Tutorial: Design Patterns in the VSM

A software *design pattern* describes an approach to solving a recurring programming problem or performing a task. The design patterns discussed in this article can improve the performance of J2EE applications over the Internet or an intranet, and make them more flexible and easier to maintain.

OTN members can download complete VSM source code and installation instructions.

Contents

Required Software

This tutorial presents several code examples. If you want to study them in context, download and install the VSM source code. If you also want to build and run the VSM application, you will need the software listed in the Required Software section of *About the Virtual Shopping Mall*.

Concepts

The VSM sample application demonstrates ways to implement the following design patterns.

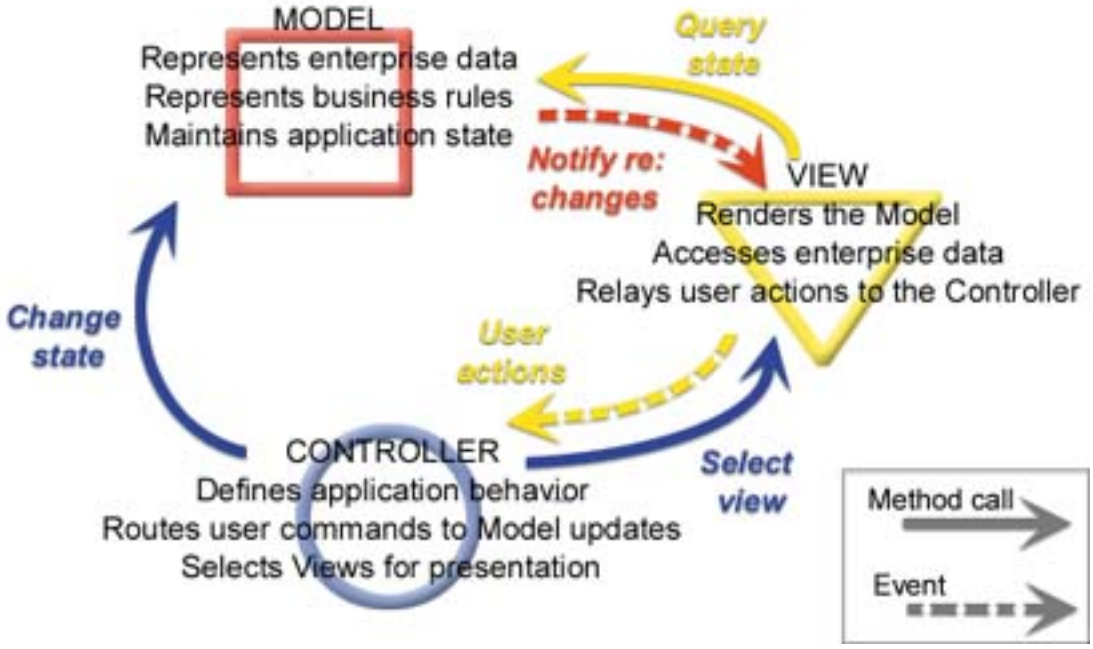
- Model-View-Controller
- Session Façade
- Value Object
- Service Locator
- UUID for EJB

MVC (Model–View–Controller)

The MVC design pattern differs from the others discussed in this article, which describe ways to solve specific programming problems or perform specific tasks, because the MVC design pattern describes an approach to an application as a whole. The approach is multi-tiered and modular; the MVC design pattern identifies three entities, each operating in a different logical layer within the application space:

- **Model**: A back-end representation of enterprise data and business logic, the model also maintains data about the state of the application. In many cases, the model is a logical representation of a real-world process.
- **View**: The front-end presentation layer that renders the model for end-users. Views also provide user interfaces for accessing the enterprise data represented by the model.
- **Controller**: The middle tier of the MVC pattern. The controller defines application behavior, selecting views for presentation and capturing a user's interactions with views and routing them as commands to the model.

The MVC design pattern defines a clear separation of application logic, presentation details, business rules and data. As a result, multiple clients, displays, and devices can use the same application and business rules to work with the same data. The following figure shows how model, view, and controller interact.



