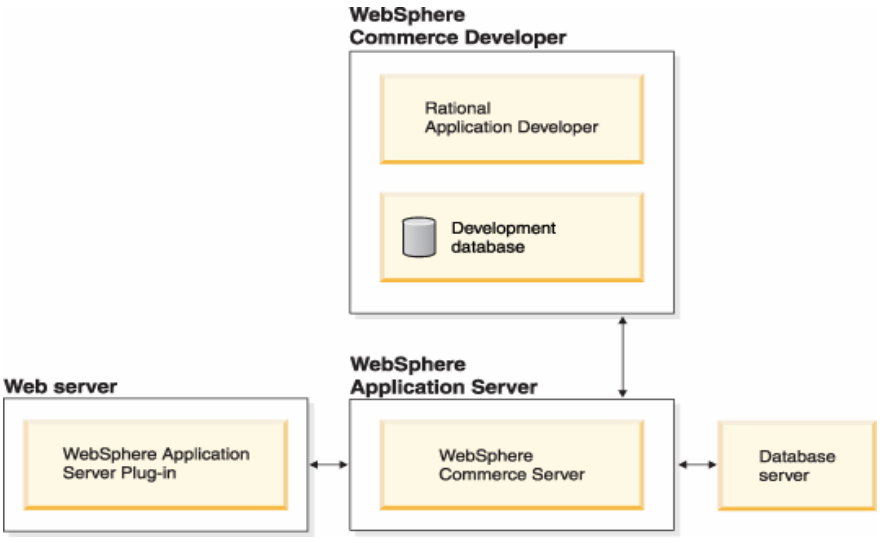
This presentation provides an overview of the WebSphere Commerce programming model and architectural framework.

Unit objectives

- Outline the runtime architecture
- Describe the development model
- Introduce the use of Struts
- Introduce commands and the command API

This presentation introduces you to the runtime architecture and development model of WebSphere Commerce. Because Struts is new to WebSphere Commerce, this unit introduces the basics of Struts and the use of Struts in WebSphere Commerce.

WebSphere Commerce runtime architecture



Before examining how the WebSphere Commerce Server functions, it is useful to look at the larger picture of the software components that relate to WebSphere Commerce. This diagram shows a simplified view of these software products.

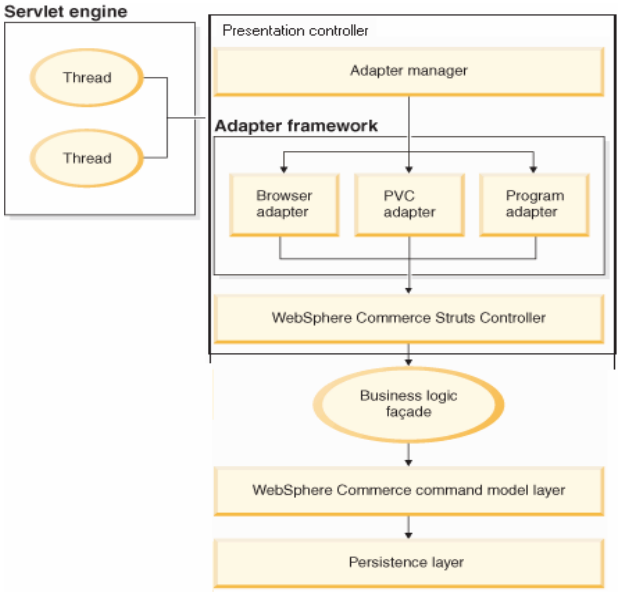
The Web server is the first point of contact for incoming HTTP requests for your e-commerce application. In order to interface efficiently with the WebSphere Application Server, it uses the WebSphere Application Server plug-in to manage the connections between the two components.

The WebSphere Commerce Server runs within the WebSphere Application Server, allowing it to take advantage of many of the features of the application server. The database server holds most of your application's data, including product and customer data. In general, extensions to your application are made by modifying or extending the code for the WebSphere Commerce Server. In addition, you might have a need to store data that falls outside of the realm of the WebSphere Commerce database schema within your database.

Developers use Rational® Application Developer to create and customize storefront assets, business logic and persistence logic. It is also used to test code and storefront assets during development.

The WebSphere Commerce development environment uses a development database. Developers can use their preferred database tools (including Rational Application Developer) to make database modifications.

WebSphere Commerce application framework



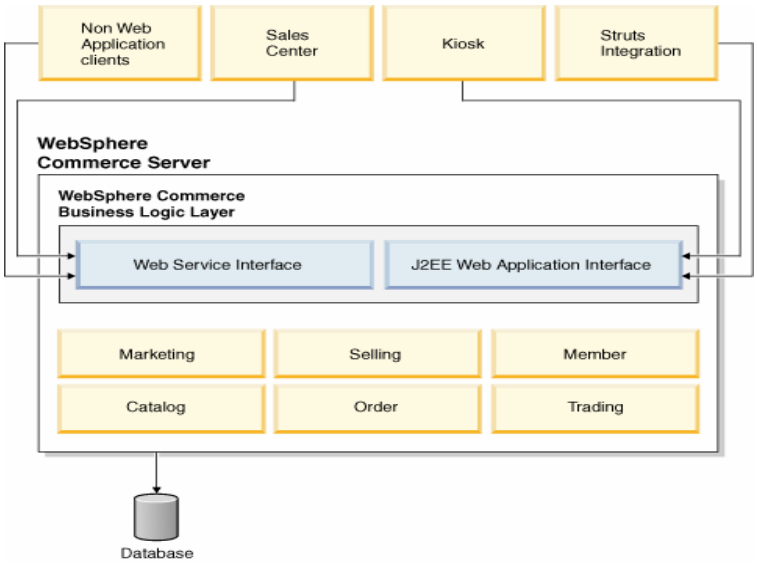
In WebSphere Commerce, the server runtime defines the framework for handling system and user requests, and performs the appropriate business logic to process the requests. The framework is built using an MVC design pattern and provides an environment that hosts business logic and handles persistence. It performs such tasks as transaction management and session management.

