



IBM Software Group

WebSphere® Commerce V6.0 Feature Pack 3

Component services



@business on demand.

© 2008 IBM Corporation
Updated April 29, 2008

This presentation describes the component services architecture and customization points.

Agenda

- Business component service
 - ▶ Review from feature pack 2
 - ▶ New functions in feature pack 3
 - Business logic layer
 - Data service layer



The agenda for this presentation is to discuss the business component service module, which controls communication between the business logic and data service layers. The functions introduced in feature pack 2 are reviewed. The new functions introduced in feature pack 3 - the business logic and data service layers - are then discussed.

Recap of Feature Pack 2

- A runtime supporting the OAGIS message format
 - ▶ Standardize approach on how to represent request and response
- Developing protocol independent service modules
 - ▶ Consists of a set of request and response Business Object Documents
 - ▶ Deployed as an Enterprise Java™ Bean
 - ▶ Command processing pattern
- Design pattern toolkit
 - ▶ Simplify developing a new service module

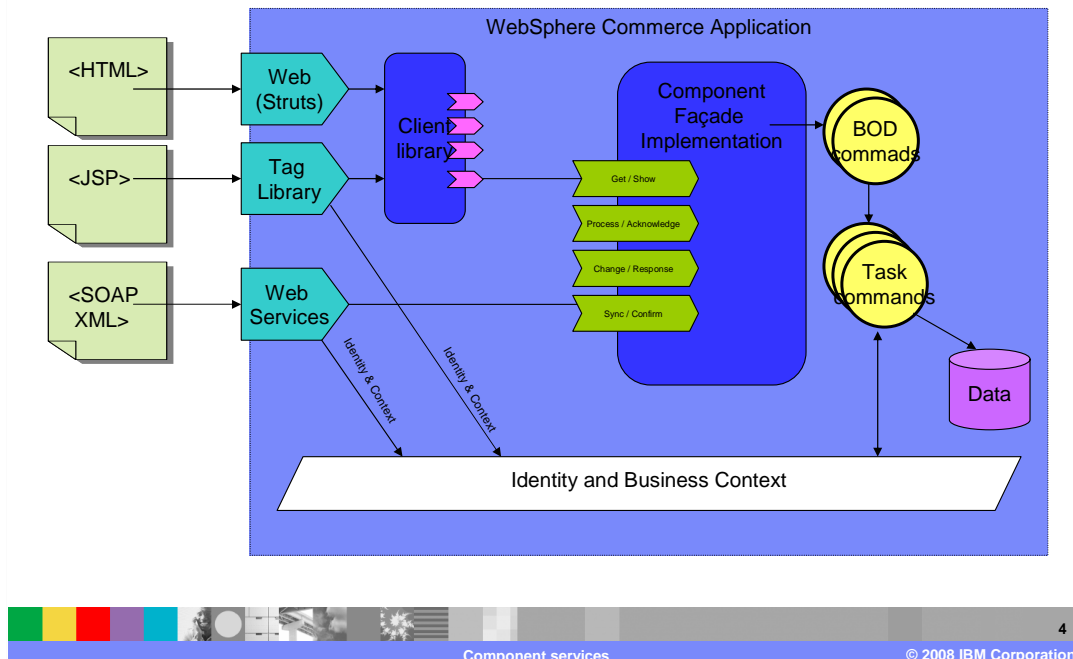


In Feature Pack 2, focus was placed on how OAGIS can be used to represent services to retrieve data, update data or run business processes. You should already be familiar with the OAGIS message architecture. If you are not, you should first view the presentation on OAGIS message architecture.

Feature Pack 2 also continued the transition to a protocol independent SOA architecture. Services use request and response business object documents to transfer information about WebSphere Commerce objects between different clients, such as Web and portal, and the server. On the WebSphere Commerce Server side, service requests are mapped back to existing Commands and Enterprise Java Beans, allowing business logic to be reused.

The design pattern toolkit was introduced to simplify developing new service modules by providing a pattern to generate much of the necessary configuration.

WebSphere Commerce service binding



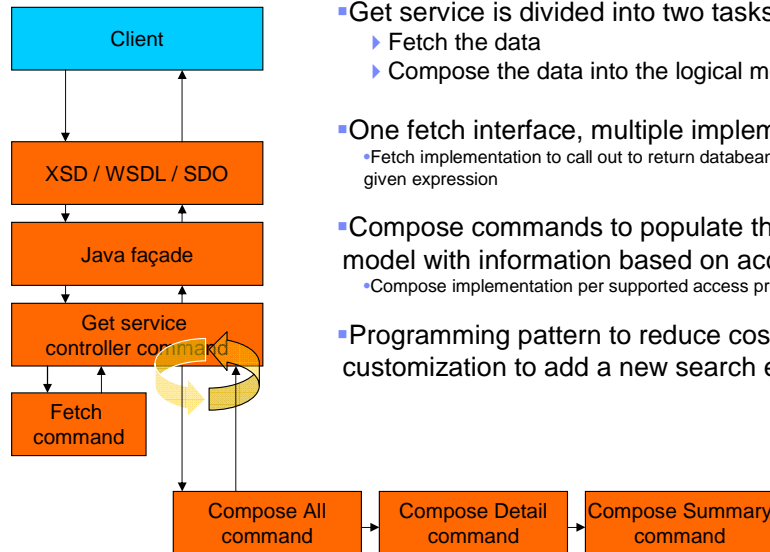
This diagram recaps service binding in Feature Pack 2.

The service binding resides between the client library and the services. It provides the transport mechanism to pass data, using SDOs, between the client and the service. There are two types of binding between the client library and the component facade implementation, Enterprise Java Beans and Web services.

The local enterprise bean connects the client to the component facade implementation in the local JVM. When the client and component facade implementation are deployed within the same application, such as for a Web interface, the client should communicate with the component facade implementation through a local EJB call. The client uses the component facade's local enterprise bean to invoke the method that matches the intended service.

The Web service connects the client to the component facade implementation remotely. When the client and component facade implementation are deployed in separate applications, such as with a portal interface, the client sends the service request using Web services as the transport layer. Because the client and component facade implementation are distinct applications, the Web service security is used for authorization.

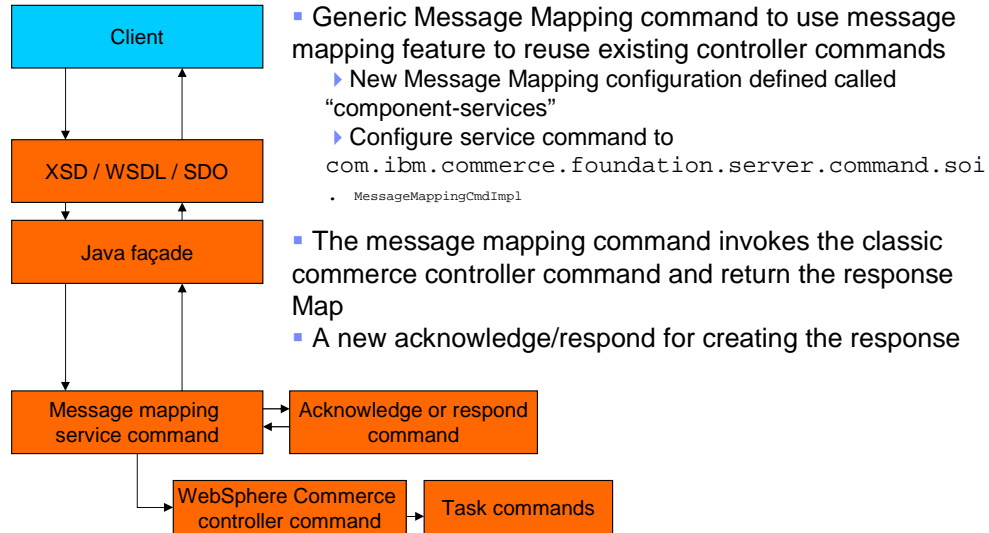
Get request design pattern using existing data beans



- Get service is divided into two tasks
 - Fetch the data
 - Compose the data into the logical model
- One fetch interface, multiple implementations
 - Fetch implementation to call out to return databeans based on the given expression
- Compose commands to populate the logical model with information based on access profile
 - Compose implementation per supported access profile.
- Programming pattern to reduce cost of customization to add a new search expression

Feature Pack 2 introduced a Get request pattern that allows existing WebSphere Commerce artifacts such as data beans to be accessed through a service. Details on this pattern can be found in the Feature Pack 2 presentation on Business Component Services.