This version of the VSM uses Apache Struts as the MVC framework. For more information, see the tutorial *Building a Web Application with Struts*.

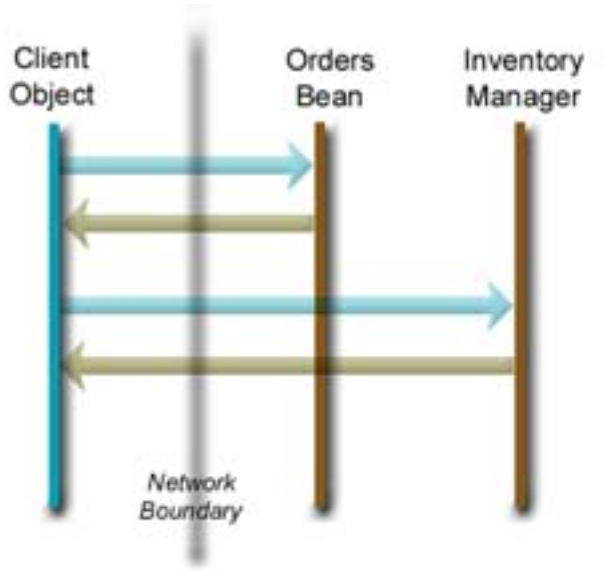
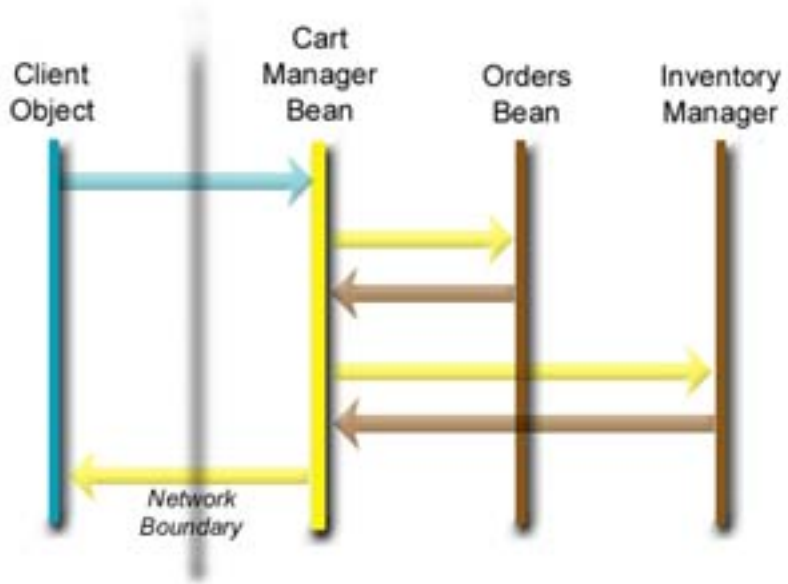
Session Façade

The Session Façade design pattern is useful in situations where client objects need to interact with a set of EJBs to perform tasks in a workflow. For example, in the VSM environment, customers browse shops and order products, shop owners track orders and maintain inventory, and adminstrators approve and reject requests for new shops and manage category lists. In an implementation where client objects interact directly with the underlying EJBs, the following problems can arise:

- When an EJB's interface changes, client objects must also be updated. This situation is analogous to the brittle class problem common in object-oriented programming.
- To carry out a workflow, client objects must make numerous remote calls to access the EJBs, leading to increased network traffic and reduced performance.

A session façade solves such problems by presenting client objects with a unified interface to the underlying EJBs. Client objects interact only with the façade, which resides on the server and invokes the appropriate EJB methods. As a result, dependencies and communication between clients and EJBs is reduced. A session façade can also simplify transaction management: for example, when a database transaction involves multiple method calls, all could be wrapped in one method of the façade and the transaction could be monitored at that level.

The following figures show this client-EJB interaction with and without the session façade, and how the session façade reduces network traffic.

