Rational Software

PearlCircle Online Auction Reference Application
Software Architecture Document

Issue 0.2

Revision History

Date	Issue	Description	Author
July 13, 2001	0.1	Initial version of the document	Wojtek Kozaczynski
September 13, 2001	0.2	Incorporated PCOA model changes	Wojtek Kozaczynski

Table of Contents

Software Architecture Document

1. Brief Description

This document provides a comprehensive overview of the architecture of the PearlCircle Online Auction reference application (hereafter called Online Auction). The document is intended to capture and convey the significant design decisions underlying the architecture of the system. It serves as a communication medium between the software architect and other project team roles regarding those decisions.

Note: some of the diagrams in this document are best viewed by increasing the Zoom to 200%-300% or more.

2. References

In the references below ".\" stands for the directory where the reference application files were placed. The internal reference documentation includes:

- Supplementary Specification [.\reference application\model and requirements\Requirements\Supplementary Specification.doc]
- Glossary [.\reference application\model and requirements\Requirements\Glossary.doc]
- Rose models [.\reference application\model and requirements\ PearlCircleJ2EE_v1.0.mdl, and its controlled units]
- Use case specifications [see .\reference application\model and requirements\Requirements\ directory]

As explained below, the architecture of the application borrows extensively from the Sun Java patterns described in the book: *Core J2EE™ Patterns, Best Practices and Design Strategies; Deepak Alur,* John Crupi, and Dan Malks

The top-level decomposition of the system uses the concept of *business component* introduced and described in the book: *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*; Peter Herzum and Oliver Sims.

3. Architectural Representation

The architecture of the application is represented following the recommendations of the Rational Unified Process and the Rational Architecture Practice guidelines. The UML specification of the systems has been divided into six models:

- Use Case Model [Use Case View::Use-Case Model]
- Analysis Model [Logical View::Analysis Model]
- User Experience Model [Logical View::User Experience (UX) Model]
- Design Model [Logical View::Design Model]
- Implementation Model [Component View::com], and
- Deployment Model [Deployment View]

The Logical View and the Component View also include packages that represent Java language and J2EE framework elements. Collectively the above models and packages form a complete UML specification of the system.

4. Architectural Goals and Constraints

The Online Auction is a reference application and as such, it has been created with the following objectives in mind:

1. To serve as an example of how to design, document and develop a J2EE-based application using the RationalRose and an IDE
2. To show a result of applying the Rational Unified Process guidelines to developing a J2EE application
3. To be a base for deriving a set of reusable design templates (referred to in the design as mechanisms) for

on-line applications.

The major design and implementation constraint has been that the application should run on an open-source, reference implementation-compliant deployment platform. In our case the platform consists of the following three components:

1. TomCat 3.2.1 JSP and JavaServlet container
2. JBoss 2.0 EJB container, and
3. MySQL 3.2.3. DBMS.

TomCat 3.2.1 implements the Servlet 2.2 and JSP 1.1 specifications. JBoss 2.0 implements the EJB 1.1 specification.