Process or Change request using existing controller commands



Feature Pack 2 also exposed existing controller commands as services by introducing a message mapping command that can delegate structured service requests to the controller commands. The BOD request is converted into a set of name-value pairs that the controller command can process. Further information on the message mapping command can also be found in the Feature Pack 2 presentation on Business Component Services.

Feature Pack 3 improvements

- **Abstract business object document commands**
 - ▶ Commands to support the OAGIS processing model
 - ▶ Designed to support more complex requests
- **Data service layer**
 - ▶ Isolates the business logic from dealing with the persistence layer
 - ▶ Business object mediator
 - ▶ Persistence layer service
- **Developer tools**
 - ▶ Improvements on the design pattern toolkit pattern
 - ▶ WebSphere Commerce Developer plug-in to help with configuration and implementation assets

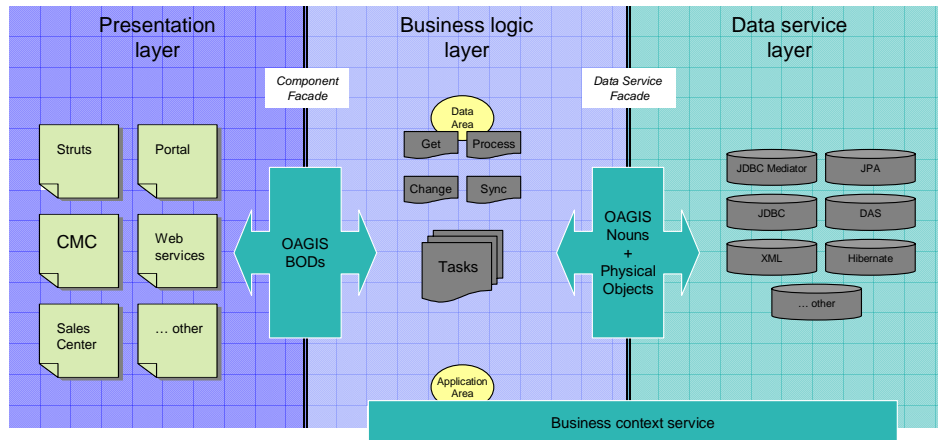


Introduced in Feature Pack 3, abstract business object document (BOD) commands give you two advantages over the message mapping command in Feature Pack 2. First, SDOs can be used directly by the business logic rather than mapping to name-value pairs as is necessary for controller commands. Second, complex requests representing multiple changes can be sent in a single BOD request. In Feature Pack 2 BOD requests were limited to a single action.

The new Data Service Layer encapsulates all interaction with the persistence layer. This decreases the cost of changing persistence technologies because the business logic no longer has ties to the persistence implementation. The Business Object Mediator and Persistence Service assist in the communication between the business logic and persistence layers. They are discussed in more detail in the coming slides.

Developer tools have been extended in Feature Pack 3. The improvements are an enhanced Design Pattern Toolkit and a new plug-in for WebSphere Commerce Developer. The plug-in assists in generating some of the artifacts required for the data service layer.

Architectural layers



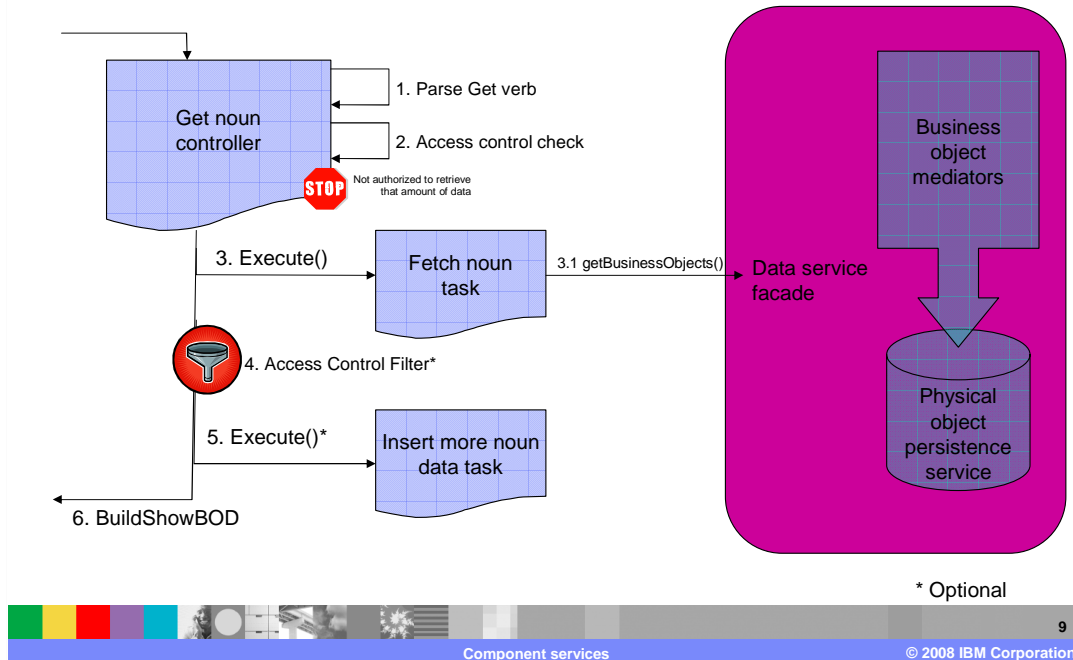
This slide shows a summary of the architectural layers in Feature Pack 3. This architecture removes implementation dependencies between the presentation layer, business logic layer and persistence (or data service) layer.

The presentation layer interacts with the business logic through the OAGIS defined services and does not contain any business logic directly. Retrieving business data or executing any business logic must be done through the OAGIS defined services of a service module.

The business logic layer contains the business components that provide OAGIS services to return data or invoke business processes. The Business Object Document processing patterns for OAGIS requests use the data service layer to accept structured objects called service data objects (SDOs) and perform the mapping between these logical structured objects and how they are persisted. The business logic never needs to interact with the technology used to persist the data.

The data service layer accepts SDOs, which are transformed (mediated) into objects called Physical SDOs before being passed to the persistence implementation to perform the data retrieval or updates. All persistence-specific assets such as SQL queries are isolated within the data service layer. SQL is stored in a query template file for easy maintenance.

Advanced Get processing pattern



The Business Object Document Get processing pattern describes the design pattern used to perform searching for and retrieving of data. The flow of the Get processing pattern is as follows:

In step 1, the GetNoun controller command parses the expression from the Get request into a search expression. In step 2, an access control check is performed to ensure the current user is allowed to use the access profile. The GetNoun command then delegates to the fetch command in step 3 to retrieve the list of nouns that match the search expression. The GetNoun command uses the XPath selector approach to choose the appropriate fetch command implementation to retrieve the data. Although the default implementation is used in most cases, the extension allows for additional business logic to be added for particular search expressions. The default fetch implementation uses a persistence object called the Business Object Mediator, passing the search expression extracted from the Get verb. This search expression includes the XPath, access profile and paging information. The Business Object Mediator in turn uses the logical mediators and data service to retrieve the data and convert it to its logical representation. The result is paging information and logical SDOs matching the expression.

In step 4, the list of nouns returned by the fetch command is filtered to ensure the current user only views those nouns that they are allowed to see. This additional filtering is optional.

Step 5, also optional, is for access profiles that need additional information based on business logic or from an external system. The GetNoun controller can instantiate an instance of an InsertMoreNounData task command for a particular access profile to populate more data. The InsertMoreNounData command should only be used when the information cannot be retrieved from the persistence layer.

Finally, in step 6, the nouns and the show verb are packaged in the ShowNoun response and returned.