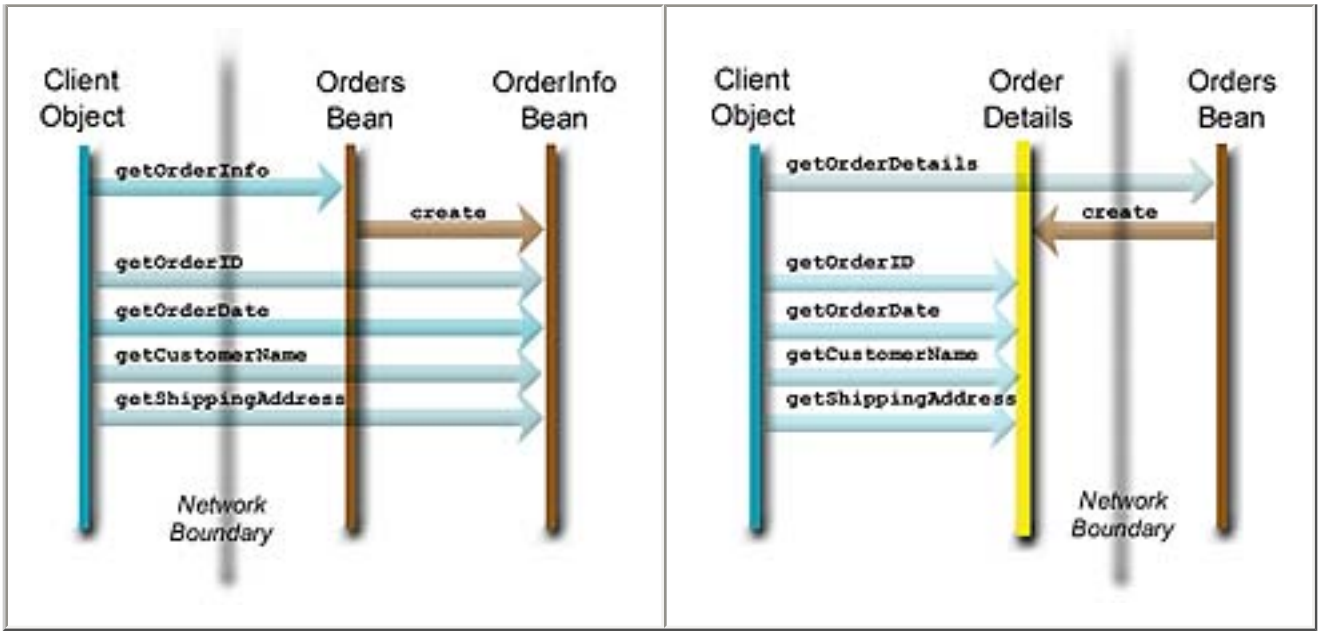
Without Session Façade	With Session Façade (CartManagerBean)
	

Value Object

The Value Object design pattern (also known as Data Transfer Object) describes a container for a set of related data. It is often (but not necessarily) used with the Session Façade pattern to reduce network traffic and the number of method calls required to get an entity's attribute values. For example, when a customer uses the VSM to buy a product, the application generates an order comprising several attributes including order ID, order date, customer name, and shipping address. To retrieve the details of an order, an application that does not implement the Value Object pattern would have to make a remote `get` method call for each attribute (example: `Orders.getOrderID`), adding to network traffic and increasing EJB container resource usage.

The figures below show interactions with and without the Value Object design pattern, and how the pattern reduces network traffic.

Without Value Object	With Value Object (OrderDetails)



Service Locator

The Service Locator design pattern gives clients a simple interface to the potentially complex details of finding and creating application objects and services. Without a Service Locator, clients must use the JNDI (Java Naming and Directory Interface™) API to perform resource-intensive lookup and creation tasks. A Service Locator simplifies client-side code: it abstracts and encapsulates such dependencies and network details, and moves logic to the server tier. In addition, the Service Locator caches the initial context objects and references to the factory objects (e.g., EJBHome interfaces, JMS connection factories) shares them with other clients to improve overall application performance.

UUID for EJB

The UUID for EJB design pattern is described in the book *EJB Design Patterns* by Floyd Marinescu. Here is an excerpt from the text:

"A UUID is a primary key encoded as a string that contains an amalgamation of system information that makes the generated UUID completely unique over space and time, irrespective of when and where it was generated. As a completely decentralized algorithm, there can be multiple instances of UUIDs across a cluster and even in the same server, allowing for fast and efficient primary key generation."

