Improving software development capability
05/20/03

Rational. software

IBM® Rational® Rapid Developer Automated Construction

Joseph G. Noonan
jgnoonan@us.ibm.com

Table of Contents

Introduction

This document describes the IBM® Rational® Rapid Developer Automated Construction System, its origins and its unique capabilities. The document does not presume any prior knowledge of Rapid Developer and is intended to provide the reader with a sound understanding of the code generation artifacts and patterns.

A Brief History of Automated Construction

Early Approaches

Automated code construction has been around for a while. Early efforts accelerated development by generating basic code structures (e.g., procedure and function stubs, class definitions, and other simple code artifacts). Additionally, database generation was usually found in tools used by DBAs, and application developers were left to their own devices regarding test database and data generation.

Over time, these tools became more sophisticated, but in the area of code generation, the market did not move much beyond their humble beginnings. Tools like Rational Rose incorporated frameworks such as Microsoft MFC and the Sun Java JDK into the modeling environment. Application developers could leverage these frameworks during modeling, and then generate class and method stubs with the appropriate references. Typically, only one technology could be targeted at a time due to the level of integration between the framework and the model.

The Rational Rapid Developer Approach

Rapid Developer has changed the course of code generation dramatically. In 1995, Arun Gupta, the founder of DataEase, and a core group of software developers, began to build a more complete approach to code generation. The basic premise was to generate fully functional applications from a set of models. *The technology selections would be isolated from the models so that application developers could rapidly move from one technology to another, at the same time leveraging the power of the target technology*. Rapid Developer was the result of this effort. Rapid Developer, like Rational Rose, provides a model-based approach to development. Application developers create class, user interface, process, site, message, code and transaction models that are technology neutral. Custom business logic, which only comprises 5-8% of a typical Rapid Developer-generated application, is written using the Java programming language. The Rapid Developer Runtime API allows application developers to create this business logic using a standardized interface that works under all supported technologies. As new technologies are added, the interface stays the same. Additionally, as new patterns of construction emerge, the template-based construction system is augmented to support them. The remaining sections of this paper describe the construction system and the generated patterns in greater detail.

Rapid Developer Automated Construction System

Overview

The Rapid Developer Automated Construction System is a state-of-the-art mechanism that converts model elements such as classes, pages, messages and components into fully functional application artifacts. The key elements to this unique product are:

- Technology neutrality
- Template-based construction
- Technology optimization
- Transaction-centric construction and the ObjectSpace (defined below)
- Application developer productivity and flexibility
- Deployment flexibility

Technology Neutrality and Optimization

From the beginning, a guiding premise of Rapid Developer was to provide a development tool that would allow developers to create applications and not worry about constantly changing technology. Since software technologies are always evolving, application developers need tools that keep pace with change. The Rapid Developer modeling system insulates application developers from the target technologies. Application developers focus on the business problem and use the Rapid Developer modeling environment to describe the application. Business rules and methods are written using the Java programming language, which can be run in any environment that supports a Java virtual machine. The Rapid Developer Runtime API is used to write the business rules and is abstracted to shield application developers from the specific implementation for a given technology. Application developers can choose to write directly for a given technology, but lose the ability to automatically migrate as new technologies are supported in Rapid Developer.

While the API provides a common interface, it does not provide a common implementation. Each target technology is constructed in an optimized manner, using the unique capabilities of the selected technology. For instance, the deletion of a parent record and its related children uses a trigger to perform a cascading delete, if the technology supports this capability.

Architectural Neutrality

Design patterns provide application developers with architectural neutrality. This means that application developers are free to pick and choose from different construction patterns for a given scenario. They can then test and perhaps select a new pattern if the selected one does not fit for the business transaction. Rapid Developer currently supports many industry-recognized patterns and more are added with each new release.

Template-based Construction

The Rapid Developer construction system employs a template-based approach to generation. For each function of a given technology, there is a template that controls

generation. This template is tuned for the given target technology. As new technologies are supported or their capabilities extended, templates are added or modified to provide construction support. The constructed code is performance and load tested to ensure that it is fully optimized.

Transaction-centric Construction

Another governing principle behind Rapid Developer is modeling of pages, messages, components, and Web services focuses on the specific business transaction being performed. As application developers define a page, message, or component they create an ObjectSpace.

ObjectSpace

The ObjectSpace is a Rapid Developer construct that defines the elements of the business transaction. The application developer creates an ObjectSpace automatically by placing controls on the page and tying them to classes, attributes or methods. The developer can view the ObjectSpace by selecting Page Properties and clicking on the ObjectSpace tab (see Figure 1). In this case, the application developer uses the OrderHeader class as the root for the ObjectSpace. The OrderItems class is also projected and has a "many" relationship to OrderHeader. Additionally, each OrderItem class has a related SellerProduct class. The applications developer has also projected a number of attributes and methods into the ObjectSpace as well.

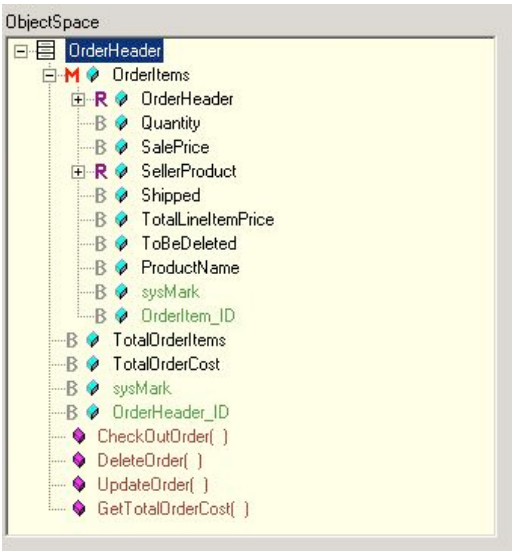

Figure 1 - ObjectSpace Design

A class may be defined to have any number of attributes (e.g., 20 attributes). For a given business transaction, however, the application developer may only need three of the attributes. When designing the ObjectSpace for the page, the application developer selects the class and projects the three attributes needed. During construction, a class is constructed that contains only the projected attributes. This means that only the data needed for the business transaction is brought in from the database, reducing network traffic and improving scalability. Custom database access functions are generated based

on the ObjectSpace class and page type definitions. For example, if a page type is read-only, no insert, update, or delete data access methods will be generated, but the load functions are generated. Rapid Developer supports the following page types: read-only, enter-only, full, search, and no data access. Rapid Developer will generate the following system methods when needed by the type:

ObjectSpace Methods

- Construct() – This method builds the ObjectSpace and loads the data for it using the various class methods

Class Methods

- getID() – returns the ID (primary key value) of the current record
- insert() – inserts the current record into the appropriate database table
- update() – updates the database record with the current values for the record
- delete() – deletes the current record
- validate() – executes all validation rules that are associated with projected attributes
- loadByID() – loads the record from the database that matches the ID (primary key value)
- loadBy Selection() – loads the record that matches the SQL selection criteria passed to the function
- loadObjectBySelection() – loads a set of objects that match the SQL selection criteria passed to the function
- load<related object>() – loads the selected related object using the foreign key for the related object or objects. From the example in Figure 1, a loadOrderItems() function would be automatically generated for the ObjectSpace.

Rapid Developer uses a declarative transaction model that defines deployed session beans (J2EE), entity beans (J2EE), or COM objects (MS-DNA) to require transactions. When any method is called, the container automatically starts a transaction if one does not exist already. Any nested method calls would participate in the original transaction.

Application Developer Productivity and Flexibility

Rapid Developer is revolutionary in its approach to development. With its rich modeling environments, application developers create a set of models that are transformed into a working application. Developers focus on the application design, while Rapid Developer handles the application construction and deployment. An application developer with basic Java programming skills can create complex *n*-tier transactional applications with very little knowledge of the target deployment environment. *In addition, if application developers need to access the technology directly, Rapid Developer does not prevent them from doing so. Also, application developers can override the code that Rapid Developer generates.*

Deployment Flexibility

Rapid Developer Partition Architect provides unmatched flexibility in deploying the artifacts that Rapid Developer generates. With Partition Architect, application developers

can divide the application artifacts into archives along functional boundaries (e.g., order entry, purchasing, inventory) or along application tiers (presentation, middle tier, database) or any other way that makes sense for the application. For instance, session beans that represent high-use business transactions can be packaged as a deployment archive and deployed on a high volume server.

Construction

Rapid Developer supports the design and construction of pages, messages, components and Web services using either J2EE- or MS-DNA-based technologies. Page development is supported using the full-featured Page Architect. Messages are used to integrate disparate applications and are developed using the Message Architect. Components and Web services allow application developers to encapsulate reusable code and expose interfaces to them. The following sections describe the kinds of artifacts Rapid Developer generates for each of the major technologies supported.

Presentation Artifacts

Cascading Style Sheets (CSS)

Rapid Developer provides extensive theme and style support for the graphic artist. At construction time, application developers have the option to generate the styles as part of the generated code or separately as cascading style sheets. CSS are basically text files that are read each time a page is rendered. By using the CSS option, the site style can be changed in seconds with an editor.

Database Artifacts

Database Objects

Rapid Developer generates database objects such as tables, indices, primary and foreign keys, and triggers (where supported) based on the DBMS select. This is done based on the definition of classes, attributes, and the relationships between classes as defined in the Class Architect. Application developers can even define classes to be in different databases (i.e., Customers may be in an Oracle DB where Orders are in DB2 on the System 390).

Stored Procedures

Based on the design of the page, message or component, application developers can choose to generate stored procedures to provide the data access CRUD function set (refer to ObjectSpace section above). These stored procedures are optimized for the target DBMS.

Dynamic SQL (Java)

The other method supported by Rapid Developer for accessing the database is the use of Java-based dynamic SQL. All database access methods (CRUD function set) are generated as Java calls that are optimized for the target DBMS.

Security Artifacts

Role-based Security

Based on selections in the page, Rapid Developer supports and generates role-based security code. Application developers can control access to pages and controls on a role-by-role basis. If LDAP is used, Rapid Developer Version 2003 will synchronize with the roles established in the LDAP server.

Data Encryption

Rapid Developer provides the capability to encrypt a variety of data for pages and messages. Application developers can encrypt URLs, parameters, data elements and message elements. Rapid Developer supports a variety of encryption algorithms and also allows the applications developer to create their own custom encryption methods.

Visibility Expressions

Rapid Developer application developers can use visibility expressions to determine if data is seen on a page or included in a message. The application developer writes these visibility expressions as business rules in Java.

Internationalization/Localization Artifacts

Resource Bundles

An exciting new capability in Rapid Developer Version 2003 is the ability to develop an application for multiple target languages. Rapid Developer provides extensive features to application developers to localize controls and locale elements within the application for any number of languages. Rapid Developer will generate the localized prompts and text elements as resource bundles. The applications developer can also configure the application to dynamically select the language based on the browser settings. The developer can also export the localized items into a variety of formats so they can be sent for translation or loaded into third-party translation tools.

J2EE Artifacts

Java Server Pages (JSP)

Java Server Pages, or JSPs, provide application developers with a technology for serving pages that contain dynamic content. JSPs provide a means to separate static content (HTML) from dynamic elements, which can be scripted using the Java programming language. Within the script sections of JSP pages, the application developer can draw on the full power of Java to access resources such as databases, message queuing systems, and Web services. By providing a separation between content and logic, application developers can make content changes without having to change Java code—as they must with Java servlet technology. Rapid Developer supports JSP-based applications by providing a variety of generation patterns for this technology. JSPs, by default, are deployed as part of the Web container.

Servlets

Java servlets are compiled Java classes that provide a platform-independent replacement for CGI scripts. Unlike JSPs, the content is not separated between static and dynamic, making them less flexible to JSP. Since all the content is managed in Rapid Developer, application developers are not as impacted as they would be if the servlets were hand-coded. They need only go into Page Architect and change the page and reconstruct to change page content. Servlets, by default, are deployed as part of the web container.

Session Beans

Session beans are used for business logic associated with pages, messages, and components. By default, these session beans are stateless. Rapid Developer allows application developers to generate stateful session beans for components, but not for pages or messages. Session beans are set to require a transaction. This means that if a transaction does not exist, the session bean container will start one. If one already exists, it will leverage the existing transaction. The home, remote and localHome interfaces are also generated.