WebSphere Commerce and WebSphere Application Server support a variety of security mechanisms that can be used to protect access to data and other assets of the server. The access control framework in WebSphere Commerce prevents users from executing particular business logic. Not only does this access control framework provide a fine gain control on business logic, it also provides the flexibility to restrict access on what data your users are allowed to view and modify. Access control allows you to group commands by access groups, assign different customer commands to different owners, assign access to all owners, assign global site administrator access. By exploiting WebSphere Application Server global security, access to Web resources such as servlets, Java™ Server Pages (JSP) files, and Enterprise Java Beans (EJB) methods can be controlled for an additional layer of security.

The command framework, an architectural component of the WebSphere Commerce server runtime, provides the ability to invoke commands which represent different business processes in the system. The command framework defines Java interfaces and abstract implementations that business logic extends and implements. Also provided is a set of base classes that commands can extend to simplify implementation.

The interactions between WebSphere Commerce components are shown in this diagram and are discussed in more detail in the coming slides.

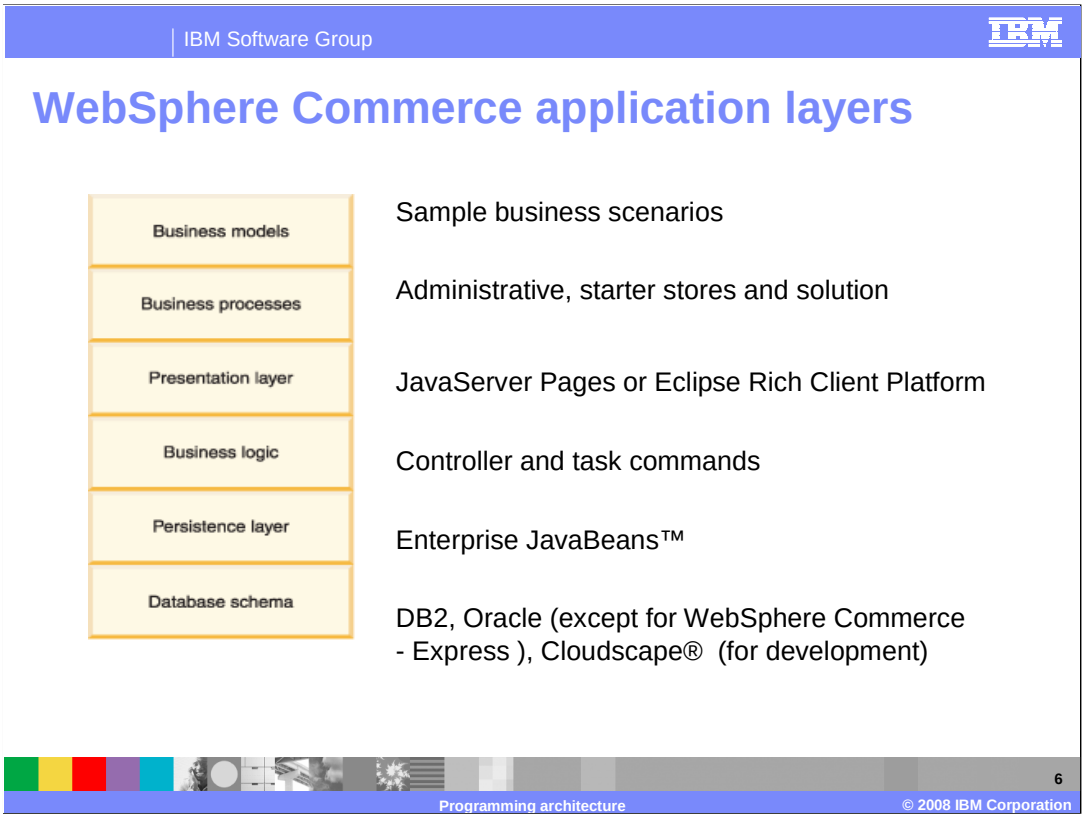
Runtime framework enhancements



New to this release is the further decoupling of the presentation tier from the business logic tier to better enable support for multiple sales channels. A sales channel is a method that a customer can use to purchase merchandise. Some examples of sales channels are in-store, from an online store, or from a call center. As shown in the diagram, requests can enter the WebSphere Commerce Server from different types of clients such as a rich client, kiosk or as a browser request.

In previous releases of WebSphere Commerce, the framework was designed for Web-based transactions. Now WebSphere Commerce is multi-channel-enabled, meaning that it can support transactions across various sales channels. The framework enhancements in this release support multiple presentation layers, responsible for displaying results, which decouple control logic from business logic.

Although there have been enhancements in the WebSphere Commerce runtime, task commands, controller commands, access beans, and entity beans continue to function as in previous releases.



Now that you have seen how the various software components related to WebSphere Commerce fit together, it is important to understand the application layers. The application layers help you to understand which parts are foundation layers and which parts you can modify. This diagram shows the various layers that compose the application architecture.

In WebSphere Commerce, a business model represents a sample business situation in which the WebSphere Commerce product might be used. The five business models provided by WebSphere Commerce are business to consumer, business to business, demand chain, supply chain and hosting.