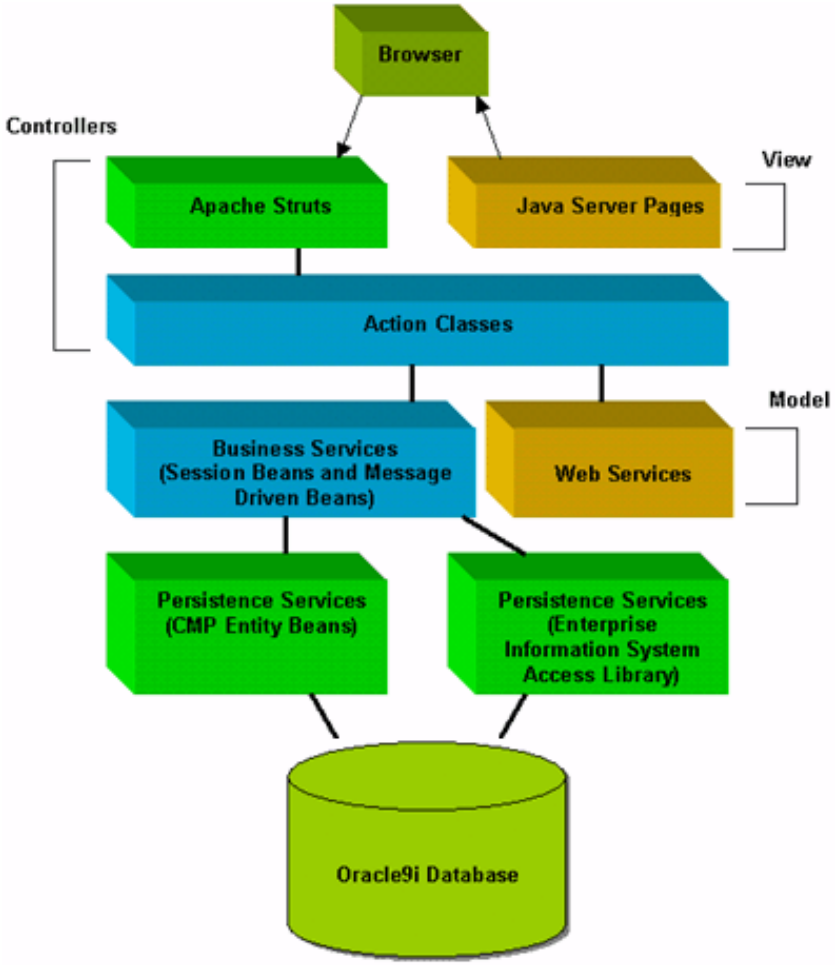
Design

The VSM sample application was designed to demonstrate key features of Oracle9*i* JDeveloper, including:

- Implementation of J2EE design patterns.
- Struts, an open source framework for building Web applications. The core of the Struts framework is a flexible control layer based on standard technologies like Java Servlets, JavaBeans, ResourceBundles, and Extensible Markup Language (XML). Support for Struts is built into JDeveloper, and OTN developers used it to build the interface and Web application for VSM end-users and shop owners. For details, see Building the End-User's Interface (Struts).