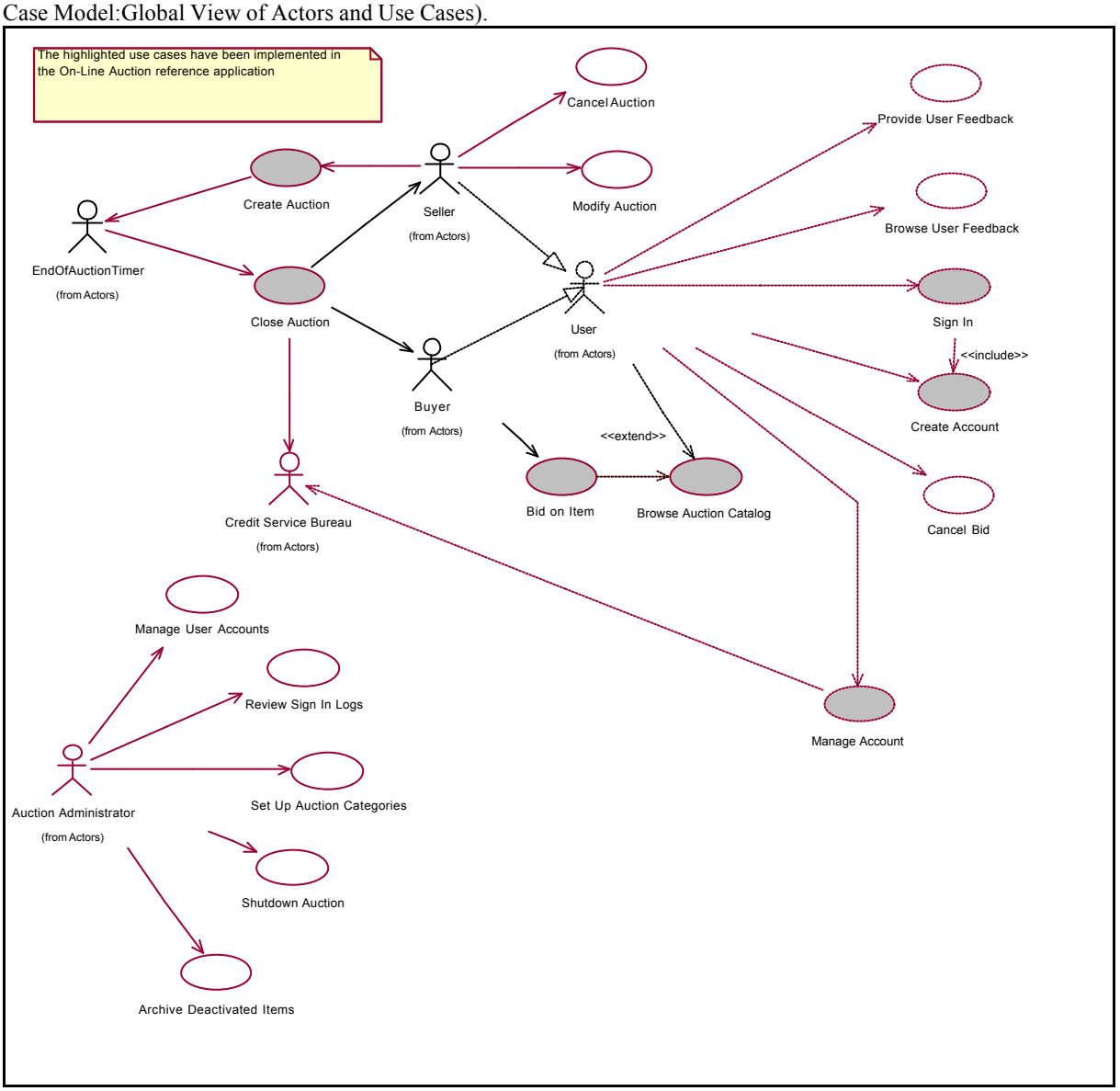
The reference application is self-contained. Any connections with external systems such as a credit verification system have been stubbed. Hence, the architecture has not been subject to integration constraints.

While designing the application we did not explicitly address scalability issues such as load balancing and/or deploying the system on a farm of application servers. We have assumed that issues like that are best addressed in the context of a particular deployment configuration and that commercial deployment environments, for example WebLogic or WebSphere, provide proprietary support for such optimizations.

We did not try to optimize DB access by using bean-managed persistency. All entity beans persist themselves via the container-provided mechanism. This implies, that in some cases the application may be performing joins at bean level when such join could have been performed more efficiently in the DB engine. We leave DB access optimization as an “exercise for the student”

5. Use-Case View

The functionality of a simple on-line auction system is captured in the use-case diagram below (Use Case View:Use-Case Model:Global View of Actors and Use Cases).



The PearlCircle Online Auction implements the architecturally-significant subset of the use cases highlighted above. These are described in the following section. These architecture-significant use cases illustrate the key functions of most auction systems and exercise all major system components. The remaining use-cases can be rapidly developed without changes to the architecture by following the application structure and by reusing the two mechanisms described in section 6.2.2.