Business processes represent the processes available in WebSphere Commerce divided by business model. The business processes are divided into three areas: administrative processes, starter stores and solution. Administrative processes are used to administer a site, a store, or an organization. Starter stores contain sample processes that can be followed by customers of the store. A solution describes the high-level view of how all the administrative processes and starter store processes fit within the framework of the overall business model.

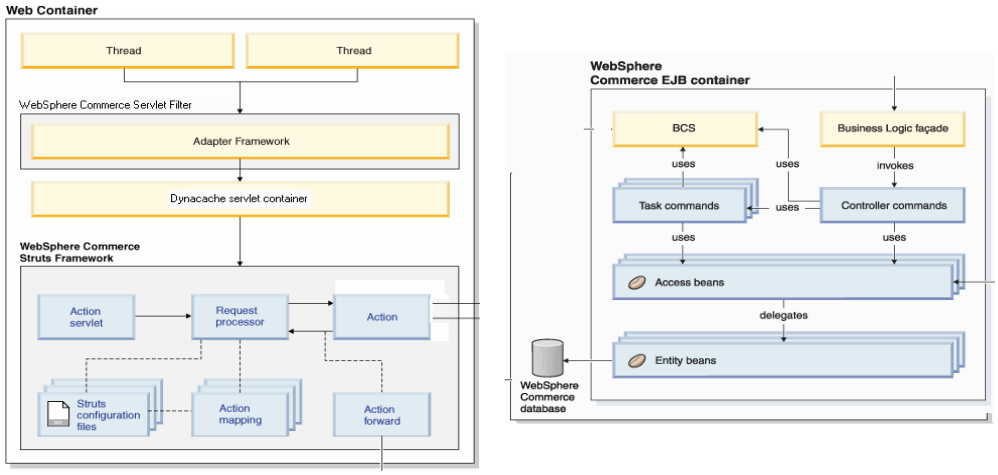
The presentation layer is responsible for displaying results. By default, there are two types of presentation layers supported: Web which is rendered using JSP files and rich client which is rendered with Eclipse views and editors.

The business logic layer is where business rules are implemented independent of the presentation layer. Business logic is implemented using the command pattern. Two types of commands are implemented: controller commands and task commands.

The persistence layer records the data and operations of the WebSphere Commerce system. The persistence layer represents entities within the WebSphere Commerce instance and encapsulates the data-centric logic required to extract or interpret information contained within the database.

The WebSphere Commerce database schema, which includes over 600 tables, is designed specifically for e-commerce applications and their data requirements. The database schema supports persistence requirements for the WebSphere Commerce subsystems.

WebSphere Commerce framework interaction



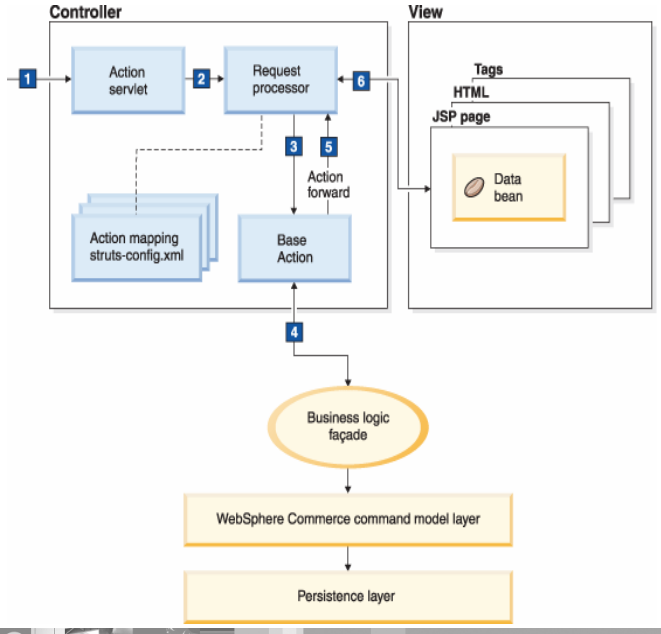
This diagram provides a summary of the interaction flow between components when forming a response to a request. It is specific to a Web application that uses Struts.

The process starts when the request is directed to the Presentation layer (Web container) in its own thread. The thread handling the request is dispatched to the WebSphere Commerce servlet filter. In turn the filter passes the request to the adapter framework. The adapter framework determines which adapter is capable of handling the request then returns the appropriate adapter. For example, if the request originated from an Internet browser, the adapter manager associates the request with the HTTP browser adapter. The adapter is passed back to the Dynacache servlet container. The filter regains control and passes the request to the servlet engine for processing. At this point one of two actions can occur. The request can be cached and the cached response can be returned or the Struts action can invoke the business logic façade by specifying the interface name of the business logic to invoke and the associated parameters. The business logic façade queries the command registry to determine the appropriate implementation for the store associated with the request. The business logic façade invokes the appropriate controller command.

Next, the controller command begins execution. The controller command can access the database using an access bean and its corresponding entity bean in addition to invoking one or more task commands. The task commands can also access the database.

The business logic façade returns a set of properties to the Struts action. One of the elements that is part of the properties is the key to the global forward that represents the response. The action looks up the global forward in the Struts configuration files. It resolves to the right one based on the store configuration. The action forward implementation that is selected is the appropriate one for the device of the request. The Struts request processor processes the action forward which invokes the appropriate JSP page. Within the JSP page, a data bean is required to retrieve dynamic information from the database. The data bean manager is used to activate the data bean. The access bean from which the data bean is extended accesses the database using its corresponding entity bean. Based on globalization information of the request, the data bean formats the data into a readable version.