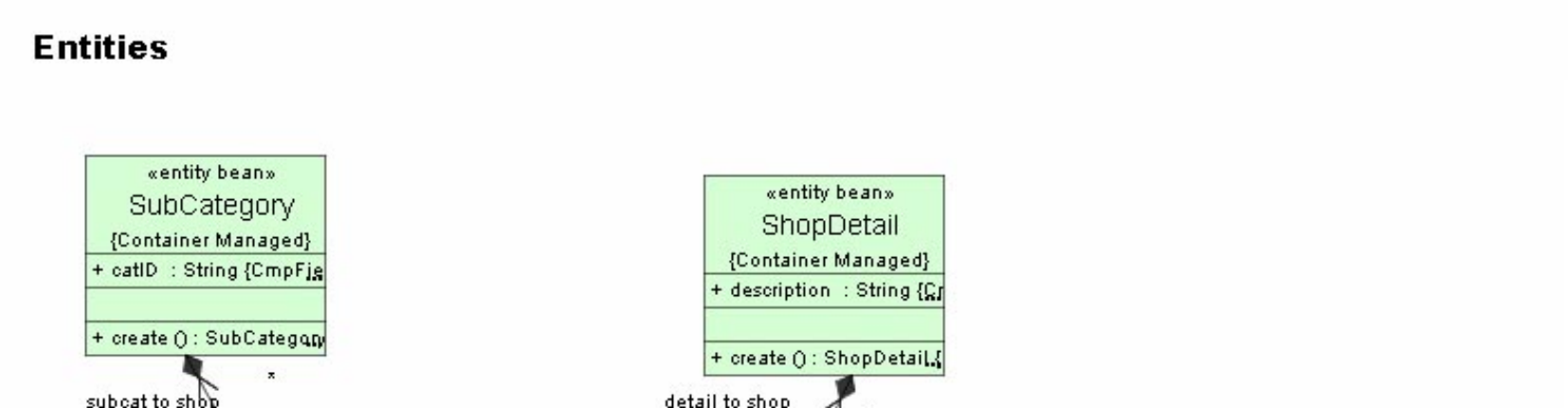
Architecture Diagram

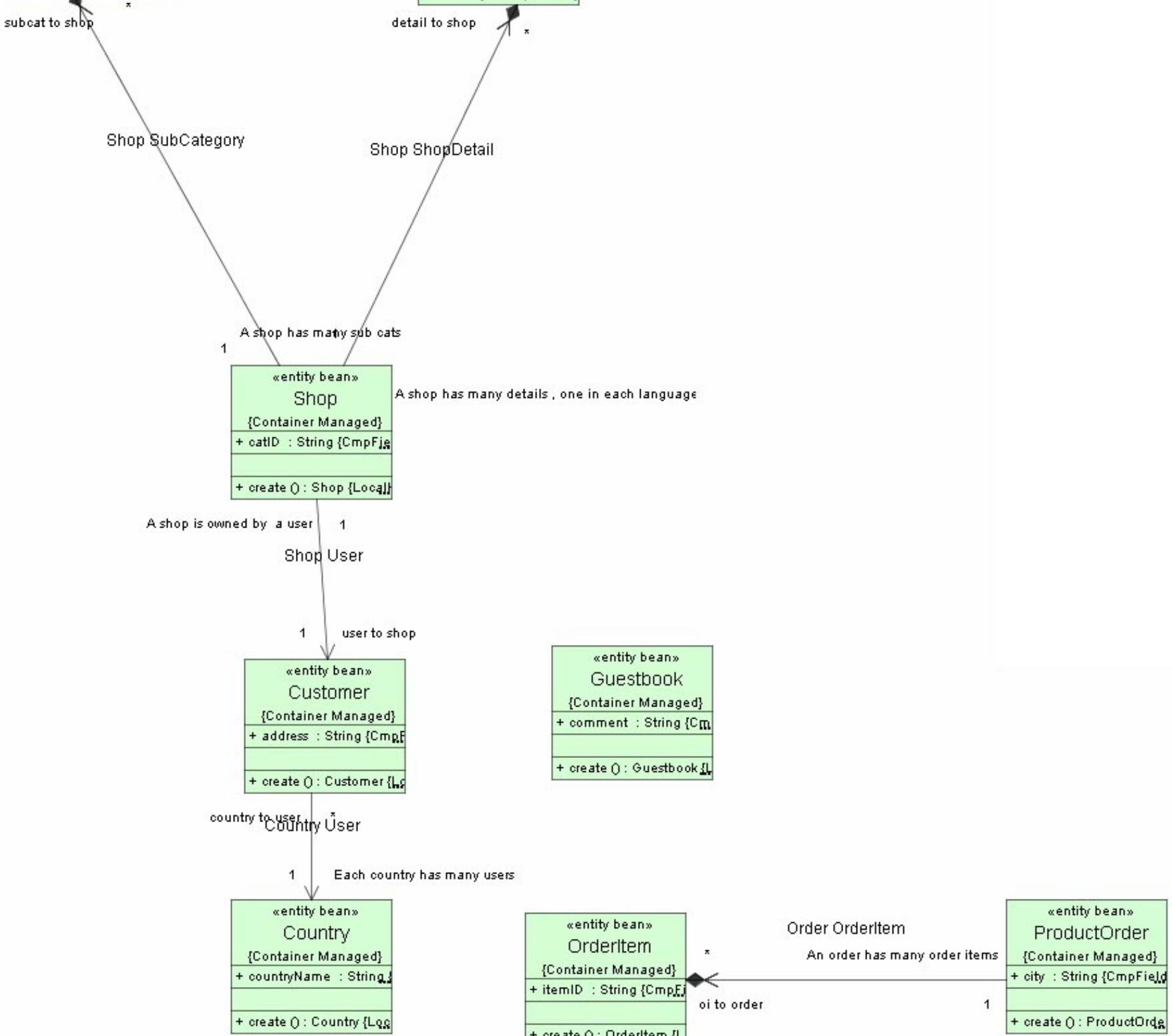
The following figure gives a high-level view of the VSM architecture.