All implemented use cases have an associated Use Case Specification document. References to these documents can be found in the Rose browser window under the respective use case model elements (Use Case View:Use-Case Model:Use Cases:<use-case name>:<use-case name>:<use-case name.DOC>). Each of the documents describes the basic flow and the alternative flows.

5.1 Architecturally-Significant Use Cases

The architecturally-significant use cases are those, that "exercise" the most critical parts of the system architecture and demonstrate the core system functionality. As stated above, the reference application imp lements selected use cases of a "typical" auction system. The implemented use cases are very much those that expose significant parts of the application (this is why they were selected.) The most interesting between them are:

- Create Account (with Sign In); shows how user accounts are created and managed
- Create Auction; shows how auctions are timed-out and closed by an internal timer process, and
- Close Auction; shows how auctions are timed-out and closed by an internal timer process, and
- Bid On Item (with Browse Auction Catalog); shows how the system imposes minimal bid increment rules and maintains multiple bids on an item.

These are shown and briefly described below.

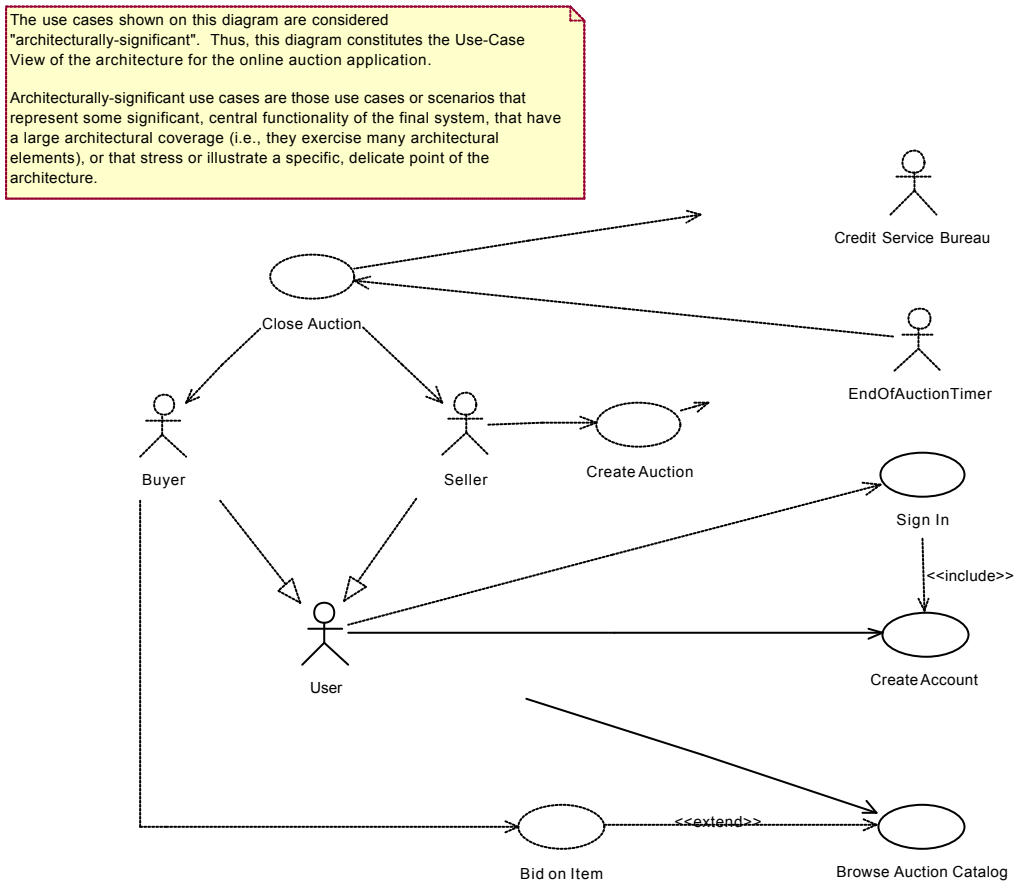


Diagram Documentation:

5.1.1 Bid on Item

Brief Description: When browsing an item currently available via auction (see the Browse Auction Catalog use case), a Buyer may opt to place a bid on the item. The entered bid must be greater than the current bid by an amount greater than the minimum bid increment specified by the Seller. Once accepted the entered bid

becomes the current bid.