Java Classes

In cases where application developers are not using application servers that support EJBs (such as Apache Tomcat), session beans will be generated as Java classes instead (servlet-only and JSP-only patterns). Since there is no container to manage transactions, they are not supported when Java classes are used.

Entity Beans

Entity beans are supported in Rapid Developer Version 2003. Application developers can select to use entity beans for data access on a class-by-class basis. Rapid Developer generates all the finder and getter/setter methods and deploys the entity beans to require a transaction, the same as the session bean. A value object is created and is passed to the ejbCreate() method on the home interface of the entity bean. The ejbCreate() method uses the setters to populate the value object which is returned to the caller with the requested record data. The home, remote, and localHome interfaces are also generated.

Microsoft DNA Artifacts

Active Server Pages (ASP)

Active Server Pages, or ASP, are generated when the user selects to use the Microsoft DNA technology. An ASP is generated for each page in an application and is used to invoke the main interface of the associated COM DLL deployed on Microsoft's Transaction Server.

MS-COM DLLs

MS COM DLLs are somewhat synonymous to the session bean insofar as their roles in a deployed application. MS COM DLLs are used to implement the business logic for a page, message or component. Rapid Developer automatically deploys and registers these COM DLLs to Microsoft Transaction Server (MTS). They are marked to require a transaction, the same as the session bean.

Page Construction

Rapid Developer supports the design and construction of complex Web pages through the use of the Page Architect. The Page Architect is a GUI tool similar to products such as Microsoft Front Page or Macromedia DreamWeaver. Application developers can drag and drop a variety of controls onto the Page Architect canvas to design the application pages. Howerver, Rapid Developer supplies a variety of data-connected controls such as grid, object table and object grid that can be rapidly added to a page and mapped to database tables or views. Additionally, these controls can be nested within one another to provide a master-detail view at any number of levels. The application developer can then construct and deploy the page using the Rapid Developer built-in construction and deployment system. The artifacts (i.e., servlets, JSP, session beans, java classes, entity beans, stored procedures) are constructed based on the technology patterns that the developer selects.

All pages constructed by Rapid Developer utilize the ObjectSpace mechanism (see "Transaction-centric Construction", above). This mechanism ensures that objects created by page construction contain only the attributes and methods needed by the page—greatly reducing the size of the objects and also limiting the amount of data the page retrieves from the database. This reduces network traffic between the database and the middle tier. In addition, the Rapid Developer framework provides technology-neutral implementations of HttpSession, HttpRequest, HttpResponse, and HttpContext that application developers can access using Rapid Developer custom methods that provide complete session functionality to the Rapid Developer application developer.

Rapid Developer supports J2EE and Microsoft DNA construction patterns. J2EE offers the most flexibility in pattern selection within Rapid Developer. There are two subsets of patterns under J2EE: servlet-based patterns and JSP-based patterns. These patterns can be matched with different database access methods to fine-tune an application transaction for optimal performance. JSP patterns and servlet patterns cannot be mixed within the same application at this time.

J2EE Patterns

J2EE or Java 2 Platform, Enterprise Edition, has brought sweeping changes in the computing industry. Using the Java programming language as the base, with its platform neutrality, J2EE-compliant application servers provide an infrastructure for running applications on a wide variety of platforms. These application servers run on everything from Intel-based servers to IBM zSeries mainframes. These application servers provide a built-in set of operation and management services for applications. Memory management, security, thread management, transaction management are but a few of the services these platforms provide. Rapid Developer creates Java components called Enterprise Java Beans (EJBs) that are deployed on application servers. Application developers write the basic business logic, and the container (EJB) provides the service interfaces to the application server facilities. This saves application developers a significant amount of effort because they don't need to write all theses services, which is difficult to write and maintain.

Even with the advent of these sophisticated application servers, the development of Java applications require a level of developer sophistication not easily found in the industry. Application developers are challenged to know the techniques for developing for J2EE, let alone the particulars for creating EJBs or other artifacts on all the different application servers that are J2EE-compliant. This is where Rapid Developer shines in and takes J2EE development to the next level. Through its sophisticated construction and deployment capabilities, Rapid Developer completely automates the construction and deployment of the J2EE artifacts for an application. Developers can select patterns for the application and Rapid Developer also provides a means to construct a different pattern for each page. These patterns fall into two mutually exclusive groups: JSP (Java Server Pages) and Servlets. Within these groups, application developers can select different construction and deployment patterns based on the specific needs of the transaction (page). The following sections describe these two groups and their patterns supported in Rapid Developer.

JSP Only

The JSP-only pattern is ideal for non-transactional, read-only dynamic pages as well as static pages (no data access). With the JSP-only pattern, each page is generated as a fully functional JSP. The ObjectSpace associated with the page is generated as a series of classes that match the classes and attributes projected during page design. Users access the JSP page through their browser. The JSP page does the work of rendering the static and dynamic elements of the page. Since the focus of this pattern is displaying data or static content, there are no transactions supported through generation. Application developers can add user transactions in custom business methods, but they need to manage all the elements of the transaction, including rollback functions. Access to the database is done through the ObjectSpace classes and their associated CRUD (Create, Retrieve, Update, Delete) function sets.