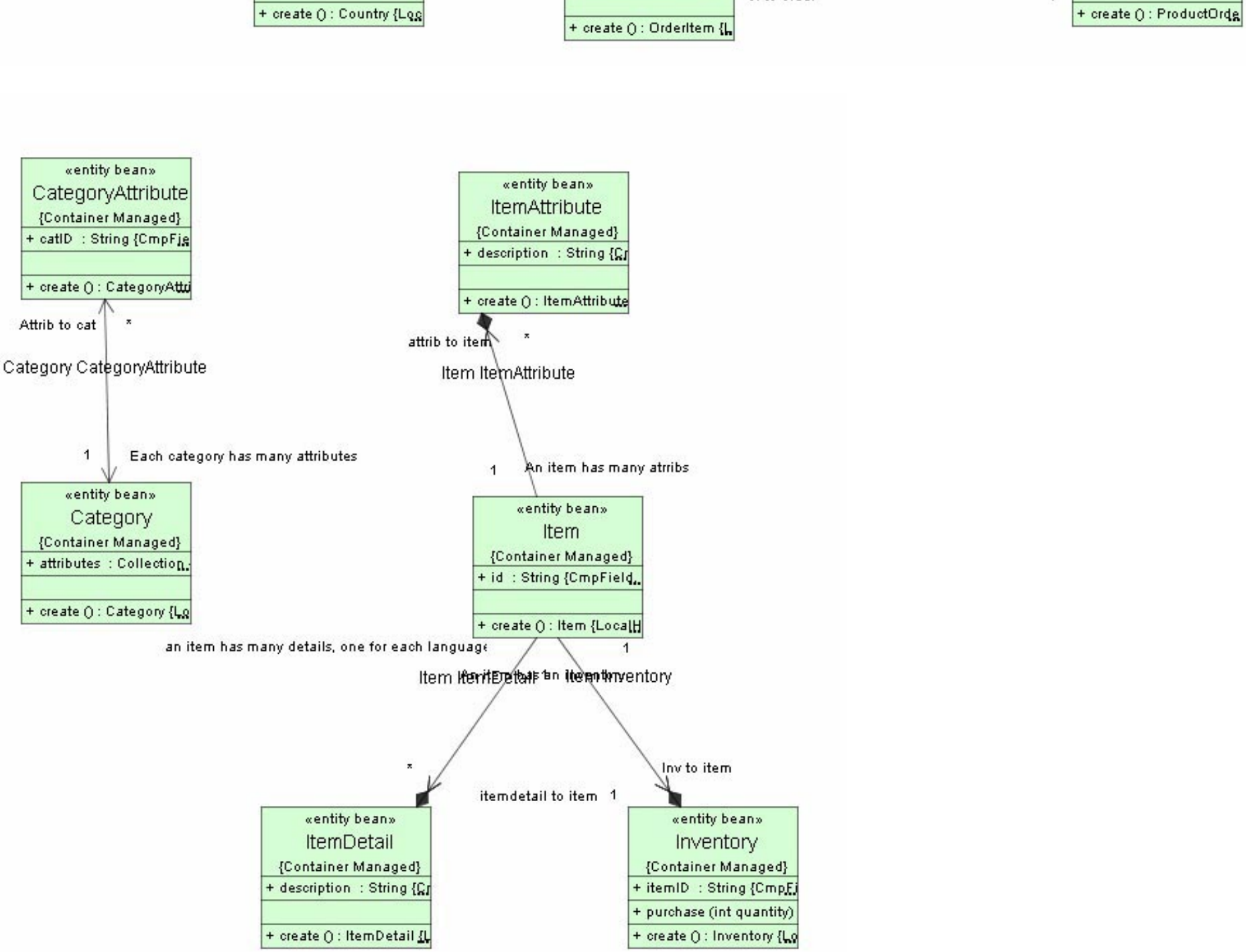


Entity Diagram

The figures below were created using JDeveloper. They show the most important VSM entities and the relationships between them.




subcat to shop
*
detail to shop
*
Shop SubCategory
Shop ShopDetail
A shop has many sub cats
1
«entity bean»
Shop
{Container Managed}
+ catID : String {CmpFie
+ create () : Shop {Local
A shop has many details , one in each language
A shop is owned by a user
1
Shop User
1
user to shop
«entity bean»
Customer
{Container Managed}
+ address : String {Cmp
+ create () : Customer {L
«entity bean»
Guestbook
{Container Managed}
+ comment : String {Cm
+ create () : Guestbook {L
country to user
*
Country User
1
Each country has many users
«entity bean»
Country
{Container Managed}
+ countryName : String
+ create () : Country {Loc
«entity bean»
OrderItem
{Container Managed}
+ itemID : String {CmpF
+ create () : OrderItem {L
*
Order OrderItem
An order has many order items
oi to order
1
«entity bean»
ProductOrder
{Container Managed}
+ city : String {CmpFiel
+ create () : ProductOrd



Other apects of the VSM design are covered in various lessons in this tutorial series.

Setup

No special setup steps are required to implement J2EE design patterns. If you use Oracle9*i* JDeveloper as your IDE, add the J2EE library to your project. For coding details, see the Implementation section.

To configure your system to build and run the VSM sample application, see the Setup section of *About the Virtual Shopping Mall*.

Implementation

The VSM sample application demonstrates ways to implement the following design patterns.

- Model-View-Controller
- Session Façade
- Value Object
- Service Locator

MVC (Model–View–Controller)

The MVC design pattern defines a clear separation of application logic, presentation details, business rules and data. As a result, multiple clients, displays, and devices can use the same application and business rules to work with the same data.

- The *model* is implemented by EJBs and other Java classes, many of which represent real-world objects. such as shops, shopping carts, and orders. For implementation examples, see `MallAdministration.java`, `ShopOwner.java`, and `ShoppingCart.java`.
- The *views* are provided by JavaServer Pages (JSPs) rendered in a browser. Examples include `shoppingMall.jsp`, `shopOwner.jsp`, and `shoppingCart.jsp`.
- The central *controller* is implemented by Apache Struts and the servlet is `org.apache.struts.action.ActionServlet`. When an end-user interacts with a view (for example, by submitting a form), this HTTP servlet dispatches the action to a controller object which in turn invokes methods on objects in the model.