Access control policy for an access profile

```

<Policies>
  <!-- Define the action for the Access Profile: BOD.<AccessProfile> -->
  <Action Name="GetCatalogEntry.Details" CommandName="GetCatalogEntry.IBM_Details" />
  <!-- The resource category for all Access Profiles -->
  <ResourceCategory
    Name="com.ibm.commerce.foundation.server.authorization.policymanager.AccessProfileResourceCategory"
    ResourceBeanClass="com.ibm.commerce.foundation.server.authorization.policymanager.AccessProfileProtectableProxy" />
  <!-- Include the access profile as part of the action group -->
  <ActionGroup Name="Catalog-CatalogEntry-AllUsers-AccessProfileActionGroup"
    OwnerID="RootOrganization">
    <ActionGroupAction Name="GetCatalogEntry.Details" />
  </ActionGroup>
  <!-- The access profile resource group -->
  <ResourceGroup Name="AccessProfileResourceGroup" OwnerID="RootOrganization">
    <ResourceGroupResource
      Name="com.ibm.commerce.foundation.server.authorization.policymanager.AccessProfileResourceCategory"/>
  </ResourceGroup>
  <!-- Define a policy for the action group -->
  <Policy Name="Catalog-CatalogEntry-AllUsers-AccessProfilePolicy"
    OwnerID="RootOrganization"
    UserGroups="AllUsers"
    ActionGroupName="Catalog-CatalogEntry-AllUsers-AccessProfileActionGroup"
    ResourceGroupName="AccessProfileResourceGroup"
    PolicyType="groupableStandard" />
  <!-- Add the policy to a policy group -->
  <PolicyGroup Name="ManagementAndAdministrationPolicyGroup" OwnerID="RootOrganization">
    <PolicyGroupPolicy
      Name="Catalog-CatalogEntry-AllUsers-AccessProfilePolicy"
      PolicyOwnerID="RootOrganization" />
  </PolicyGroup>
</Policies>

```

Should you have access to this view of the Noun?



This slide shows an example of an access control policy that allows the WebSphere Commerce server to answer the question “should you have access to this view of the noun?” when processing a Get request.

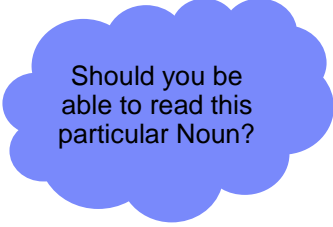
The policy has six distinct parts: action, resource category, action group, resource group, policy and policy group. The action command name is based on the name of the BOD and the access profile. Next, the resource category is constant across all policies. If you are creating your own policy you can just copy and paste this portion. The `AccessProfileProtectableProxy` interface is part of a bridge layer that allows the generated SDOs to make use of the existing WebSphere Commerce access control engine. The action group allows you to specify multiple actions that are controlled by the same policy. Only one action is shown in this example. Resource group performs a similar function for resource categories as action group does for actions. However, since there is only one resource category for all profiles, the resource group definition is also constant across all profiles. Next is the policy, which brings all the parts together to define who is allowed to do what. This policy specifies that all users are allowed to perform the actions listed in the action group `Catalog-CatalogEntry-AllUsers-AccessProfileActionGroup` defined above. Policies can define broad access, such as this example for all users, or can be used to create very specific access profiles such as a catalog managers group that can access catalog authoring commands. Finally, the policy is registered in the global policy group for WebSphere Commerce.

Access control policy to read a noun

```

<Policies>
  <!-- The command to read the noun is always display -->
  <Action Name="DisplayResourceAction" CommandName="Display" />
  <!-- Registering the resource category for the Protectable object that represents the noun -->
  <ResourceCategory
    Name="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryTypeResourceCategory"
    ResourceBeanClass="com.ibm.commerce.catalog.facade.server.authorization.CatalogEntryTypeProtectableProxy" />
  <!-- Defining an action group to read the noun -->
  <ActionGroup Name="Catalog-CatalogEntry-AllUsers-ActionGroup"
    OwnerID="RootOrganization">
    <ActionGroupAction Name="DisplayResourceAction" />
  </ActionGroup>
  <!-- Defining a resource group that contains the resource category of the noun -->
  <ResourceGroup Name="Catalog-CatalogEntry-ResourceGroup" OwnerID="RootOrganization">
    <ResourceGroupResource Name="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryTypeResourceCategory"/>
  </ResourceGroup>
  <!-- Defining the policy that controls who can read the noun -->
  <Policy Name="Catalog-CatalogEntry-AllUsers-Policy"
    OwnerID="RootOrganization" UserGroup="AllUsers"
    ActionGroupName="Catalog-CatalogEntry-AllUsers-ActionGroup"
    ResourceGroupName="Catalog-CatalogEntry-ResourceGroup"
    PolicyType="groupableStandard" />
  <!-- Registering the policy -->
  <PolicyGroup Name="ManagementAndAdministrationPolicyGroup"
    OwnerID="RootOrganization">
    <PolicyGroupPolicy Name="Catalog-CatalogEntry-AllUsers-Policy"
      PolicyOwnerID="RootOrganization" />
  </PolicyGroup>
</Policies>

```



Should you be able to read this particular Noun?

11

Component services

© 2008 IBM Corporation

You can also define an access control policy that answers the question “should you be able to read this particular noun?”.

For each noun that is returned by the FetchNounCmd, a Display access control check is performed. This is similar to previous versions of WebSphere Commerce where the Display command was used for data beans. The action here is hard coded to be Display. The resource category is the protectable proxy object of the returned noun. As in the previous example, the protectable proxy allows the generated SDO objects to use the WebSphere Commerce access control engine. The action group and resource group are defined followed by the policy. In this example, the policy says that all users are allowed to perform the actions contained in the Catalog-CatalogEntry-AllUsers-ActionGroup. In this case, all users can display any catalog entry. The policy can also be used to enforce relationships. For example, a creator relationship can be defined such that only the creator of an order is allowed to view the order. As a final step, the policy is registered in a policy group.

Advanced Get pattern customization points

- New XPath expression
 - ▶ Expression represents an SQL statement
 - ▶ Business logic computes the result
- Access profile update
 - ▶ Return more data from the database
 - ▶ Return more data from an external system
- New access profile
 - ▶ Return a logical model that has a different amount of data populated



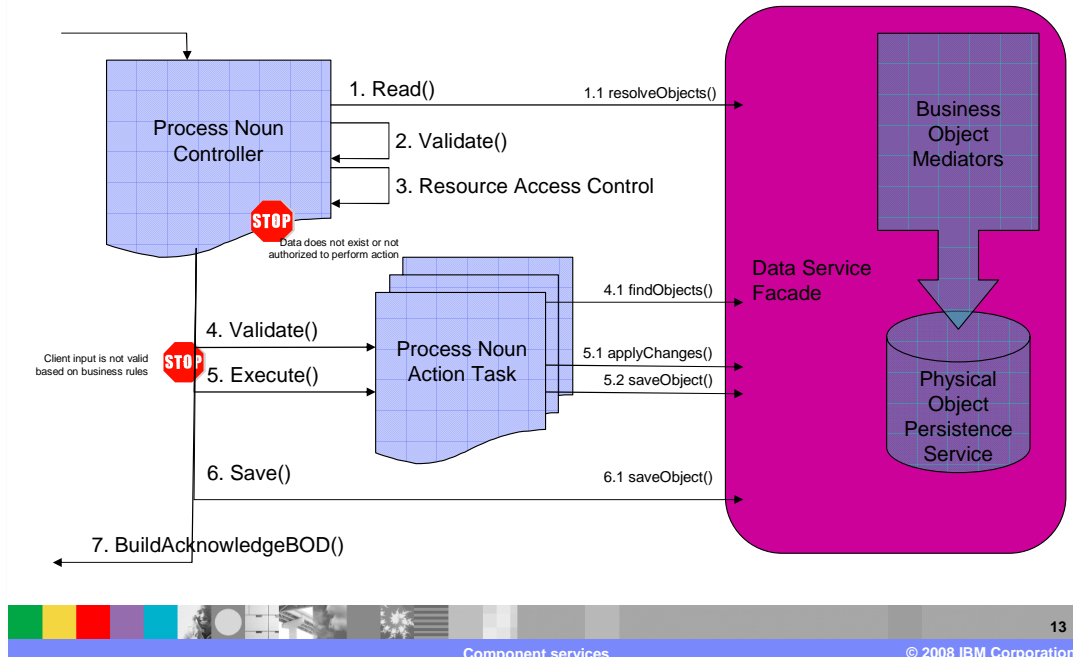
The new Get pattern allows for customization at several points.

The first type of customization is to add a new XPath expression that can represent either an SQL statement or a computation that should be performed by the business logic. To add an XPath expression that represents an SQL statement, you need to update the data service layer template to register the XPath to SQL mapping. You can introduce a new XPath expression that is computed by the business logic, but you should do so only if really needed. To add this type of expression, you need to implement a new FetchNoun task command to handle retrieving the data and update the command configuration to associate the XPath key with your FetchNoun task command.