WebSphere Commerce Struts framework



As of WebSphere Commerce V6.0, the WebSphere Commerce Web application has moved from a proprietary model-view-controller implementation to the Struts open source implementation. Struts is a well-documented framework for Java Enterprise Web application development, which has become an industry standard for deploying model-view-controller applications. Struts enforces best practices and design patterns, boasts a large developer community, and is supported by IBM development tools such as Rational Application Developer. Among key benefits of Struts are its support for dynamic action forms, form validation, versatile tag libraries, and Tiles. WebSphere Commerce has extended the base Struts configuration model to provide the traditional WebSphere Commerce function seen in previous releases.

This diagram expands on the processing of a web-based request through the WebSphere Commerce Struts framework.

Step 1: The Action servlet receives an HTTP request.

Step 2: The Action servlet routes the request to the module's request processor.

Step 3: The request processor passes the request, action form, and action mapping to the base action.

Step 4: Actions use action form data to invoke business logic operations on behalf of the client

Step 5: An ActionForward object is returned that indicates what view the controller should forward to.

Step 6: The request processor forwards to the appropriate view element when the action is completed.

The next slide describes the function of some key Struts components in more detail.

WebSphere Commerce Struts

- Action servlet
- Request processor
- Actions
- Action mappings
- Action forms
- Action forwards
- Struts configuration files

http://struts.apache.org

The action servlet is a key controller component. WebSphere Commerce has extended the base Struts ActionServlet class so all action servlets in the WebSphere Commerce Web application must extend ECActionServlet. The action servlet provides additional functionality, such as the ability to refresh the Struts configuration without restarting the Web application. Several WebSphere Commerce components require a refresh of the Struts configuration.

Having received an HTTP request from the action servlet, the request processor performs core request processing. The request processor uses the Struts configuration file to determine the action mapping for the request and, if necessary, locates or instantiates an action form for the request. The request processor then locates, or instantiates, the appropriate action for the request and passes it the request, action form and action mapping. When the action is complete, the request processor forwards to the appropriate view element.

Actions carry out the controller logic for the individual request. This includes authorization and session logic, locating and calling the appropriate business (model) logic, and returning an ActionForward object that indicates where the controller should forward to. BaseAction is the WebSphere Commerce extension of the Struts Action class. It supplies the functionality that is necessary to invoke WebSphere Commerce commands based on the action's configuration as provided by the corresponding action mapping. Any custom action that invokes WebSphere Commerce commands must extend this class.

Action mappings represent the information that the controller knows about the mapping of a particular request to an instance of a particular Action class. The request processor selects an appropriate action for each request based on the action mappings (action-mapping elements) defined in the Struts configuration file. ECActionMapping is the WebSphere Commerce extension of the Struts ActionMapping class that allows custom configuration information to be passed to actions of the WebSphere Commerce Web application. To specify a mapping for an instance of the BaseAction class or its descendants, you must use or extend the ECActionMapping class.

Action forms are a key view component that is used to transfer information between the view and the controller-model pair. An ActionForm object is a Java representation of HTML form data. As such, it encapsulates form data for easy population and retrieval, supports form data validation, and is reusable. Input data is transferred to ActionForm objects automatically by the Struts framework.

Action forwards, considered to be a part of the controller, represent the next Web resource, typically an HTML or JSP page, in the control flow of the application. Returned by an action, an ActionForward object indicates where the request processor should forward to. ECActionForward is the WebSphere Commerce extension of the Struts ActionForward class.

Both the action servlet and request processor rely on Struts configuration files for application-specific component information, allowing for a declarative, rather than programmatic, configuration of a Struts-based Web application. By default, WebSphere Commerce provides multiple Struts configuration files per module, one for definitions, and the other for customizations and migration. Using multiple Struts configuration files is a best practice, making parallel development easier and configuration less prone to errors and conflicts.

Web services in WebSphere Commerce

- Uses WebSphere Application Server Web service engine
  - Hides the complexity of SOAP
- Reuses existing WebSphere Commerce assets
  - Existing programming model
  - Existing integration technology
- Designed for customization
  - XML request message mapping
  - JSP response building
- Inbound/Outbound Web service implementations

WebSphere Commerce V6.0 also introduces a new Web services framework to better enable support for multiple sales channels such as the Eclipse-based Sales Center tool.

Web services are self-contained, self-describing, modular applications that can be published and invoked across the Web. Web services perform functions ranging from simple requests to complicated business processes. After a Web service is registered, other applications can discover it.