Session Façade

In the VSM, the `CartManagerBean` implements a session façade that provides an interface to the EJBs that manage the items in a customer's shopping cart. The `CartManagerBean` exposes the `checkOutCart` method to clients, encapsulating inventory management and order creation tasks performed by underlying objects `InventoryManager` and `OrdersBean` (accessed via `OrdersHome`).

```
public StringBuffer checkOutCart( ShoppingCart cart )
       throws CartException {
  try {
    // Get the order home
    OrdersHome home = (OrdersHome)ServiceLocator.getLocator().
                                                  getService( "Orders" );
    ...
    // For each item in the cart
    for( int i = cart.getItems().length - 1; i >= 0; i-- ) {
      currentItem = (CartItem)cart.getItems()[i];
      // check the inventory
      if( !InventoryManager.inventoryCheck( currentItem.getID(),
                                            currentItem.getQuantity() ) ) {
        response.append( "Failure," );
        response.append( currentItem.getID() );
        response.append( "," );
        response.append( InventoryManager.getInventory(currentItem.getID()));
        continue;
      }
      ...
        // Create the order and add it to the table
        order = home.create( new Integer( orderID ), details );
        shops.put( shopID, order );
      }
      ...
      // Add it to the order for the shop
      order.setOrderItemID( item );
      cart.removeItem( i );
    }
    if( response.length() < 1 ) {
      response.append( "Checked out your cart successfully" );
    }
    return response;
    ...
  }
  ...
}
```

Value Object

Tthe VSM implements the Value Object design pattern in several places, creating a container object and sending it across the network to the client, which can then access the data via local method calls. For example, the following code from `CartManagerBean.checkOutCart` uses an `OrderDetails` object to store data for a given order.

```
public StringBuffer checkOutCart( ShoppingCart cart )
       throws CartException {
  ...
        // Create order details
        details = new OrderDetails( orderID,
                                    new Date(),
                                    cart.getUserName(),
                                    shopID.intValue(),
                                    shippingAddress.getAddress(),
                                    shippingAddress.getCity(),
                                    shippingAddress.getState(),
                                    shippingAddress.getCountry(),
                                    shippingAddress.getZip(),
                                    shippingAddress.getPhone() );
        // Create the order and add it to the table
        order = home.create( new Integer( orderID ), details );
        shops.put( shopID, order );
  ...
}
```

The VSM application implements the Value Object design pattern in these files:

- `UserDetails.java`
- `ItemDetails.java`
- `ItemAttributes.java`
- `OrderDetails.java`
- `ShopDetails.java`

Service Locator

In the VSM, `ServiceLocator.java` implements this design pattern. This singleton class is the central place for looking up objects in the JNDI tree, as shown in the following code from `ServiceLocator.getService`, which finds an object matching a supplied JNDI name.

```
public class ServiceLocator {
...
  /**
   * Method to return an object in the default JNDI context, with
   * the supplied JNDI name.
   * @param <b>jndiName</b> The JNDI name
   * @returns <b>Object</b> The object in the JNDI tree for this name.
   * @throws <b>UtilityException</b> Exception this method can throw
   */
  public Object getService( String jndiName )
                throws UtilityException {
    try {
      // If the service is not in the cache,
      if( !homeCache.containsKey( jndiName ) ) {
        // Get the object for the supplied jndi name and put it in the cache
        homeCache.put( jndiName, defaultContext.lookup( jndiName ) );
      }
    } catch( NamingException ex ) {
      throw new UtilityException( "Exception thrown from getService " +
                                  "method of ServiceLocator class of given "+
                                          "status : " + ex.getMessage() );
    } catch( SecurityException ex ) {
      throw new UtilityException( "Exception thrown from from getService " +
                                  "method of ServiceLocator class of given "+
                                          "status : " + ex.getMessage() );
    }
    // Return object from cache
    return homeCache.get( jndiName );
  }
...
}
```

Resources

This tutorial is part of a series based on the Virtual Shopping Mall (VSM) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in the tutorials. See the Required Software section to obtain the VSM source code and related files.

Resource	URL
Oracle9*i* JDeveloper Online Help	http://otn.oracle.com/jdeveloper903/help/
J2EE Design Patterns	http://java.sun.com/blueprints/patterns/j2ee_patterns/index.html
EJB Design Patterns	http://www.theserverside.com/books/EJBDesignPatterns/index.jsp
J2EE 1.3 SDK	http://java.sun.com/j2ee/sdk_1.3/

Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the OTN Sample Code discussion forum. OTN developers and other experts monitor the forum.
- Send email to the author. mailto:Robert.Hall@oracle.com

If you have suggestions or ideas for future tutorials, you can

- Post a message in the OTN Member Feedback forum.
- Send email to mailto:Raghavan.Sarathy@oracle.com.