The second type of customization is updating the access profile. You need to update the access profile if you want to return more data from the database or from an external system for an existing Get request. When the additional data is coming from the database, you need to update the data service layer template to return more data for the access profile. You also need to update the business object mediator to handle the association of the new data in the logical model. To support returning data from an external system you need to create an InsertMoreNounData command to access the external system, retrieve the data and add it in the logical model. The command configuration then needs to be updated to associate this access profile with the InsertMoreNounData command.

The final type of customization is to create a new access profile. This is required if you need to return a logical model that has a different amount of data populated. The first step in this customization is to update the access control policies to register who can use the access profile. Next, you need to update the data service layer configuration to register the data contained in the new access profile. An optional third step is to create and register an InsertMoreNounData command as just described.

Process processing pattern



The Business Object Document Process pattern is similar to the Change design pattern, except that the Process pattern includes a Process controller. This Process controller reads common data across the actions within the request, instantiates the task command implementations for those actions, and runs them. This pattern is a simplified version of the pattern required for processing the Change request. The key difference between Process and Change is the rule that the Process action must act upon the entire noun. The information within that noun controls the actions of the business logic.

Because the Feature Pack 2 implementation mapped an action to a controller command, the Process pattern was limited to one action per request. For the BOD Process pattern, the restriction of one actionCode per request is lifted.

The flow of the BOD Process processing pattern is shown in this diagram.

Step 1. The Process noun controller command breaks down the BOD and calls the `read()` to resolve the root object of the nouns to change.

Step 2. The `validate()` method is called to perform any common validation required.

Step 3. An access control check is performed to ensure the current user has permission to perform the action on the specified noun.

Step 4. The Process noun action task commands are instantiated and for each command, the `validate` method is called to report potential errors that can occur during processing. The Process noun action command reads any information required and validates whether the input is valid for the operation.

Step 5. The Process noun action task commands are invoked to apply the changes and save any changes made to data objects retrieved in the current instance of the command.

Step 6. The root object retrieved in step 1 is saved.

Step 7. The response is created and returned.

Access control policy for process actions

```

<Policies>
  <!-- Register the possible process actions as CommandName that can be performed on the Noun -->
  <Action Name="AddResourceAction" CommandName="Add" />
  <Action Name="DeleteResourceAction" CommandName="Delete" />
  <Action Name="CreateResourceAction" CommandName="Create" />
  <!-- Registering the resource category for the Protectable object that represents the noun -->
  <ResourceCategory Name="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryTypeResourceCategory"
    ResourceBeanClass="com.ibm.commerce.catalog.facade.server.authorization.CatalogEntryTypeProtectableProxy" />
  <!-- Defining an action group that includes the process actions -->
  <ActionGroup Name="Catalog-CatalogEntry-CatalogEntryManagers-ActionGroup" OwnerID="RootOrganization">
    <ActionGroupAction Name="AddResourceAction" />
    <ActionGroupAction Name="DeleteResourceAction" />
    <ActionGroupAction Name="CreateResourceAction" />
  </ActionGroup>
  <!-- Defining a resource group that contains the resource category of the noun -->
  <ResourceGroup Name="Catalog-CatalogEntry-ResourceGroup"
    OwnerID="RootOrganization">
    <ResourceGroupResource Name="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryTypeResourceCategory" />
  </ResourceGroup>
  <!-- Defining the policy that controls who can run the action group that contains the actions -->
  <Policy Name="Catalog-CatalogEntry-CatalogEntryManagers-Policy"
    OwnerID="RootOrganization" UserGroup="CatalogEntryManagersForOrg"
    ActionGroupName="Catalog-CatalogEntry-CatalogEntryManagers-ActionGroup"
    ResourceGroupName="Catalog-CatalogEntry-ResourceGroup"
    PolicyType="groupableTemplate" />
  <!-- Registering the policy -->
  <PolicyGroup Name="ManagementAndAdministrationPolicyGroup" OwnerID="RootOrganization">
    <PolicyGroupPolicy Name="Catalog-CatalogEntry-CatalogEntryManagers-Policy"
      PolicyOwnerID="RootOrganization" />
  </PolicyGroup>
</Policies>

```

Should you be able perform the Process action on the Noun?

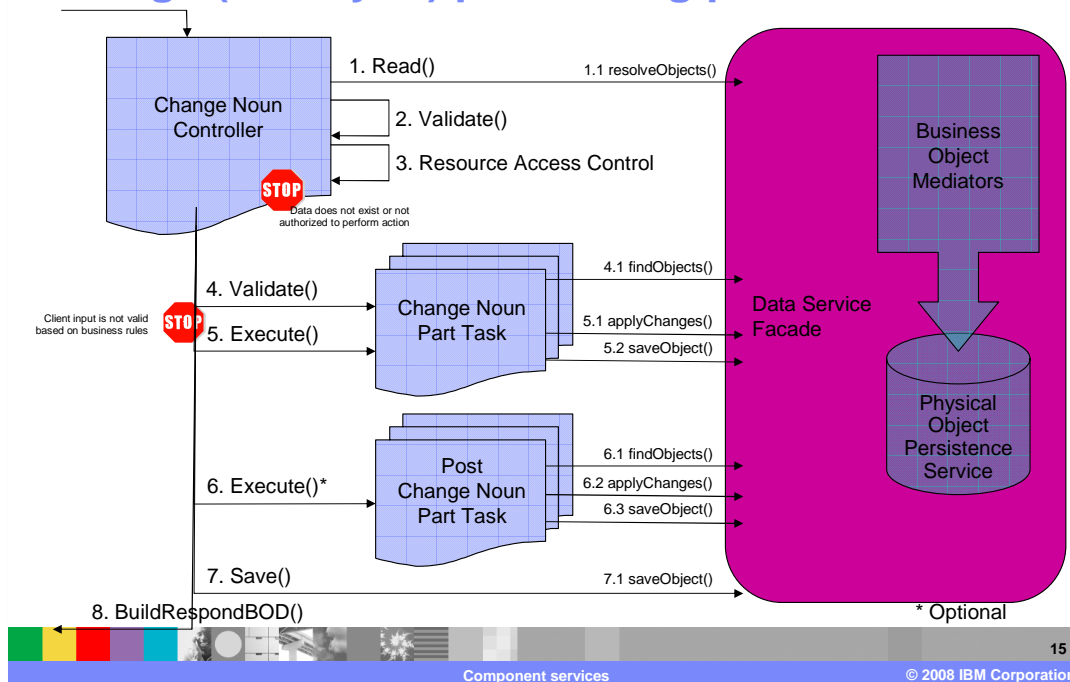
14

Component services

© 2008 IBM Corporation

Here is an example of an access control policy for Process services, which determines whether the current user has permission to perform the action on the noun. It is very similar to the access control policy for Get requests, the main difference is in the action definition. In the Get access control policy, the action is the Display command. For Process requests, the action is the action code found in the Process verb in the BOD. The resource category is the same Protectable proxy object used in the Get policy. This example also shows multiple actions being registered in the action group. The policy defines who is allowed to perform the actions. In this example, catalog entry managers are allowed to perform the add, delete and create actions on catalog entries.

Change (and Sync) processing pattern



The Business Object Document Change processing pattern is used to add, change, or delete a business object. The BOD Sync processing pattern is used by systems that contain master data records to push out notifications when their data has changed.

In step 1, the Change noun controller command breaks down the BOD and calls the read method to resolve the root objects of the nouns to change. The controller fetches the required data for the request up-front and passes this data to the other task commands so that the command uses and updates common data elements. In step 2, the validate method is called to perform any common validation that is required. This is followed in step 3 by an access control check to ensure the current user has permission to change the specified noun.

Once the access control check is complete, the request is split into smaller Change-noun-part task commands that act upon parts of the noun instead of the entire noun. In step 4, these commands are instantiated, and for each command, the validate method is called to report potential errors that can occur during processing. In step 5, the change noun part tasks are invoked to deal with a specific part of the noun. These tasks are passed the action to perform, the part of the noun that has changed and the original request noun along with the data that was retrieved by the controller. The controller can have many instances of these task commands which act upon the many noun parts that are being changed as part of the message.

Step 6 is optional. The post change noun part command is instantiated and run if additional business logic processing is required because of a change of that part of the noun – for example, auditing or logging.

Once all the tasks have been run, the data read up-front is persisted in step 7 so the changes are committed in the database. Finally, the response is created and returned in step 8.