Page elements such as style and control definition are generated as static JSP. Elements such as projections, role-based security and custom business logic are generated as the dynamic (Java) content of the page. Figure 2 shows the basic flow of a request:

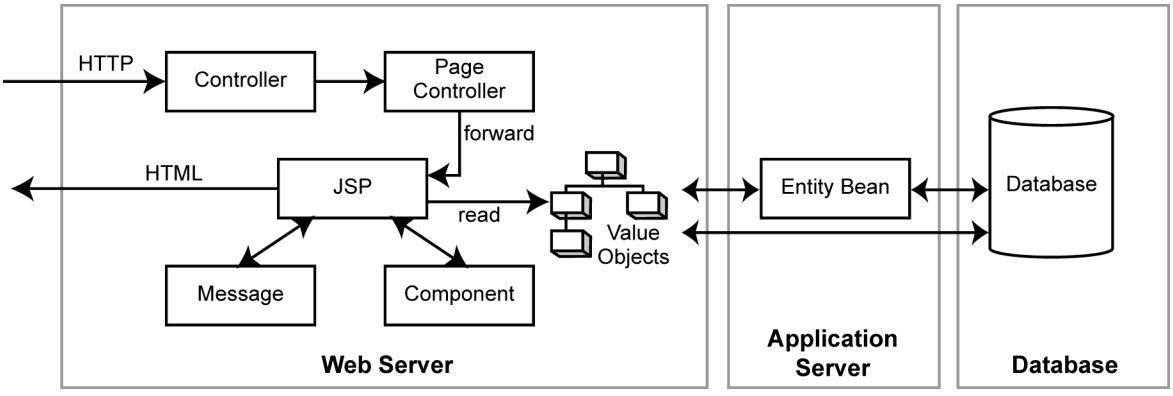

Figure 2 – JSP-Only Construction Pattern

The controller, deployed as a servlet in the Web container, determines which page controller to create. There is one controller per JSP application deployed using Rapid Developer. If the page is used to display data from the database, the page controller creates the data set for the page. The JSP page dynamic elements (scriplets) format the data for display. The data is retrieved from the database either using dynamic SQL (Java) or stored procedures.

JSP with Entity Beans

In systems that support entity beans, the data access methods can be generated to interact with the entity beans. Entity beans are deployed to the application server and perform all the data load/manipulation operations. Entity beans are generated to require transactions. Since JSP-only pages do not create transactions, the entity beans start transactions for every call that is made to them from the ObjectSpace classes. If an error occurs in these transactions, the ObjectSpace initiated method fails, and the JSP presents the appropriate error message to the user.

JSP Model II (MVC)

The JSP Model II pattern uses a Model-View-Controller paradigm in which the static and content elements are separated from the dynamic aspects of the page. HTML designers can code the layout and content using JSP, and Java programmers can create Java elements for data retrieval and formatting. Content (view) can change independently of the code (controller) used to retrieve and format the content from the data (model). Rapid Developer supports this pattern when JSP is selected as the construction pattern and the target platform is an application server that supports Enterprise Java Beans (e.g., IBM WebSphere, BEA WebLogic). Application developers can mix and match pages to use this pattern or the JSP-only pattern based on the needs of the page. This pattern is useful when the page has transactional requirements such as insert, update, or delete functions.

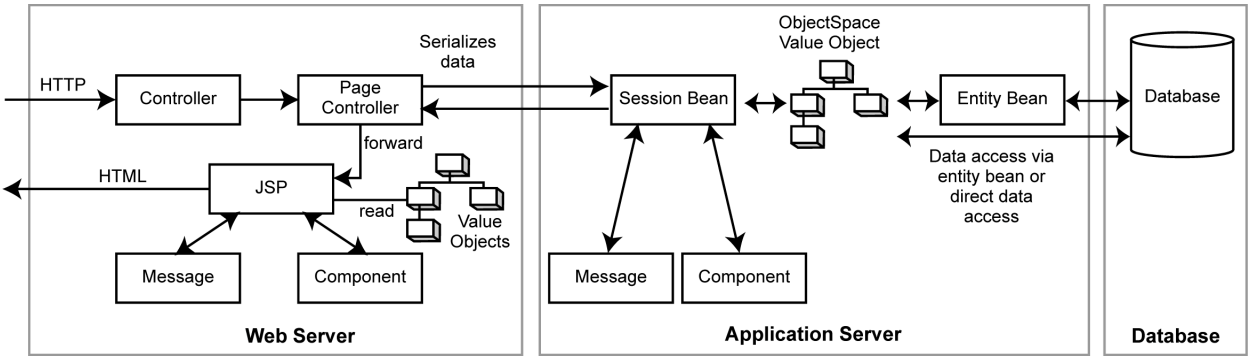

Figure 3 - JSP Model II Construction Pattern

In this pattern (see Figure 3), application developers can separate custom business logic between the presentation and business tiers. Presentation methods can be used to control display elements, while business tier methods can perform a variety of business-rule operations on the data being sent to or retrieved from the database. As with the JSP-only pattern, a controller creates a page controller specific for the page. The page controller instantiates the session bean associated with the page. The session bean exposes two

methods called requestData(), to retrieve data from the database, and submitData(), if the request is a post. These methods populate the ObjectSpace with the data and execute any custom business logic the applications developer may have created. Once the session bean has completed its work, the data is forwarded to the JSP page by the page controller. The JSP page then formats and displays the data.

JSP Model II (MVC) with Entity Beans

This pattern is a slight variation of the previous pattern. In this pattern, the classes from Rapid Developer are represented as entity beans and are deployed on the application server. The session beans and entity beans are deployed to require a transaction. In this case, the session bean container starts the transaction and any calls to an entity bean participate in the main transaction. The applications developer gets all the distributed features of entity beans without having to know specifically how to code and deploy them.

Servlet-Only

The servlet-only pattern is analogous to the JSP-only pattern. The servlet-only pattern is used when the page is a read-only or static page (no data access). The page is compiled and deployed as a servlet in the web container. The ObjectSpace is created as a set of classes that are called by the page servlet. The ObjectSpace classes contain methods to interact with the database to provide data for the servlet to display. As with JSP-only, the servlet-only pattern does not support automatically generated transactions. However, application developers can write custom business methods that contain user transaction code. Additionally, application developers can call third-party components or other session beans from within these methods (see Figure 4).