The User must be signed in order to bid on an item. See the Sign In use case.

If the auction has been closed, the bid is not accepted. See the Close Auction use case.

If the Buyer has any pending payment notices, a message is displayed to the Buyer, reminding the Buyer that payment for the notices must be made (i.e., new credit card information must be entered) before the User can participate in any auction (as either the Buyer or the Seller. New credit card information can be entered via the Manage Account use case.

5.1.2 Browse Auction Catalog

Brief Description: This use case allows a User to browse the items currently available for auction. The User may search for a specific item, or may look at all of the items currently available for auction, sorted by category.
The system displays information on the requested item, including, but not limited to the item description, current best bid, minimum bid increment, and bidding history.
Once an auction item is displayed, the User has the option of bidding on the item. See the Bid on Item use case.
The User does not have to be signed on to browse the auction catalog.

5.1.3 Close Auction

Brief Description: At the expiration of the End of Auction Timer, this use case closes out an auction. The End of Auction Timer is set when the Seller specifies a bidding time limit when the auction is created. See the Create Auction use case.

When an auction is closed, the current best bid for the item is accepted (the best bid becomes the purchase price for the item). The system notifies the Buyer and Seller that the auction has completed and provides each with the final purchase price for the item, as well as contact information for the other.
The Seller's credit card (credit card information is maintained with the auction) is charged the transaction fee.

5.1.4 Create Account

Brief Description: The Create Account use case allows the User to create and activate an account that contains information for the User.
Once the account has been created and activated, the user is considered to be signed in. For more information on sign in, see the Sign In use case.

5.1.5 Create Auction

Brief Description: The Create Auction use case allows a Seller to create an online auction. The Seller specifies auction information (including the start and end of the auction), his credit card information (if not provided during account creation, see the Create Account use case, or different for this auction) and can provide an image of the item, and an online auction is created for the item. Buyers can then bid on the item in the auction (see the Bid on Item use case).
An end of auction timer is set for the entered end of the auction day and time. When that auction time expires, the auction is closed (see the Close Auction use case).
If a Seller has previously listed an item for auction, he/she may re-list the same item instead of re-entering the same information from scratch.
The seller can use the credit card information that is stored with her/his account or input credit card information that will be used with this auction only.
The User must be signed in in order to create an auction. See the Sign In use case.

If the User has any pending payment notices due, a message is displayed to the User informing him/her that an outstanding balance is due, and the User is not permitted to create an auction. The user must go into his/her account and modify the credit card information (see the Manage Account use case).

5.1.6 Sign In

Brief Description: The Sign In use case is where the User identifies him/herself to the system.
If the User already has an account in the system, the User supplies a username and password to authenticate him/herself. If the User cannot be authenticated, the User is not signed in.
For auditing purposes, both successful and unsuccessful sign in attempts are logged. For more information on how these logs are used, see the Review Sign In Logs use case.
If the User does not have an account in the system, the User is given the opportunity to create a new user account. See the Create Account use case. Once the account is created, the User is signed on.
If the user has any pending payment notices, a message is displayed to the User, reminding the User that payment for the notices must be made (i.e., new credit card information must be entered) before the User can participate in any auction (as either the Buyer or the Seller. Credit card information can be entered via the Manage Account use case.

6. Logical View

This section describes the logical structure of the system. It starts from the overview of the architecture and then presents its key structural and behavioral elements.

6.1 Architecture Overview

There are two dominant structures in the application:

1. Logical decomposition of the system into business components and then into layers inside the components, and
2. The structure of the use case realizations derived from model templates of two architectural mechanisms.

These are briefly introduced below and discussed in more detail in the Architecturally-Significant Model Elements section below.

A *business component* "is a software implementation of an autonomous business concept or business process. It consists of all the software artifacts necessary to represent, implement, and deploy a given business concept as an autonomous, reusable element of a large application". From the logical design point of view it is a large-grained system element that crosses all layers. It provides a complete set of functionality and hence can be developed and deployed as an autonomous unit.