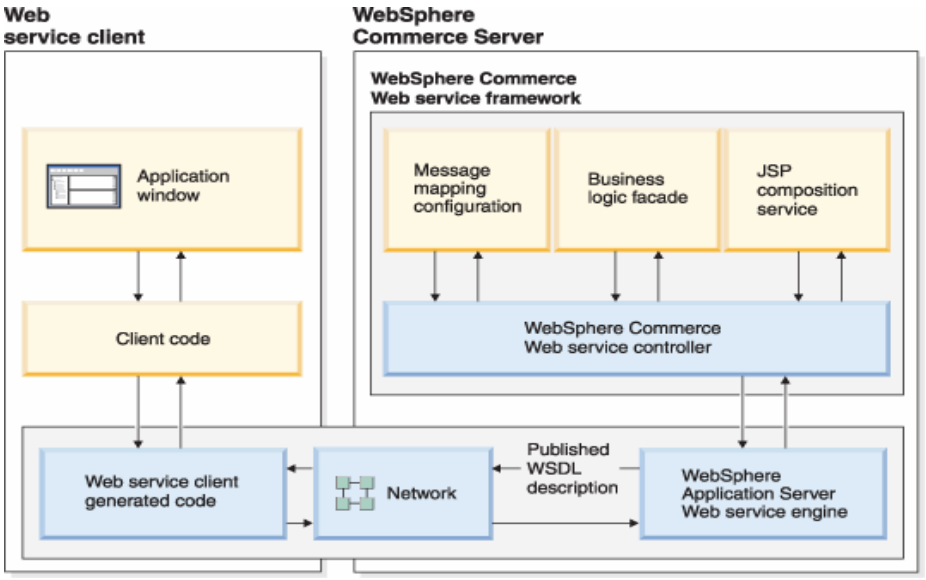
The new Web services framework uses the WebSphere Application Server Web service runtime infrastructure and Rational tools. It also promotes the use of industry-standard service definitions and allows top-down and bottom-up creation of Web services.

The existing command pattern is used to represent the business logic, and allow for existing URL-based controller commands to also be used by the Web services channel. Conversion from the Web service XML-based message format to name-value pairs used by existing commands is performed by a message mapper, the same integration technology that is used for parsing back-end integration legacy messages. The JSP page composition service is used to generate the Web service response, allowing dynamic caching to be optimized for either full-page or fragment caching.

Both the XML message mapper and the JSP used with the page composition service can be customized to process new Web service requests and create new Web service responses.

You can allow WebSphere Commerce to be the service provider by enabling its business functions as Web services that can be accessed by external systems. You can also allow WebSphere Commerce to be the service requester by enabling it to invoke Web services that are hosted by external systems.

Web services in WebSphere Commerce



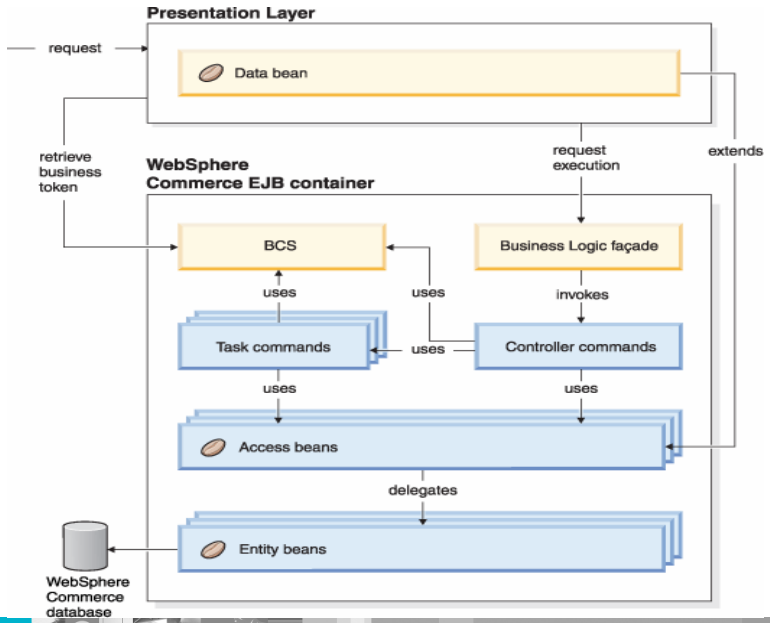
When you enable business operations in WebSphere Commerce as Web services that can be accessed by external systems, WebSphere Commerce becomes the service provider.

The deployment model for Web services is to have a central server with published WSDL that defines the services it supports. From there, clients, whether they are external systems, Web applications, or rich client applications, connect to the central server and invoke services that are defined by the publicly available WSDL.

The WebSphere Commerce Web service framework uses the approach of mapping the XML (SOAP body) request into the name-value-pair parameters passed to the service command that are invoked. In terms of overall Web service request handling, the WebSphere Application Server Web service engine is responsible for delegating the request to the WebSphere Commerce Web service framework. The framework is responsible for processing the request and generating the response. Processing the request consists of many steps. First the credentials associated with the request are resolved and the SOAP body is converted into name-value pairs. The name-value pairs are mapped to a controller command, the command is invoked, and the JSP composition service is used to create the XML response to the Web service request.

The JSP composition service allows you to easily modify the JSP template file in order to insert additional nodes in the response instead of having to provide Java code for this capability. JSP page resolution differs somewhat between the WebSphere Commerce Struts-based Web application and the WebSphere Commerce Web services. In a Struts-based Web application, the result of executing the business logic is a set of response properties, one of which is the name of the view to be used to determine which JSP page to invoke. The JSP page invoked generates an HTML response. In a Web services based application, the view for determining the response is specified in the mapping configuration. The view in the response properties is only used if no view is found in the mapping configuration. The view for a Web service response uses a JSP page to generate an XML response, which is returned as the response message to the Web service request. The Web service engine handles adding the necessary SOAP envelope to the response, and the Web service framework provides the capability of converting the XML JSP response to the SOAP element.

WebSphere Commerce functional architecture



This diagram is a recap of the WebSphere Commerce functional architecture seen earlier in this presentation. The components that make up the business logic layer are examined in more detail.

The business logic façade is a generic interface implemented as a stateless session bean. The Struts controller calls the business logic façade to invoke controller commands. A controller command performs business process logic such as OrderProcess. It invokes task commands to accomplish different units of work in the business process. By default, access control is enabled for controller commands. A task command is an autonomous task that accomplishes a specific unit of application logic such as check inventory. A task command typically works with other task commands to complete the processing of a controller command. By default, access control is not enabled for task commands.