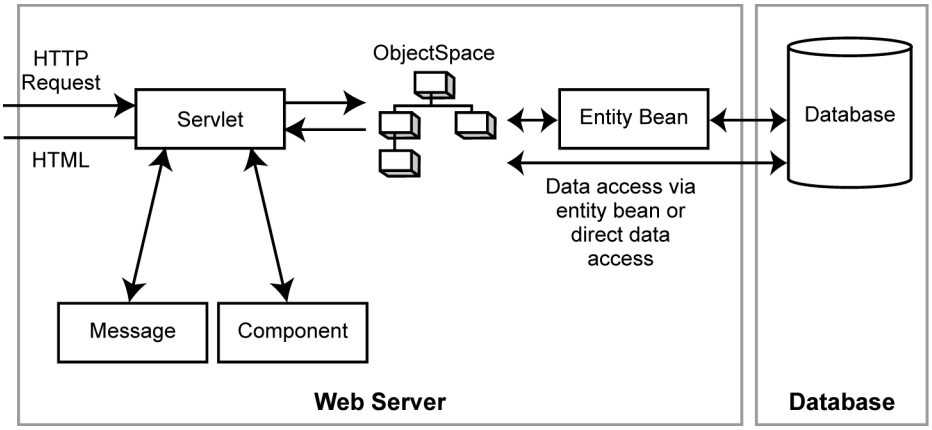

Figure 4 – Servlet-Only Construction Patterns

When a page is constructed as a servlet, the primary method is the generate() method, which is auto-generated based on the design of the page. The generate() method orchestrates all elements of page display, retrieving the data from the database, executing custom business methods on the data and then displaying the page by outputting the HTML to the response object.

Servlet with Entity Beans

If the applications developer elects to use entity beans for data classes, the data access methods are generated to interact with the entity beans. Entity beans perform all data load/manipulation operations and are deployed to the application server. As stated previously, entity beans are generated to require a transaction. Since servlet-only pages do not create transactions, the entity beans start transactions for every call that is made to them from the ObjectSpace classes.

Servlet with Session Bean

In this pattern (see Figure 5), a servlet and a session EJB are generated for each page. The servlet is a lightweight controller that is used to instantiate the home interface of the session bean and invoke the generate() method in the session bean. All the presentation logic, business logic and data-access logic are located in the session bean.

This is a great pattern for pages that need maximum performance. All the tiers are contained within the session bean; no data needs to be serialized between tiers. This marshalling and demarshalling of data is very time consuming, particularly when larger datasets are involved. Transactions are supported for this pattern and are automatically initiated by the session bean container.

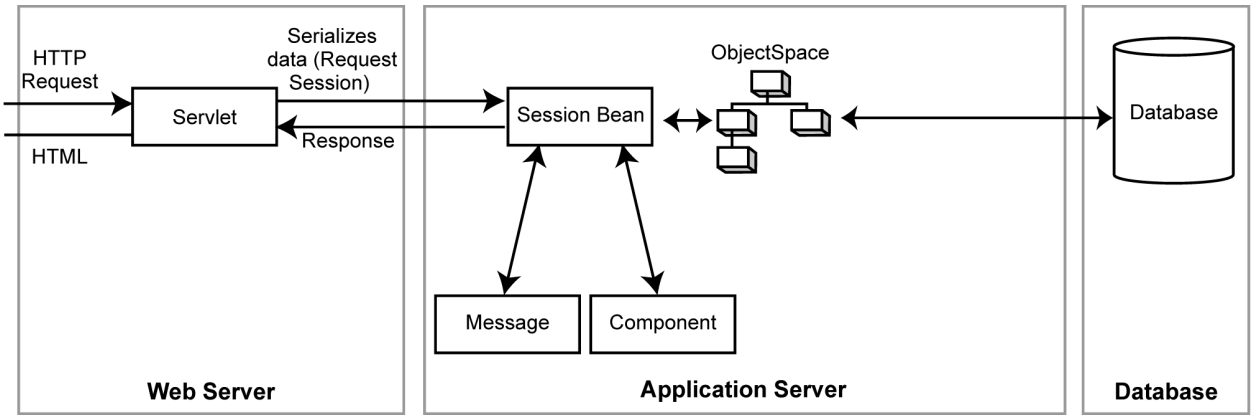

Figure 5 - Servlet With Session Bean Construction Pattern

Servlet with Session Beans and Entity Beans

This pattern (see Figure 6) is a slight variation of the previous pattern. In this pattern, the classes from Rapid Developer are represented as entity beans deployed on the application server. Entity beans are deployed to require transactions. In this case, the session bean starts the transaction and any calls to an entity bean participate in the main transaction. If anything should fail at the entity bean level, the calling method is notified using an appropriate error message. Any custom methods written for the business tier automatically participate in the transaction for the page.

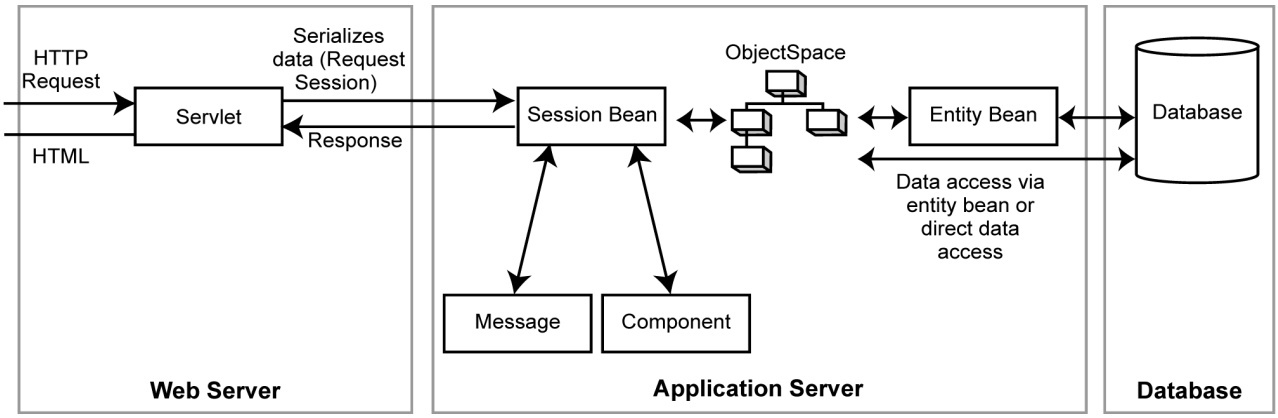

Figure 6 - Servlet With Session Bean and Entity Bean Construction Pattern

MS-DNA Patterns

Rapid Developer provides support to the Microsoft COM applications developer through the generation of pages as COM DLLs that are deployed onto Microsoft Transaction Server (MTS). Pages are compiled using the Microsoft SDK for Java version 4.0 (jvc).

