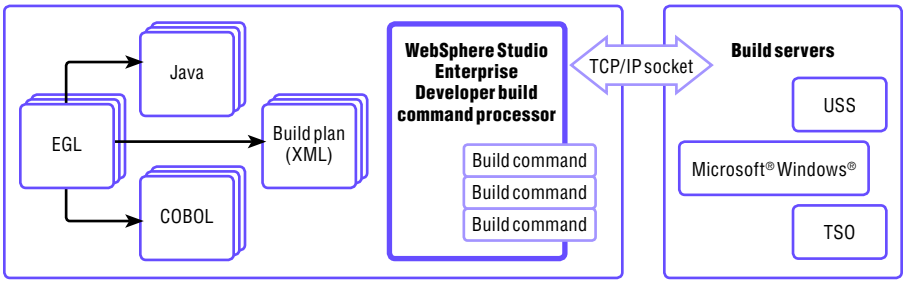

Figure 9. WebSphere Studio Enterprise Developer build command process

Summary

Competitive pressure to move all phases of business to emerging e-business models drives the need for enterprises to optimize business processes. An integral part of an e-business solution is the Web application. Implementing a Web application requires robust component architecture and development tools. WebSphere Studio Enterprise Developer provides the architectural models and development tools for successful creation, deployment and maintenance of enterprise Web applications.

e-business is a necessity in today's marketplace and your enterprise faces a competitive opportunity if you act fast or a competitive liability if you move too slowly. Development tools for Web applications must enable the enterprise to act quickly and effectively. WebSphere Studio Enterprise Developer can simplify the development of enterprise Web applications by helping teams reduce development cycle times and enable the transformation of existing systems.

For more information

For more information about IBM Application Framework for e-business, visit:
http://www.research.**ibm.com**/journal/sj/401/flurry.html

For more information about WebSphere Enterprise Developer, visit:
ibm.com/websphere/studio

For more information about MVC, visit:
http://ootips.org/mvc-pattern.html

For more information about Struts, visit:
http://jakarta.apache.org/struts/index.html

For more information about XML, visit:
http://www.w3.org/xml

For IBM Glossary of Computing Terms, visit:
http://www-3.**ibm.com**/ibm/terminology/goc/gocmain.htm

For more information about Web Services and WSDL, visit:
http://www.w3.org/TR/wsdl

For more information about SOAP, visit:
http://www.w3.org/TR/SOAP/

Appendix: e-business technologies

WebSphere Studio Enterprise Developer leverages and supports a number of emerging technologies that are becoming industry standards in the effort to create e-business Web applications. Some examples of these technologies are: XML, Struts, WSDL and SOAP. A brief description of each follows.

Extensible Markup Language

Extensible Markup Language (XML) is a markup language for documents containing structured information. Structured information contains both content—text and graphics—and labels the content (for example, content in a section heading has a different meaning than content in a footnote). Almost all documents have some structure. XML specifications define a standard way to add markup to documents. XML is a subset of Standard Generalized Markup Language (SGML). XML does not exploit the complex, less-used parts of SGML, making it much easier to:

- *Write applications to handle document types.*
- *Author and manage structured information.*
- *Transmit and share structured information across diverse computing systems.*

XML was created so that richly structured documents could be used on the Web. HTML and SGML are not practical for this purpose. HTML comes bound with a set of semantics and does not provide arbitrary structure. SGML provides an arbitrary structure, but it is too difficult to implement for a Web browser.

Struts

Struts is a set of cooperating classes, servlets and JSP tags that comprise a reusable MVC2 design. The definition implies that Struts is a framework rather than a library. Struts also contains an extensive tag library and utility classes that work independently from the framework.

Client browser

An HTTP request from the client browser creates an event.

Controller

The controller receives the request from the browser and makes the decision where to send the request. With Struts, the controller is a command design pattern implemented as a servlet. The struts-config.xml file configures the controller.

Business logic and model state

The business logic updates the state of the model and helps control the flow of the application. In Struts, this is accomplished with an action class as a thin wrapper to the actual business logic. The model represents the state of the application. The business objects update the application state.

View

The view is simply a JSP page. There is no flow logic, no business logic and no model information – just tags.

Web Services Description Language (WSDL)

WSDL is an XML format for describing network services as sets of endpoints that operate on messages containing document- or procedure-based information. The messages are described abstractly, and then bound to a network protocol and format that define an endpoint. Related endpoints are combined into abstract endpoints, or services.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstraction of endpoints and messages is separated from the network deployment or data format bindings. This allows the reuse of abstract definitions – messages that are abstract descriptors of data being exchanged, and port types that are abstract collections of operations. The protocol and data format specifications for a particular port type comprise a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service.

The following elements define a network service:

- *Types – a container of data type definitions*
- *Message – an abstract, typed definition of data*
- *Operation – an abstract description of an action supported by the service*
- *Port type – an abstract set of operations supported by one or more endpoints*
- *Binding – a protocol and data format specification for a specific port type*
- *Port – an endpoint defined as the combining of a binding and network address*
- *Service – a collection of related endpoints*

Simple Object Access Protocol (SOAP)

SOAP provides a protocol for exchanging structured and typed information between peers in a decentralized environment using XML. The protocol consists of three parts:

- *A framework definition for describing the content of a message and how to process it*
- *A set of rules for expressing instances of application defined data types*
- *A convention for representing remote procedure calls and responses*

SOAP does not itself define any semantics. Instead, it defines a simple mechanism for expressing semantics by providing a modular packaging model and mechanisms for encoding module data.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-02

G325-2008-00