Business context service, labeled BCS in the diagram, is new in version 6. It is a service that manages contextual information used by business components. The contexts include such information as globalization and entitlement.

Access beans are simple persistent objects with setters and getters. The access bean behaves like a Java bean and hides all the enterprise bean specific programming interfaces, like JNDI, home and remote interfaces from the clients. Rational Application Developer provides tool support to generate access beans from the schema. Entity beans are used in the persistence layer within WebSphere Commerce. The architecture is implemented according to the EJB component architecture. The EJB architecture defines two types of enterprise beans: entity beans and session beans.

Finally, the presentation layer displays the result of command execution. The presentation layer can use JSP pages, or other rendering technologies.

The remaining slides discuss the WebSphere Commerce command framework, business context service and access beans in more detail.

WebSphere Commerce commands

- Controller commands
- Task commands
- Data bean commands
- View commands

WebSphere Commerce commands are Java beans that contain the programming logic associated with handling a particular request. Commands perform a specific business process, such as adding a product to the shopping cart, processing an order, updating a customer's address book, or displaying a specific product page.

Controller commands encapsulate the logic related to a particular business process. In general, a controller command contains the control statements (for example, if, then, else) and invokes task commands to perform individual tasks in the business process. Upon completion, a controller command returns a view name. Based upon the view name, the store identifier, and the device type, the solution controller determines the appropriate implementation class for the view and then invokes it. Examples of controller commands include the OrderProcessCmd command for order processing and the LogonCmd that allows users to log on.

Task commands implement a specific unit of application logic. In general, a controller command and a set of task commands together implement the application logic for a URL request. Task commands are run in the same container as the controller command. Examples of task commands are DoPayment and DoLuhnCheck, which are used in the OrderProcess controller command.

Data bean commands are invoked by the data bean manager when a JSP page needs to instantiate a data bean. The primary function of a data bean command is to populate the fields of a data bean with data from a persistent object.

The purpose of view commands is to compose a view as a response to a client request. View commands are deprecated in this release of WebSphere Commerce. Since WebSphere Commerce is a Struts application, view commands have been replaced by global forwards. For compatibility with previous releases, view commands continue to work.

Command framework

- Command design pattern
- Command factory
- Command flow
- Command registration framework

Property	Entry for StoreA	Entry for StoreB
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce.mycommands.myUrl	com.ibm.commerce.mycommands.myUrl
CLASSNAME	com.ibm.commerce.mycommands.myUrlStoreAImpl	com.ibm.commerce.mycommands.myUrlStoreBImpl

Command beans follow a specific design pattern. Every command includes both an interface class and an implementation class. From a caller's perspective, the invocation logic involves setting input properties, invoking an execute method, and retrieving output properties. From the perspective of the command implementer, commands follow the WebSphere command framework, which implements the standard command design pattern allowing a level of indirection between the caller and the implementation. This level of indirection provides the ability to invoke an access control policy manager that determines if the caller is allowed to invoke the command. It also allows you to specify a different command implementation for different stores, based upon the store identifier.

In order to create new command objects, the caller of the command uses the command factory. It is based on the factory design pattern, which defers instantiation of an object away from the invoking class, to the factory class that understands which implementation class to instantiate. The command factory provides a way to determine the correct implementation class to use based upon the individual store.

Since controller commands encapsulate the logic for a business process, they frequently invoke individual task commands to perform specific units of work in the business process. Access beans are invoked when information in the database must be retrieved or updated. Either a task or controller command can invoke access beans. Requests then flow from access beans to entity beans that can read from, and write to, the WebSphere Commerce database.

WebSphere Commerce controller and task commands are registered with the command registration framework. The command registry is implemented by means of the CMDREG database table. The command registry provides a mechanism for mapping the command interface to its implementation class. Multiple implementations of an interface allow for command customization on a per store basis. If the command you are writing always uses the same implementation class, you do not necessarily have to register the command in the command registry. In this case, you can use the defaultCommandClassName attribute in the interface to specify the implementation class.

Command error handling

- Exception types
  - ECApplicationException – no retry
  - ECSystemException – retry allowed
- Customized code and exception handling
- Exception handling flow
- Exception logging
  - WebSphere Application Server's logs
  - WebSphere Commerce tracing facility

WebSphere Commerce uses a well-defined command error handling framework that is simple to use in customized code. By design, the framework handles errors in a manner that supports multicultural stores. A command can create one of two exception types.

An ECApplicationException is created if the error is related to user input and always fails. For example, when a user enters an incorrect parameter, an ECApplicationException is created. When this exception occurs, the solution controller does not retry the command, even if it is specified as a retriable command.

An ECSystemException is created if a runtime exception or a WebSphere Commerce configuration error is detected. Examples of this type of exception include create exceptions, remote exceptions, and other EJB exceptions. When this type of exception occurs, the solution controller retries the command if it is retriable and the exception was caused by either a database deadlock or database rollback.

When creating new commands, it is important to include appropriate exception handling. You can take advantage of the error handling and tracing provided in WebSphere Commerce, by specifying the required information when catching an exception.

The exception handling flow begins when the solution controller invokes a controller command. The command creates an exception that is caught by the solution controller. This can be either an ECApplicationException, or an ECSystemException. For a Web application, the struts framework determines the error global forward and invokes the specified error view. When invoking the view command, the solution controller composes a set of properties from the ECException object and sets it to the view. The ErrorDataBean passes the error parameters to the message helper object. The StoreErrorDataBean maps the error codes to messages.