A model template, which is a modeling concept, is a collection of model elements designed to be imported into a solution with substitutions. A model template contains model template element(s) and is not intended to be used directly in the solution, but is intended to be "unfolded" or "instantiated" into a part of the solution. In UML model templates are represented as packages stereotyped as «framework».

A mechanism is a pre-designed solution to a common problem that has to be addressed repeatedly in an application (mechanisms are often derived from patterns). Mechanisms are captured as model templates, hence they are represented in «framework» packages. The reference application makes extensive use of two mechanisms:

1. Presentation Request Processing, and
2. Session EJB Access.

These mechanisms are discussed in some detail in the next section.

6.2 Architecturally-Significant Model Elements

6.2.1 Business Components

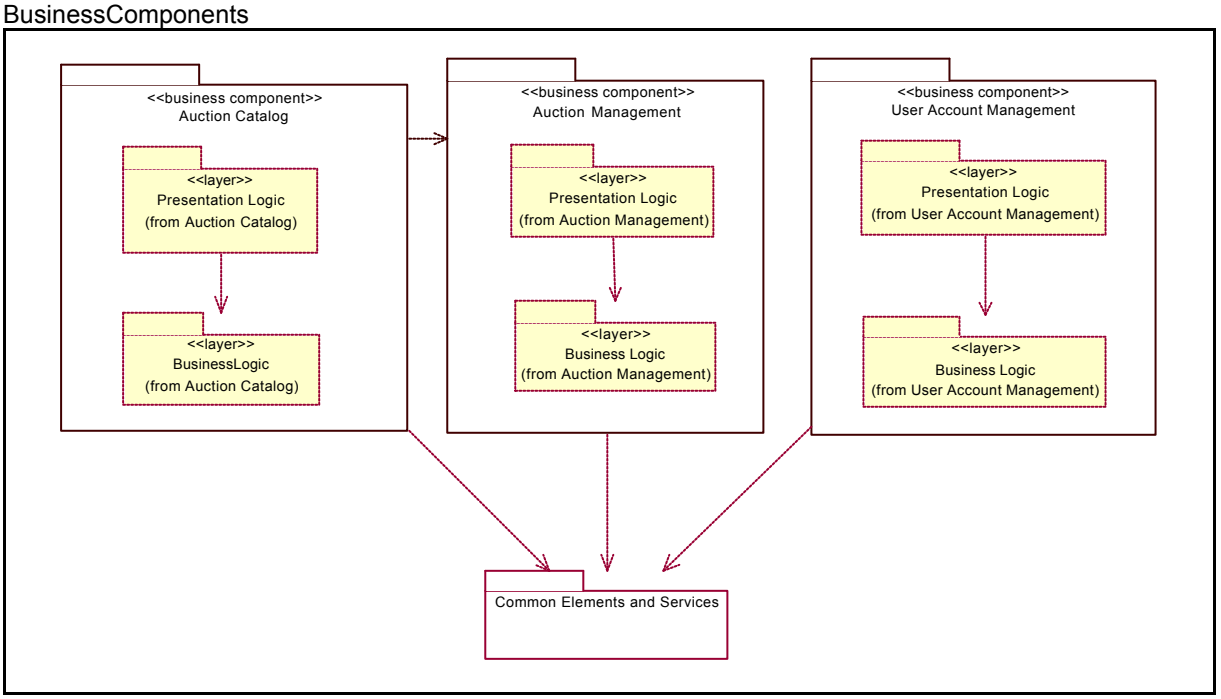
The PearlCircle Online Auction has been decomposed into three *business components* and one common elements and services component. Each of the business components is further divided into three layers: (1) Presentation Logic (2) Business Logic and (3) Integration Logic. In other words, the architecture decomposes the systems along two dimensions

1. The first dimension is along the system functionality lines
2. The second dimension is along the commonly recognized layers separating three kinds of concerns
    a. Presentation concerns, or how to handle communication with the user and control his access to the system services and resources
    b. Business concerns, or how to organize the system elements that perform business and system services functions, and
    c. Integration concerns, or how to connect the system elements with the persistency mechanism, other systems, physical devices, etc.