Access control policy for change and sync actions

```

<Policies>
  <!-- Register the Change action as CommandName that can be performed on the Noun -->
  <Action Name="ChangeResourceAction" CommandName="Change" />
  <!-- Registering the resource category for the Protectable object that represents the noun -->
  <ResourceCategory
    Name="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryTypeResourceCategory"
    ResourceBeanClass="com.ibm.commerce.catalog.facade.server.authorization.CatalogEntryTypeProtectableProxy" />
  <!-- Defining an action group that includes the change action -->
  <ActionGroup
    Name="Catalog-CatalogEntry-CatalogEntryManagers-ActionGroup"
    OwnerID="RootOrganization"
    <ActionGroupAction Name="ChangeResourceAction"/>
  </ActionGroup>
  <!-- Defining a resource group that contains the resource category of the noun -->
  <ResourceGroup Name="Catalog-CatalogEntry-ResourceGroup"
    OwnerID="RootOrganization"
    <ResourceGroupResource Name="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryTypeResourceCategory" />
  </ResourceGroup>
  <!-- Defining the policy that controls who can run the action group that contains the pro
  <Policy Name="Catalog-CatalogEntry-CatalogEntryManagers-Policy"
    OwnerID="RootOrganization" UserGroups="CatalogEntryManagersForOrg"
    ActionGroupName="Catalog-CatalogEntry-CatalogEntryManagers-ActionGroup"
    ResourceGroupName="Catalog-CatalogEntry-ResourceGroup"
    PolicyType="groupableTemplate" />
  <!-- Registering the policy -->
  <PolicyGroup Name="ManagementAndAdministrationPolicyGroup"
    OwnerID="RootOrganization"
    <PolicyGroupPolicyName="Catalog-CatalogEntry-CatalogEntryManagers-Policy"
      PolicyOwnerID="RootOrganization" />
  </PolicyGroup>
</Policies>

```

Should you be able to change the Noun?



The access control policies for Change and Sync services determine whether the current user under the current context can perform the change actions on the specified noun. This policy is very similar to the Process access control policy. The only difference is there is a single action, Change, specified in this policy.

The default access control policies provided with WebSphere Commerce support access checking at the noun level. It is possible to extend the default policies to provide access control at the noun changeable part level if you need that in your application. For example, the default WebSphere Commerce policy checks whether the current user can change a catalog entry. You can extend the policy to check whether the current user can change the description or price for a catalog entry.

Data service layer

Key Features

- Layer of abstraction for data access
 - ▶ Abstract object-relational mapping frameworks (which transform database tables to Java)
 - ▶ Transform between physical SDOs and logical SDOs (OAGIS nouns)
- Create, read, update, and delete operations on both physical and logical data
- XPath:
 - ▶ Convert extended XPath query (logical schema) to SQL (physical schema)
 - ▶ Generate SQL statements from Query Template (business context aware)
 - ▶ Access profile



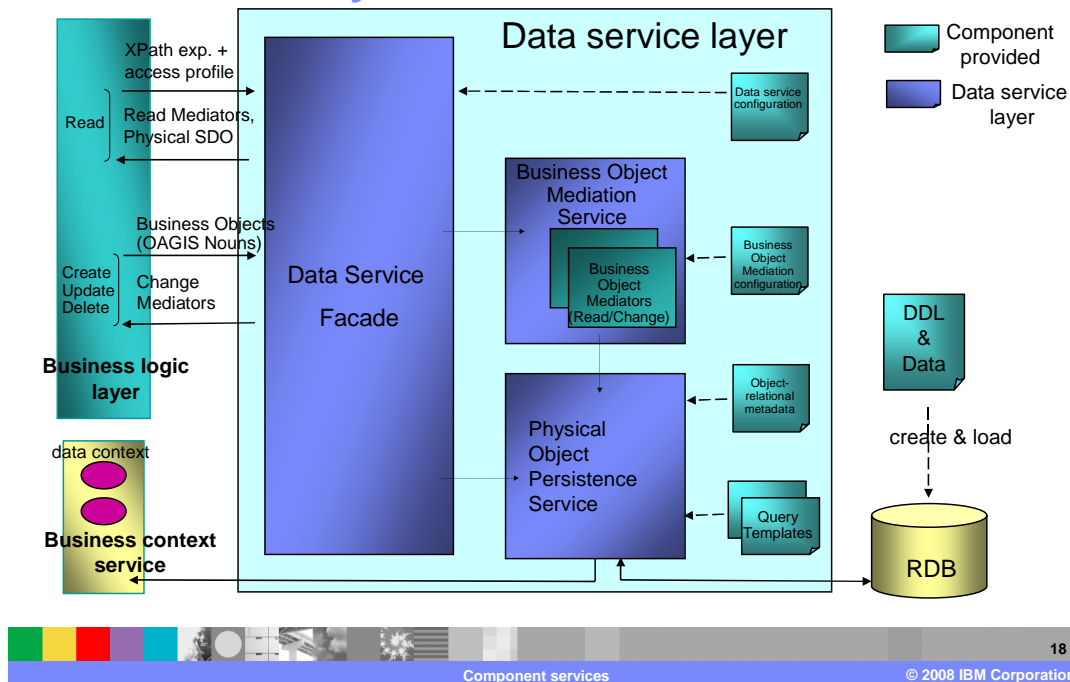
The purpose of the data service layer is to provide a neutral interface for accessing data, independent of the object-relational mapping framework. Examples of such frameworks are EJB, Data Access Service (DAS) from WebSphere Application Server, or Java Persistence API. In its turn, the abstracted mapping framework is used to transform the data retrieved from the database into a collection of Java objects. Internally, these objects are implemented by the data service layer as physical SDOs.

They are essentially a Java representation of the database schema and are different from the logical SDOs which were discussed earlier. Logical SDOs represent the OAGIS nouns, which are the external view of the business objects, as exposed by the components.

The data service layer performs bi-directional transformations between physical SDOs and logical SDOs. You do create, read, update and delete operations on the logical SDOs by plugging component-specific code into the data service layer. The data service layer also lets you do these operations directly on the physical data, bypassing the logical schema altogether.

XPath is used as a query language on the logical schema. The data service layer maps XPath queries to templates of SQL statements. These templates are used to generate actual SQL statements, which access the database. The templates can contain business context variables, which get substituted when the SQL statements are generated. The XPath queries sent in by the client can result in different actual SQL queries, depending on the property values of the business context. The statements are stored in a separate query template file, which isolates the runtime logic from the SQL query code. Finally, the data service layer implements the access profile, by which the client can provide a hint on the amount of data it wants to receive.

Data service layer overview



The Data service layer consists of three pieces: the data service façade, the business object mediation service, and the physical persistence service.

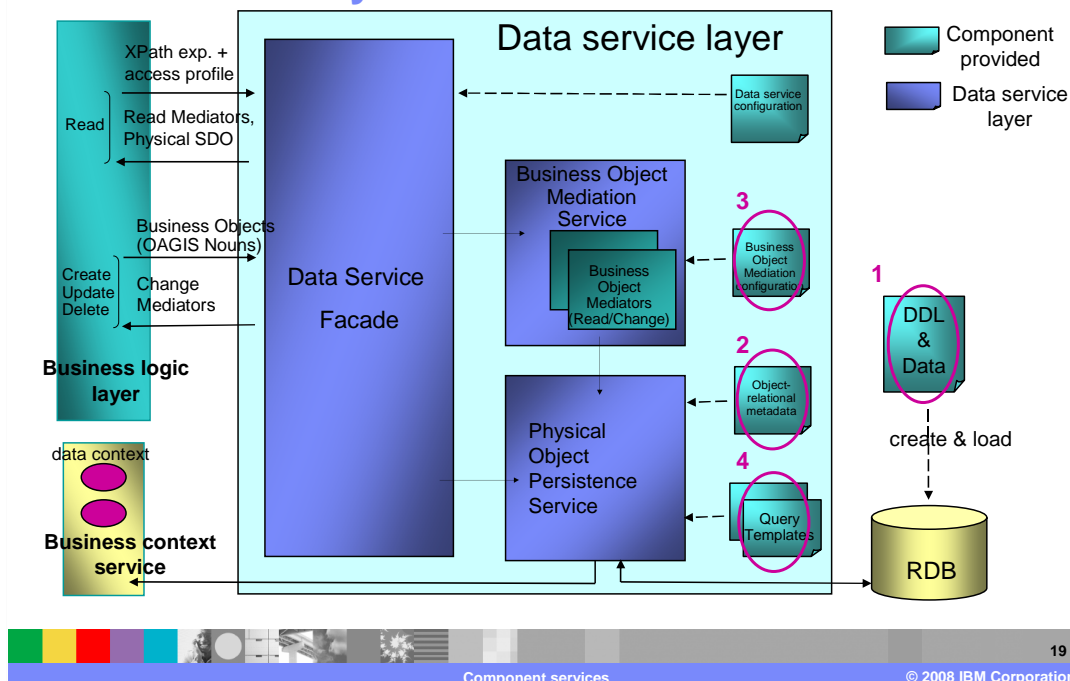
The data service façade is a thin layer that provides a single entry point into the data service layer. It provides interfaces to work with both physical and logical data. It delegates to the business object mediation service or to the physical persistence service. It also allows each component to register with the data service layer and loads the component-specific configuration files.