ASP with COM DLL on MTS

This pattern (see Figure 7) is almost identical to the servlet/session bean pattern supported for J2EE in which the ASP is analogous to the servlet and the COM DLL is the session bean. Using MTS mechanisms, transactions are supported in the generated page. Java is used as the programming language and for the most part; application developers can switch between MS and J2EE technologies as long as they have written to the common Rapid Developer Runtime API.

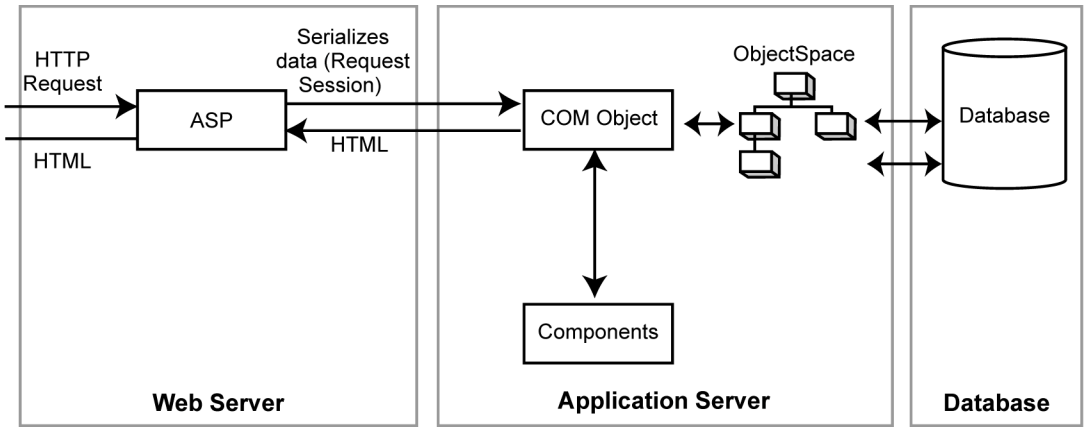

Figure 7 - ASP with COM DLL on MTS Construction Pattern

Message Construction

In addition to pages, Rapid Developer also supports the design and construction of messages. Messages are generally used as a means of integrating disparate applications. This is commonly referred to as enterprise application integration or EAI. In the past EAI projects could easily turn into quagmires because of the different needs of each system and the time it took to hand code messages and their respective handlers. Rapid

Developer eliminates that headache by providing a rich message-modeling environment at design time, and automated construction of the tedious elements of EAI applications (e.g., message parsing, transport interfacing).

At the heart of Rapid Developer's messaging support is the Rapid Developer Messaging Runtime API. This API provides a technology-neutral interface to queues, messages, message envelopes, and transports. Application developers can change the underlying transport (e.g., JMS to WebSphereMQ) without a lot of expensive re-coding. Rapid Developer currently supports WebSphereMQ, Java Messaging Service or JMS, Microsoft Message Queue (MSMQ), and TIBCO transports. It is important to note that several application server vendors bundle JMS transports as part of the application server, and these are supported through the JMS interface.

As with pages, the ObjectSpace plays a central role in the design and runtime execution of a message. During message design, the applications developer selects the objects and attributes that are needed to perform the tasks associated with the message. The Message Modeler then allows application developers to load an XML DTD and match the elements of the DTD to the ObjectSpace classes and attributes. This creates a map that Rapid Developer uses during code generation to create the parse code for the message. Additionally, the Message Modeler allows the applications developer to generate a DTD from an ObjectSpace definition to give to partners who may be receiving messages generated by the application. The Message Modeler also provides a means of performing transforms on the data either as it is received or when it is transmitted, or both.

At runtime, based on the design and needs of the messaging application, an inbound message can go through the following process:

- Retrieve message from queue
- Route message to appropriate session bean handler
- Decrypt message (optional)
- Determine role-based security requirements for the message (optional)
- Map message elements to ObjectSpace using design-time map
- Perform any necessary transforms (optional)
- Execute custom business methods
- Signal success or failure (Exception handling can be done via system or custom error handlers)

It is important to note that Rapid Developer automatically generates the code for all of the above steps for the specified transport with the exception of applications developer-written custom business logic. An outbound message goes through the following process:

- Create outbound message session bean from client call (page or other mechanism)
- Build ObjectSpace and load appropriate data
- Perform any preprocessing of data (optional)

- Perform any transforms or localization (optional)
- Execute custom business methods
- Create message (XML or plain text) from ObjectSpace map definition
- Encrypt message (optional)
- Send message to appropriate queue or retrieve XML representation of message

As with the inbound message, the only code application developers need to write are the custom business methods. This progress is supported by a large number of pre-populated code templates. The rest is generated from settings chosen in Rapid Developer during design time.

J2EE Message Patterns

As previously stated, Rapid Developer supports J2EE application development on a variety of compliant application servers. Currently, Rapid Developer supports IBM WebSphere Application Server Versions 3.5, 4.0 and 5.0, BEA WebLogic 5.x to 7.x, Oracle 9iAS Releases 1 and 2, and Apache Tomcat (servlet- and JSP-only) Version 4.0.x. Through the JMS interface, Rapid Developer supports message transports for WebSphereMQ, WebLogic, and Oracle 9iAS in addition to other third-party JMS providers such as SonicMQ.

Message as a Session Bean

In this pattern, the message and its associated elements (e.g., parser, transforms, encryption) are created as a single EJB session bean. The interface to this bean varies based on decisions made by the application developer during design time. The two basic interface methods are processIn() or Out(). These represent handling of inbound and outbound messages, respectively. Inbound message session beans receive the message through the processIn() method. This method extracts the message body, performs decryption on it if necessary, parses it and maps the data elements to their associated ObjectSpace elements. Transforms supplied by the application developer at design time are also performed. Once the data from the message is mapped to the associated ObjectSpace elements, the custom business rules that application developers create are called to process the message.