Exception handling is tightly integrated with the logging system. When a system exception occurs, it is automatically logged. Exception messages can be written to both the WebSphere Application Server logs and the WebSphere Commerce trace files.

Command access control

- Users
- Actions
- Resources
- Relationships
- Command-level access control
- Resource-level access control

Access control in a WebSphere Commerce application is composed of four elements: users, actions, resources, and relationships.

Users are the people that use the system. For access control purposes, users must be grouped into relevant access groups. One common attribute that is used to determine membership of an access group is roles. Roles are assigned to users on a per organization basis. Some examples of access groups include registered customers, guest customers, or administrative groups like customer service representatives.

Actions are the activities that users can perform on the resource. For access control purposes, actions must also be grouped into relevant action groups. For example, a common action used in a store is a view. A view is invoked to display a store page to customers. The views used in your store must be declared as actions and assigned to an action group before they can be accessed.

Resources are the entities that are protected. For example, if the action is a view, the resource to be protected is the command that invoked the view, for example com.ibm.commerce.command.ViewCommand. For access control purposes, resources are grouped into resource groups.

Relationships are the relationship between a user and the resource requested. Access control policies might require that a relationship between a user and the source be satisfied. For example, users might only be allowed to display the orders that they have created.

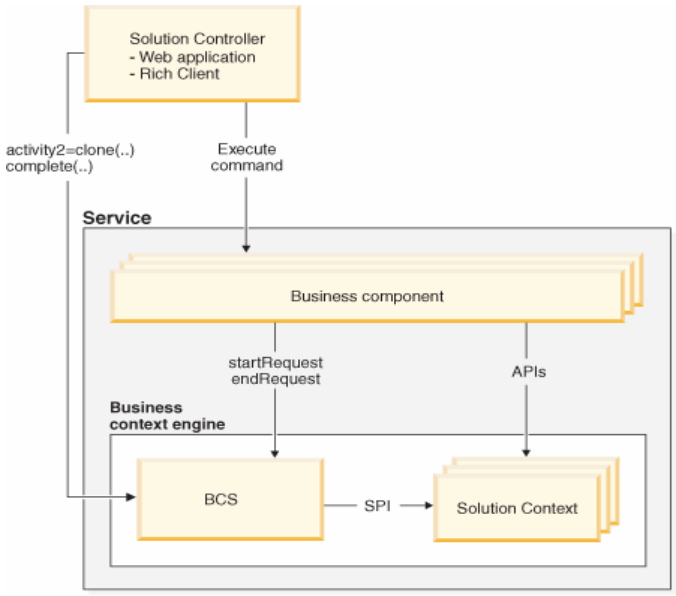
These four elements are combined to create access policies. Access control is based on a user's context, action invoked, resources used, and the relationships within the stores and site implementation. There are two types of policy-based access control in WebSphere Commerce, command-level and resource-level. In command-level access control, the "resource" is the command itself and the "action" is to invoke the command within the current store. The access control check determines if the caller is permitted to invoke the command within the current store. In resource-level access control, the "resource" is any protectable resource that the command or bean accesses and the "action" is the command itself.

In previous versions of WebSphere Commerce, runtime infrastructure was designed to serve customers from the Web channel. Since WebSphere Commerce introduces support for different client types, a new infrastructure is needed to provide contextual information to the business logic independent of channel. Business context services track user session information from different channels by using different types of business contexts. A complete list of business contexts is provided in a later slide. Examples include the BaseContext and EntitlementContext. The first contains the basic attributes that a request needs, such as store ID, caller ID, and the run-as ID. The second holds information about entitlement criteria, such as reduced prices for gold club membership. You can also define custom context information to extend WebSphere Commerce predefined session attributes.

Abstracting out user contextual information offers several benefits. Components can be created in a generic fashion with specific actions being triggered by the context information. Content and shopping experience can be tailored to the individual resulting in precisely targeted offers in addition to appropriate prices, entitlements and terms. Business policies can be easily enforced.

Multiple business context services can be logically grouped into business components. A component specializes in a function such as catalog management or tax calculation.

Business context services



This diagram shows the artifacts in the business context model. The solution controller is responsible for managing the activities of a user. In the WebSphere Commerce Struts framework the solution controller exists as part of the base action. In the Sales Center the solution controller exists as part of the rich client application. The client performs a limited set of actions while the solution controller provides the additional business context service process hooks.

An activity token is an identifier for an activity that spans multiple requests and transactions. An activity typically begins when a user logs on to the site and ends when a user logs off. The activity token is unique for each caller, run-as user, and store. The activity token groups the set of contexts effective during a particular client's request to the various business components. The client supplies this activity token on every request to indicate the experience that it expects from the business components. In most cases, the solution controller initiates a request to the business context service to request an activity token.

A business component group is a set of related services. A service provides a specialized function. It interacts with the business context service to signify the beginning and the end of a request such that any pre-processing and post-processing on business contexts can be done. Also, components can retrieve and update various business contexts during the execution of a request.

The business context engine provides contexts that dictate the behavioral characteristics of business components over the lifetime of the session. The business context engine consists of the business context service, labeled BCS in this diagram, and business contexts, shown here as the solution context.

The solution controller uses the business context service (BCS) to manage contexts required by business components. BCS is also an interface used by the components to obtain contexts they need. The contexts associated with an activity maintain their state for the lifetime of the activity. BCS also manages a cache of the business contexts associated with an activity within a request.