The business object mediation service initializes mediators. These are classes that transform between the logical and physical representations of the domain model. This allows the business logic layer to deal only with the logical representation of the data. Each component provides its own mediators. There are two kinds: read and change mediators. They are listed in the business object mediation configuration file. Read mediators are used to process the OAGIS Get requests.

In a read scenario, the façade receives a query from the business logic layer. The query consists of an XPath expression plus an access profile, which are extracted from the OAGIS Get verb. The façade forwards this query to the business object mediation service which, in turn, passes it to the persistence service. That service looks up the right SQL template for the query, and uses it to generate one or more SQL statements. It then runs these statements, and maps their result sets into physical SDOs. The persistence service returns the physical SDOs to the business object mediation service where its configuration describes how to instantiate the necessary mediators. These are returned to the business logic layer.

The update scenario is different. The façade receives the OAGIS nouns as input, and passes them to the business object mediation service, which instantiates the appropriate change mediators. The mediators fetch the physical representation of the nouns from the persistence service, which fetches them from the database. The business object mediation service then returns the mediators to the business logic layer. There, they are called with specific actions to create, update, or delete nouns and noun parts. After making all its changes, the business logic layer instructs the change noun mediator to save the updated physical SDOs. The mediator calls the physical object mediation service to update the database.

Data service layer customization



Consider a customization scenario where you add new information to the logical model, such as warranty information for products. You want the new data to be stored in the database and to provide it to the client on demand. In this case, you can use your noun's UserData element to make this information available to the client. This does not require any changes to the logical model.

You store the new data in a new table that has a foreign key relationship to the table representing your noun. For example, if you are adding warranty information for a product, you create a warranty table that references the CATENTRY table. You can do this without writing any Java code. You only need to change the configuration and add SQL statements that fetch the new information.

The circled components on the slide indicate where changes are needed. First, a DDL script must be written to add the new tables. Next, the object-relational metadata file must be updated to describe the new tables and how they map to the physical Java objects. This metadata is also used to generate the physical SDO Java classes. These classes represent tables in the WebSphere Commerce schema. Third, the business object mediation configuration must be updated to map from physical SDOs to the logical model. The physical data properties are represented as name-value pairs in the UserData area of the noun. Finally, in the query template file, new queries must be added to fetch the new data (from the database).

Based on the DDL created manually in step 1, WebSphere Commerce Developer provides a wizard that performs steps 2 and 3, generating the object-relational metadata and the business object mediation configuration. After running the wizard, you must manually add queries to the query template file.

The next series of slides will talk about the query template files in more detail.

Query template file: Purpose

- Map XPath extended query to SQL statements
- Isolate SQL statements from Java code



The query template file provides a mechanism to easily map from the query on your logical model, that is, your XPath query, to one or more SQL statements. These statements retrieve the physical data from the database. Later slides will show how this mapping actually happens.

The file also isolates the SQL statements from the runtime Java code. This makes the code easier to maintain. It is also useful to database administrators when they want to locate and analyze queries. Changes to the SQL queries do not require Java re-compilation.

Query template file: Structure

Five sections

- ▶ Column symbol definitions
- ▶ XPath to SQL statements (defines name, template)
- ▶ Association SQL statements (defines name, template)
- ▶ Profile (references name, graph composer)
- ▶ SQL statement (defines name, template)



The query template file has five main sections. Column symbol definitions define symbols that are used in your SQL template statements. XPath to SQL and association SQL templates are used to generate the queries to be run. A profile section defines the access control for the queries and the SQL statement section contains ad-hoc SQL statements. The next slide shows an example query template file and describes each section in more detail.

Query template file: Example

```

BEGIN_SYMBOL_DEFINITIONS
    COLS:CATENTRY_ID=CATENTRY.CATENTRY_ID
    COLS:CATENTRY=CATENTRY.*
    COLS:CATENTDESC=CATENTDESC.CATENTRY_ID,SHORTDESCRIPTION
END_SYMBOL_DEFINITIONS

BEGIN_XPATH_TO_SQL_STATEMENT
    name=CatalogEntry({PartNumber=})
    base_table=CATENTRY
    sql=
    SELECT
        CATENTRY.$COLS:CATENTRY_IDS
    FROM
        CATENTRY,STORECENT
    WHERE
        CATENTRY.CATENTRY_ID = STORECENT.CATENTRY_ID AND
        STORECENT.STORECENT_ID = $CTX:STORE_IDS AND
        CATENTRY.PARTNUMBER IN (?PartNumber?)
END_XPATH_TO_SQL_STATEMENT

BEGIN_ASSOCIATION_SQL_STATEMENT
    name=IBM_CatalogEntryWithDescription
    base_table=CATENTRY
    sql=
    SELECT
        CATENTRY.$COLS:CATENTRY$,
        CATENTDESC.$COLS:CATENTDESC$
    FROM
        CATENTRY
    LEFT OUTER JOIN CATENTDESC ON
        (CATENTDESC.CATENTRY_ID = CATENTRY.CATENTRY_ID AND
        CATENTDESC.LANGUAGE_ID IN ($CONTROL:LANGUAGES$))
    WHERE
        CATENTRY.CATENTRY_ID IN ($ENTITY_PKSS)
END_ASSOCIATION_SQL_STATEMENT

BEGIN_PROFILE
    name=IBM_Summary
    BEGIN_ENTITY
        base_table=CATENTRY
        associated_sql_statements=IBM_CatalogEntryWithDescription
    END_ENTITY
END_PROFILE

```

Column symbol definitions section

XPath to SQL statements section

Association SQLs section

Access profile section

22

Component services

© 2008 IBM Corporation

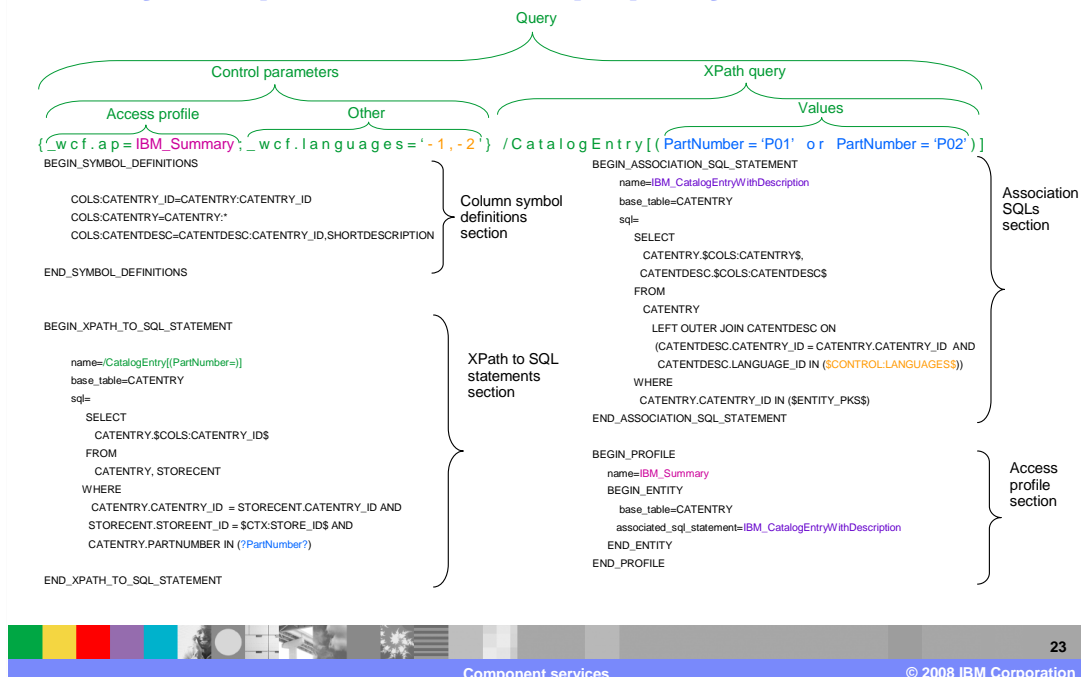
Here is an example of a query template file. The first section defines columns used in the select list of the SQL statements defined in the file. Symbol definitions are similar in concept to constants in Java. The idea is to allow the information to be defined once and used in multiple places. This not only helps localize where to update if changes are required, it also clearly identifies the intended usage with meaningful symbol names. Queries that use these definitions select the new columns. You can define the value of the column symbol to be a list of columns that are selected from a table. In addition, wildcards ('*') can be defined in the symbol definition. Then, the actual column names are retrieved from the object-relational metadata. In this case, adding a new column to the select clause is done automatically once the column information is added to the object-relational metadata.