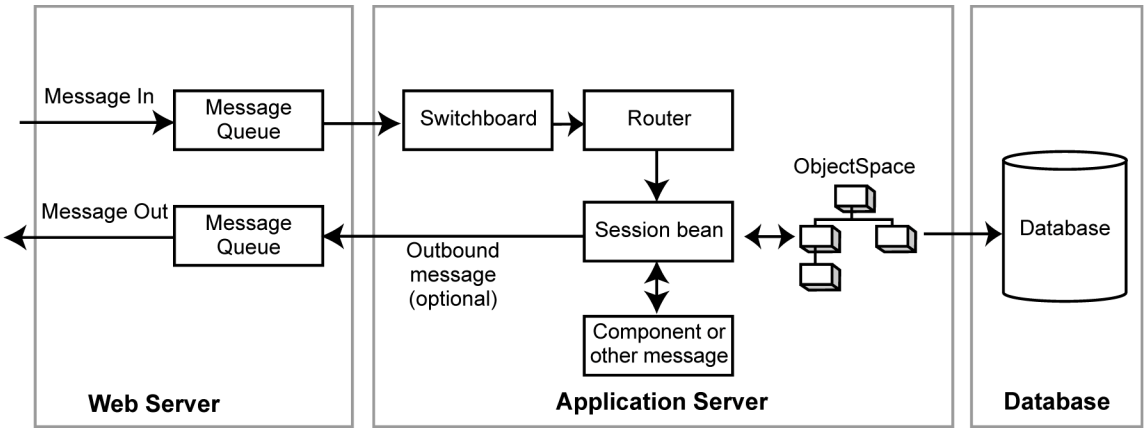

Figure 8 – Message as a Session Bean

For outbound messages, the reverse occurs. The Out() method is called to create a message from the ObjectSpace based on a DTD definition. Transforms are performed and an XML message is created and encrypted if need be. It is then sent to the designated queue. All of these operations are done within the confines of a Rapid Developer-generated transaction. It is important to note that these transactions only cover the processing of the message and do not cover the removal of the message from the queue or the sending to the queue. This is a limitation only for inbound messages when using the Rapid Developer Switchboard runtime component. This limitation does not exist when using Message-driven beans. The custom methods in the session bean created by application developers can also invoke other session bean messages or components as part of the transaction associated with the message.

Message as a Java Package

In order to support lightweight environments such as Tomcat, Rapid Developer can generate messages as a set of Java packages rather than as a session bean. In this pattern, transactions can only be supported through custom business methods written by the applications developer. The ObjectSpace is also represented as a series of classes with which the message classes interact. All other messaging operations such as security, encryption and transformations are performed in the same manner as the session bean pattern.

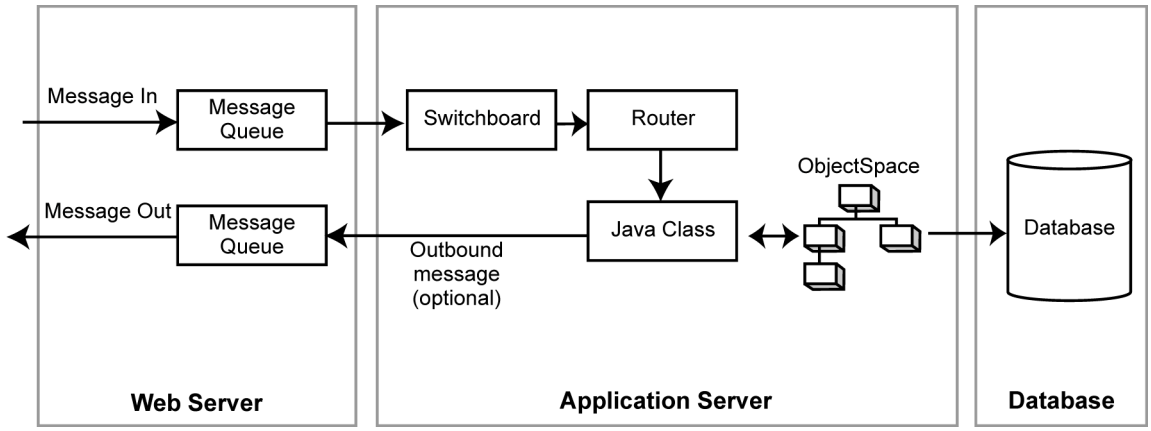

Figure 9 - Message as a Java Package

Message-driven Bean

Message-driven beans or MDBs are supported by a variety of application servers that support the J2EE 2.0 specification. MDBs provide a variety of services that are time consuming to develop. MDBs are only supported using transports that support JMS. The MDB is associated with a specific queue and the container provides a listener service that monitors the queue and detects when a message arrives. The MDB opens a transaction and then takes the message out of the queue. The MDB then invokes the single interface method *onMessage()* from which all further processing of the message is done. In the case of Rapid Developer-generated MDBs, the *onMessage()* function constructs the ObjectSpace and performs all other processing in the same manner as with a session bean message. The advantage provided by MDBs is that the transaction can be started before the message is taken from the queue, so that message retrieval is included in the transaction. If the transaction should fail for some reason, the message is placed back in the queue for retry.

MS-DNA Message Patterns

Rapid Developer supports generation of messages in the MS-DNA environment in a similar manner to the J2EE session bean pattern. In addition, Rapid Developer supports construction for the Microsoft Message Queue (MSMQ) transport as well as the COM interfaces to IBM WebSphereMQ (formerly IBM MQSeries), TIBCO and TCP/IP.

Message as a COM DLL on MTS

In this pattern, the message and its associated elements (parser, transforms, encryption, etc) are created as a single MS COM object that is deployed to Microsoft Transaction Server (MTS). The interface to this COM object is basically the same as with the J2EE session bean pattern. The two basic interface methods are processIn() and Out(). These methods handle inbound and outbound messages respectively. Both XML and plain text messages are currently supported. All of the standard operations (e.g., parsing, encryption/decryption, security, and transformation functions) are performed within the boundaries of a Rapid Developer-generated transaction. These transactions only cover the processing of the message, and do not cover the removal of the message from the queue,

or sending to the queue, which are covered as part of the transaction provided by the transport. The custom methods in the COM DLL created by the applications developer can also invoke other messages or components as part of the transaction associated with the message.

Queue Monitoring and Message Routing

The previous sections described the patterns for message processing. Using the Rapid Developer Switchboard runtime component, Rapid Developer also provides a means for retrieving messages from their queues and routing them to the appropriate message handling code. The following sections describe what Rapid Developer supports.