Business contexts establish an execution environment that affects the output of a business component for the equivalent input, based on the solution needs. Contexts are not directly invoked by clients of the business processes. Instead it is the business component that uses the services provided by the contexts during the execution of a request. The lifespan of a context starts with the creation of an activity and ends with the completion of the activity. Each business context implements two interfaces. The service provider interface, labeled SPI in the diagram, defines methods for business context service to indicate events such as the beginning of an activity, the start and end of a request. The context APIs are extended by each business context which defines methods for business logic to use to retrieve context specific information.

Business contexts are grouped together in business context sets. A business context set is a logical grouping of business contexts for a particular type of request. Different context sets can be configured for requests coming from different Web applications. For example, requests that are initiated from the Accelerator Web application need to include content and task contexts for authoring purposes, while requests from the store Web application do not have this requirement.

Business contexts

- BaseContext
- EntitlementContext
- GlobalizationContext
- ContentContext
- TaskContext
- AuditContext
- PreviewContext
- ExperimentContext

WebSphere Commerce predefines the contexts shown here.

BaseContext contains the basic attributes that an activity needs, such as store ID, caller ID, and the run-as ID.

EntitlementContext holds information about entitlement criteria, such as reduced prices for gold club membership.

GlobalizationContext helps components determine locale-specific information such as what language a message should be rendered in, or what currency should be used in the calculation of a price.

If Workspaces are enabled, ContentContext determines the content or business objects that can be displayed or edited based on workspace task group information and TaskContext determines which task an administrator is currently performing.

AuditContext is typically provided by vendor components. You might want to bridge the gap to the vendor interface instead of programming it directly. This context enables you to connect to a different vendor's implementation of the service in the future without the need to rewrite your component.

The preview context allows you to validate independent content without influencing other users. The context object associated with the preview operation represents the state information that is used when deciding the content to display. The preview context contains the preview date that is used to render the content to be displayed.

ExperimentContext is used to store the result of all active experiments for individual users, where the result is a system-generated number which determines the control or test element to be selected in the experiment. This information is persisted throughout the user session, so the same result is used in the same session without re-generation of the number.

Java beans in WebSphere Commerce

- Data beans
- Access beans
- Enterprise Java Beans
  - Entity beans
  - Session beans

There are three types of Java beans used in WebSphere Commerce: data beans, access beans and Enterprise Java Beans.

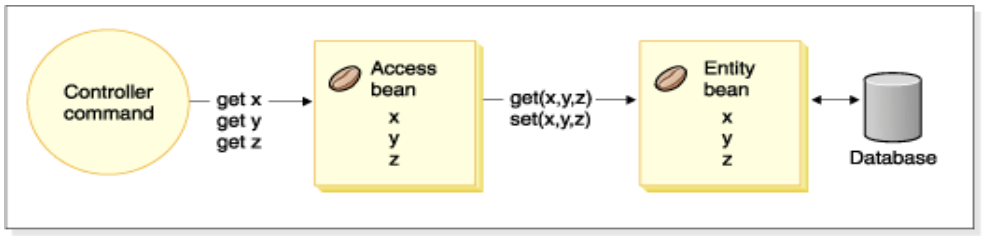
A data bean is a Java bean that is used within a JSP page to provide dynamic content. A data bean normally provides a simple representation of a WebSphere Commerce entity bean. The data bean encapsulates properties that can be retrieved from or set within the entity bean. As such, the data bean simplifies the task of incorporating dynamic data into JSP pages.

Access beans, which can be generated by Rational Application Developer, provide a simple way to access information from an entity bean. WebSphere Commerce commands interact with access beans rather than directly with the entity beans. Access beans are discussed more on the next slide.

The persistence layer within the WebSphere Commerce architecture is implemented according to the EJB component architecture. The EJB architecture defines two types of enterprise beans: entity beans and session beans. WebSphere Commerce uses mostly entity beans allowing developers to use the EJB tools provided in Rational Application Developer. A small number of stateless session beans are used to handle intensive database operations, such as performing a sum of all the rows in a particular column.

Access beans in WebSphere Commerce

- Behave like a Java bean
- Hide EJB interfaces (JNDI, Home, Remote)
- Cache the EJB Home object (expensive)
- Implement a copyHelper object



As illustrated in the diagram, WebSphere Commerce commands interact with access beans rather than directly with entity beans.

Using the access bean provides several advantages over working with entity beans directly. The first is a simpler programming interface. The access bean behaves like a Java bean and hides all the enterprise specific programming interfaces, like JNDI, home and remote interfaces from the clients. Another advantage is at run time the access bean caches the enterprise home object because look ups to the home object are expensive, in terms of time and resource usage. Also, the access bean implements a copyHelper object that reduces the number of calls to the enterprise bean when commands get and set enterprise bean attributes. Therefore, only a single call to the enterprise bean is required, when reading or writing multiple enterprise bean attributes.

Summary

- Runtime architecture
- Struts in WebSphere Commerce
- Web services in WebSphere Commerce
- Command framework
- Business context service
- Java beans in WebSphere Commerce

This presentation introduced you to many topics related to the runtime architecture and development model of WebSphere Commerce. The architecture overview was followed by an introduction to the use of Struts and Web services in WebSphere Commerce. Next was a summary of the command framework. The new business context service was introduced and the presentation concluded with a recap of Java beans in WebSphere Commerce.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_wcs60_ProgrammingArchitecture.ppt

This module is also available in PDF format at: ../wcs60_ProgrammingArchitecture.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

Cloudscape DB2 WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

EJB, Enterprise JavaBeans, Java, JavaServer, JSP, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.