The sections for XPath to SQL statements and association SQL statements each define a list of template SQL statements. Each SQL template has a unique name and a base table name. The base table refers to the main table in the query, which is typically the table representing your noun. Each XPath to SQL template, when used in combination with the association SQL statements, returns a list of primary keys satisfying a search criteria. For example, a search criteria might be to select a list of product IDs given the category name.

The profile section defines access profiles that make use of association SQL statements. If needed, more than one association SQL statement can be used by a profile. In that case, each associated SQL statement is performed in turn and the results of the different associated SQL statements are merged together.

At the conceptual level, XPath to SQL statements locate the objects that are of interest based on primary keys; but the association SQL statements, scoped by profile name, retrieve the information about those objects.

Query template file: 2-step query



23

Component services

© 2008 IBM Corporation

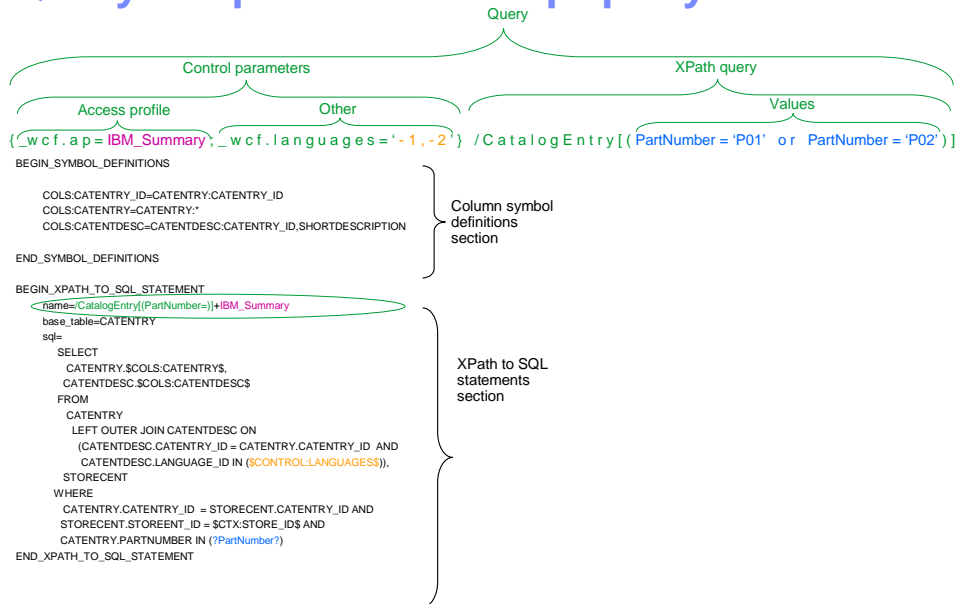
At the top of this slide is an example of a query. It consists of the XPath query and control parameters. The XPath query is a string from the slash to the end of the string. The control parameters come before it, between the braces. The names of the predefined control parameters are prefixed with '_wcf'. The first parameter is the access profile. The profile's name is IBM_Summary. You can have many control parameters, but the access profile is always needed. In this example, there is one extra control parameter, the languages, specifying languages with ids -1 and -2 (English and French). The XPath query asks for catalog entries with part numbers 'P01' and 'P02'.

When the query arrives, the XPath query portion is isolated and split into a key and values. The key is used to find an XPath to SQL statement entry by the same name in the file. This is the one that fetches the primary keys. The values, 'P01' and 'P02', are substituted into the template, replacing the 'PartNumber' parameter.

Next, the name of the access profile is used to lookup its definition. The profile points to the definition of an association SQL statement. The other control parameters are then substituted into both queries. In this example, there is no substitution variable for languages in the XPath to SQL template, but there is one in the association SQL template.

Once all substitutions are made, the XPath to SQL query is run to fetch the primary key values. These values are injected into the associated SQL statement replacing the \$ENTITY_PKSS\$ tag. Finally, this statement is run to retrieve all the data requested by the original query.

Query template file: 1-step query



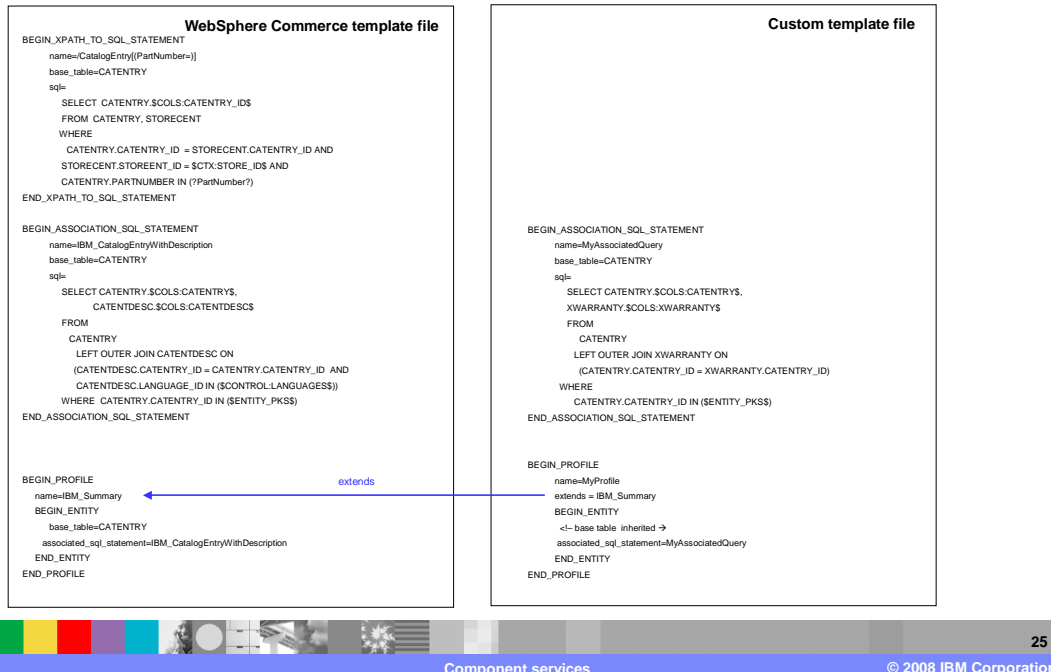
Here is a template file with a single-step SQL statement. In this example the two templates have been combined into one.

Similar to the last example, there is an XPath query, an access profile and other control parameters. This time the access profile and XPath query are combined into a single name, which selects a single template in the file.

You should use single-step queries if possible. However, in some cases it is not possible to fetch all the data in a single query or such a query needs to join a very large number of tables and might not perform well. In this case a 2-step query should be used.

Another reason to use a 2-step query is when paging is requested by the client. Paging on the result of a 1-step query is not possible if it returns multiple records for each base table record. A 2-step query allows you to page on the result set returned by the first statement (the primary keys) rather than on the result set of the second statement.

Extending 2-step query





For example, you have added a new table to the WebSphere Commerce schema and you want your client to receive the new data along with the regular data. For a 2-step query, you must add a new access profile. This profile extends from the standard one. It adds a new association SQL template to fetch the new data. This slide shows an example of this configuration.