Rapid Developer Switchboard (J2EE and MS-DNA)

Rapid Developer Switchboard is a runtime component that provides queue-monitoring services and asynchronous message processing capabilities (e.g., listening, message retrieval and routing). Switchboard is a multithreaded mechanism that launches listeners for all of the queues that it is assigned to monitor. As messages come in, the listener removes the message from the queue and signals Switchboard that it has work to do. Switchboard then launches a new listener. This listener starts a transaction and uses the Rapid Developer RouterFactory runtime component to create a router for the message. The router matches the message with the appropriate message handle (a session bean, Java package, or COM DLL). The message handler opens a separate transaction and then finishes processing the message. If the either transaction fails, the message is placed in an error queue, and is not placed back into the original queue for retry.

Message-driven Beans (J2EE only)

The Message-driven Bean, which is only available on certain J2EE platforms, provides the same basic services as Switchboard, with the added advantage that the transaction includes message retrieval, whereas with Switchboard it is handled as two separate transactions.

Component Construction

Component-based software development has increased in importance as the cost of software development has risen. Creating well-defined, reusable software "parts" allows application developers to reuse and leverage existing code for common functions. Rapid Developer supports this development paradigm by providing component modeling coupled with support for web services (although Web services are not required).

J2EE Patterns

As with pages and messages, Rapid Developer supports the design, compilation and deployment of components. Components can be created as a stateful or stateless session bean, or as a Java package. These components can be invoked by any other Java code, specifically in presentation or business-tier methods developed in Rapid Developer.

Session Bean Components

Consistent with the other artifacts that Rapid Developer generates, the design of a component is centered on the definition of an ObjectSpace. The applications developer can create an ObjectSpace, projecting the needed classes, attributes, and methods. Additionally, the applications developer can create ObjectSpace-level methods that can access all of the classes in the ObjectSpace. The applications developer can also access global methods and attributes that can span components. The applications developer can then identify the methods to expose as an interface for the component. The interface can be a standard interface or it can be generated as a Web services interface. The called interface method does not start a new transaction, but participates in the transaction of the calling method.

Stateful vs. Stateless Session Bean Components

Rapid Developer permits the designation of a session bean component as either stateful or stateless. A stateful session bean maintains the instance variable states throughout the conversation between the client (caller) and the session bean component. In a stateless component this does not occur.

Why would an application developer use one over the other? Generally, stateless components are used for calculating a value when retaining the state through multiple calls is not needed or desired. However, in a case where an aggregate is being calculated, the state must be maintained. Stateless session beans provide performance advantages so care should be used when determining at design time which type to use for a given component.

Java Package Components

As with pages and messages, components can also be generated as Java packages. This pattern is obviously always stateful. This pattern is useful for systems that do not support EJBs such as Tomcat. In this case, transactions must be initiated by the calling method.

MS-DNA Patterns

Rapid Developer also supports the generation of components as COM DLLs deployed onto Microsoft Transaction Server or as a Java package.

COM DLL Component

As with components under J2EE, MS–DNA-based components can be marked as either stateful or stateless. This decision is made by the applications developer at design time and needs to be made based on the needs of the component. As described previously, stateless components do not maintain the state of instance variables between calls where stateful components do. The COM objects are deployed to either participate in a transaction or start one if one does not exist.

Java Package Component

As with pages and messages, components can also be generated as Java packages. This pattern is obviously always stateful. In this case, transactions must be initiated by the calling method.

Web Services Construction

Web services allow applications to access methods over the Internet through an infrastructure in which the Web service definitions (WSDL) are published to directory servers (UDDI) where application developers can then discover them. Rapid Developer supports the design, publication, and discovery of Web services. When the applications developer uses a Web service method in Rapid Developer, the method is marked as such. When constructed, Rapid Developer replaces the marker with code to generate the appropriate SOAP messages to communicate with the web service.

J2EE Patterns

Rapid Developer supports the design of Web services through the component designer. When application developers add methods to the component's ObjectSpace, they can make the method part of the component interface. They also can choose to make the method interface a web service. Once the component is defined, it can be compiled and deployed. During this operation, the application server generates the WSDL specification file, if this is supported (otherwise, Rapid Developer generates the WSDL). Web service components are always generated and deployed as stateless session beans that require transactions.

Application developers can also publish the constructed Web services to a UDDI server, which is a registry of Web services that application developers can access to discover and use existing Web services. The applications developer provides the URL and credentials to the target UDDI server. Based on the credentials, Rapid Developer publishes the Web service to the first business name it finds associated with the credentials.

Rapid Developer also provides a means for application developers to discover Web services by providing the location and name of the associated WSDL for the Web service. Once they have been discovered, Web services appear in the selector list in the Rapid Developer logic editor. The applications developer can select the web service and it will be added to the code currently displayed in the logic editor with a "[Web Services]" marker tag. During construction, this marker tag is replaced with Java code that will generate the appropriate SOAP messages to invoke the web services. As stated previously, all session beans deployed by Rapid Developer are marked to require a transaction.

Conclusion

Rapid Developer provides an innovative approach to automated code construction of enterprise-class, *n*-tier business applications. By separating application-specific models from technology-specific code patterns, Rapid Developer enables developers to substantially reduce the amount of time-consuming, handcrafted code required to implement applications. Over the lifetime of an application, less code written results in less code to maintain, further enhancing the project team's responsiveness to enhancement and maintenance needs and maximizing the organization's ROI for its application development tooling. Additionally, by having a development tool that provides the generation capabilities of Rapid Developer, developers are assured that applications are consistently produced with the highest quality, and readily targeted to modern deployment technologies.

IBM software integrated solutions

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

• DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.

• Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.

• Tivoli® software helps you manage the technology that runs your e-business infrastructure.

• WebSphere® software helps you extend your existing business-critical processes to the Web.

• Rational® software helps you improve your software development capability with tools, services, and best practices.

Rational software from IBM

Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

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

Printed in the United States of America
01-03 All Rights Reserved. Made in the U.S.A.

The IBM home page on the Internet can be found at **ibm.com**

Part No. TP800