The new association SQL statement is fetching the data from the new table, XWARRANTY. The new profile, MyProfile, references this association statement. It also extends from the predefined profile, IBM_Summary, which continues to fetch all the regular data. Finally, the client hooks into this new functionality by sending a modified query, which contains the new profile instead of the standard one.

Extending 1-step query

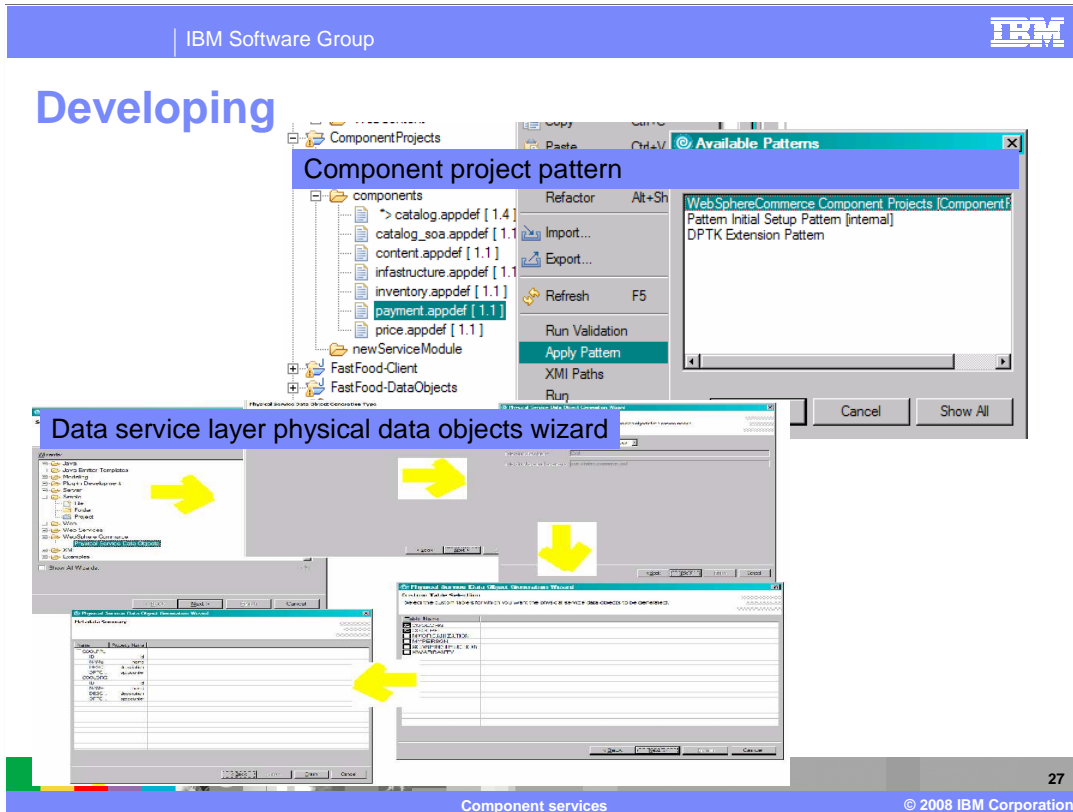
WebSphere Commerce template file	Custom template file
<pre> BEGIN_SYMBOL_DEFINITIONS COLS:CATENTRY_ID=CATENTRY:CATENTRY_ID COLS:CATENTRY=CATENTRY:* COLS:CATENTDESC=CATENTDESC:CATENTRY_ID,SHORTDESCRIPTION END_SYMBOL_DEFINITIONS BEGIN_XPATH_TO_SQL_STATEMENT name=/CatalogEntry({PartNumber-})+IBM_CatalogEntryWithDescription base_table=CATENTRY sql= SELECT CATENTRY.SCOLS:CATENTRYS, CATENTDESC.SCOLS:CATENTDESCS FROM CATENTRY LEFT OUTER JOIN CATENTDESC ON (CATENTDESC.CATENTRY_ID = CATENTRY.CATENTRY_ID AND CATENTDESC.LANGUAGE_ID IN (\$CONTROL:LANGUAGES)), STORECENT WHERE CATENTRY.CATENTRY_ID = STORECENT.CATENTRY_ID AND STORECENT.STOREENT_ID = \$CTX:STORE_IDS AND CATENTRY.PARTNUMBER IN (?PartNumber?) END_XPATH_TO_SQL_STATEMENT </pre>	<pre> BEGIN_SYMBOL_DEFINITIONS COLS:CATENTRY_ID=CATENTRY:CATENTRY_ID COLS:CATENTRY=CATENTRY:* COLS:CATENTDESC=CATENTDESC:CATENTRY_ID,SHORTDESCRIPTION COLS:XWARRANTY=XWARRANTY:* END_SYMBOL_DEFINITIONS BEGIN_XPATH_TO_SQL_STATEMENT name=/CatalogEntry({PartNumber-})+MyProfile base_table=CATENTRY sql= SELECT CATENTRY.SCOLS:CATENTRYS, CATENTDESC.SCOLS:CATENTDESCS, XWARRANTY.SCOLS:XWARRANTYS FROM CATENTRY LEFT OUTER JOIN CATENTDESC ON (CATENTDESC.CATENTRY_ID = CATENTRY.CATENTRY_ID AND CATENTDESC.LANGUAGE_ID IN (\$CONTROL:LANGUAGES)), STORECENT LEFT OUTER JOIN XWARRANTY ON (CATENTRY.CATENTRY_ID = XWARRANTY.CATENTRY_ID) WHERE CATENTRY.CATENTRY_ID = STORECENT.CATENTRY_ID AND STORECENT.STOREENT_ID = \$CTX:STORE_IDS AND CATENTRY.PARTNUMBER IN (?PartNumber?) END_XPATH_TO_SQL_STATEMENT </pre>



26

Component services
© 2008 IBM Corporation

This slide shows an example of extending a 1-step configuration. Note that for a 2-step query you created a new association template and used it in conjunction with an existing XPath to SQL template. For a 1-step query, you instead create a new XPath to SQL template, not an association template.

The name of the query is a combination of the new access profile and the XPath key from the default 1-step query. Extending a 1-step query is similar to adding a new finder method to an Enterprise Java Bean.



WebSphere Commerce Developer provides tools that simplify the customization process. The Design Pattern Toolkit is an Eclipse-enabled template engine for generating applications based on customizable, model-driven architecture transformations. WebSphere Commerce uses the Design Pattern Toolkit plug-in for creating WebSphere Commerce service modules from a simple XML file. By describing the service module in a specialized XML syntax, the service modules can be generated. This allows you to start directly with the service module implementation without having to spend time on the setup and configuration of a service module.

The WebSphere Commerce Physical Service Data Objects Wizard has been added to WebSphere Commerce Developer to simplify data service layer customization by automatically generating configuration information. The wizard generates the physical SDO Java classes for your customizations, along with the required object-relational metadata and physical to logical business object mediator mappings. These steps are required if the database schema has changed (new table, new relationships between existing tables, new columns, changed column types) and for any new service modules you develop. XML assets that are generated are stored in the component configuration extension directories and custom physical SDOs are stored inside the `WebSphereCommerceServerExtensionsLogic` project.

Problem determination

- Tracing is controlled at a Java package level
 - ▶ Tracing can be much more granular
- Noteworthy tracing components
 - ▶ Data service layer
 - `com.ibm.commerce.foundation.server.services.dataaccess.*`
 - ▶ Command framework
 - `com.ibm.commerce.foundation.server.command.*`
 - ▶ Client
 - `com.ibm.commerce.foundation.client.*`



Beginning in Feature Pack 2 you now have the ability to enable trace at the Java package level. This provides you with the ability to trace only the areas of the code you need and speeds up problem determination by reducing unneeded information in the trace file.

Tracing can be enabled at different points throughout the Java package hierarchy as shown here. You can enable data service layer tracing for diagnosing configuration issues with the data service layer. Command Framework tracing is useful for diagnosing the processing of the request (including the request and response BODs). Client tracing is used to diagnose configuration issues when the client sends the request to the server.

Tracing is enabled through the WebSphere Application Server administration console.

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_WCS6003_ComponentServices.ppt

This module is also available in PDF format at: [../WCS6003_ComponentServices.pdf](..WCS6003_ComponentServices.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:

IBM WebSphere

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

Java, JDBC, 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.

© Copyright International Business Machines Corporation 2008. All rights reserved.

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.