The result is a matrix-like structure of the system where each design element belongs to a business component (or the common elements and service component) and a layer within that component/service.

The reference application business components are shown in the diagram below.

BusinessComponents



The businesses components and their responsibilities are as follows:

1. User Account Management. This component is responsible for creation, management and deletion of user accounts. In particular, the elements of this businesses component collaborate to realize the user-account-related use cases: Create Account and Manage Account. The component elements also realize the Sign In use cases.
2. Auction Management. This component is responsible for creation, management and closing of auctions. Its elements realize the Create Auction , Close Auction and Bid on Item use cases.
3. Auction Catalog. This business component is responsible for browsing the auction catalog; the Browse Auction Catalog use case.

6.2.2 Mechanisms

The reference application makes extensive use of two mechanisms:

3. Presentation Request Processing, and
4. Session EJB Access

Described in the package Logical View:Design Model:Mechanisms.

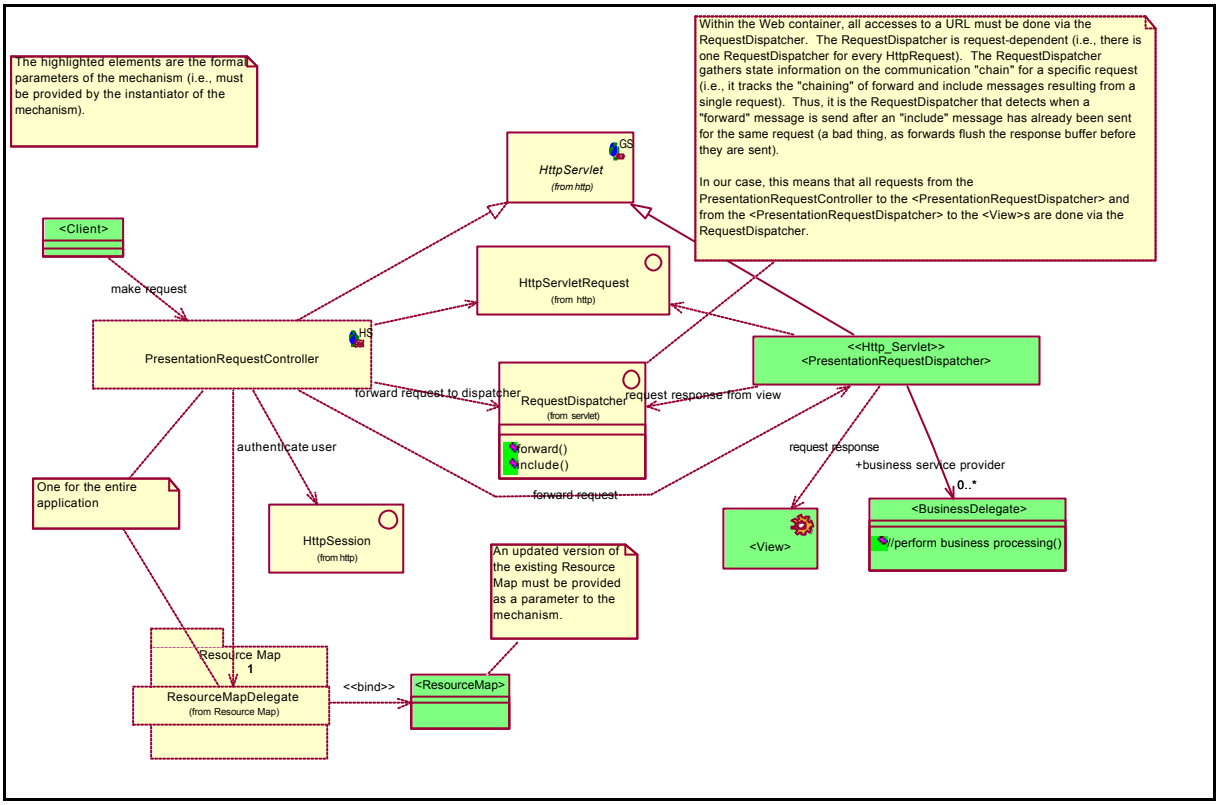
Both of the mechanisms have been constructed by combining implementation strategies of a few Sun J2EE patterns. The first mechanism, Presentation Request Processing, has been derived by combining implementation strategies of three Sun J2EE patterns:

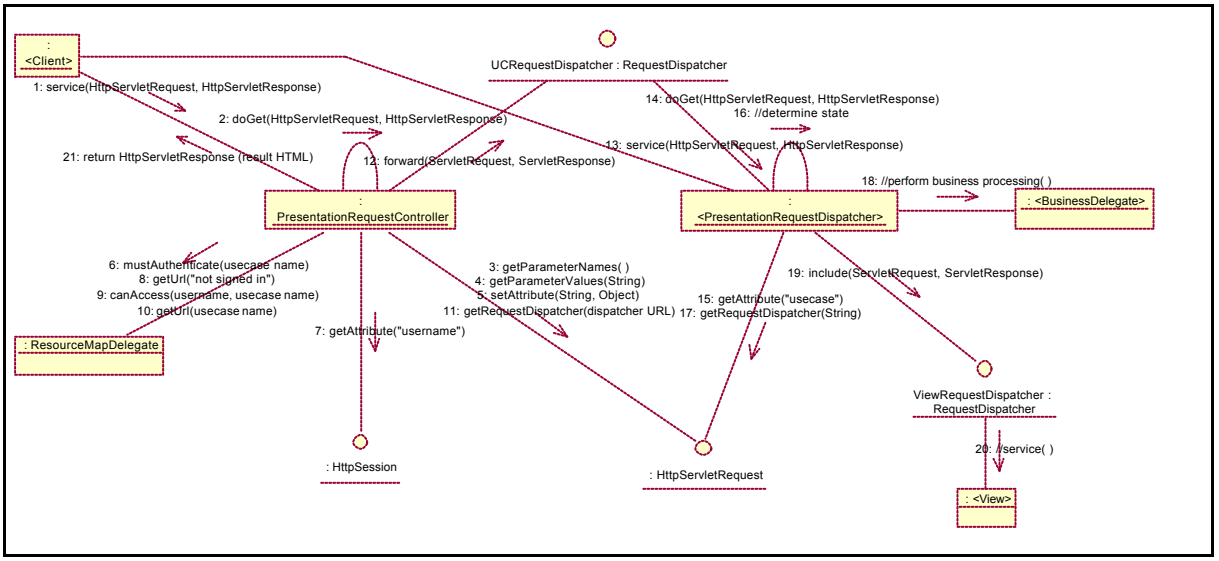
1. Front Controller
2. Service to Worker and
3. Business Delegate

with simple security services.

The participant diagram and the collaboration diagram of the mechanism are show below.

Presentation Request Processing





The elements colored yellow on the diagram are concrete. The placeholders (parameters) are shown in green. The following key architecture decisions have been captured by the Presentation Request Processing mechanisms:

- All user requests are handled by a single PresentationRequestController (the front controller of the

application)

- The front controller inspects each request for consistency and verifies if the user needs to be authenticated and authorized (the ResourceMapDelegate is the front controller's intelligent proxy to the security services).
- If the user is authenticated and authorized, the PresentationRequestController forwards the request to an appropriate dispatcher
- A dispatcher handles user interactions for a specific use case (there is one dispatcher for each use case realization). Dispatchers "understand" the use case flow and coordinate use case-related interactions with the user.
- The dispatchers do not produce any user output. Instead, they call views (see <View> parameter in the mechanism) to create HTML or XML or other typed of a stream that is returned to the user's device.
- The dispatchers also don't do "business logic". Instead, they use business delegates (see <Business Delegate> parameter in the mechanism) as smart proxies to the application server services
- The front controller and the dispatchers are implemented as Servlets that forward URL requests to each other. Hence, they can be placed in a different process and/or on different machines, which can be desirable for security reasons.
- Views are implemented as JSPs for ease of development.
- Business delegates are implemented as Java Beans that run in the thread of the calling delegates.

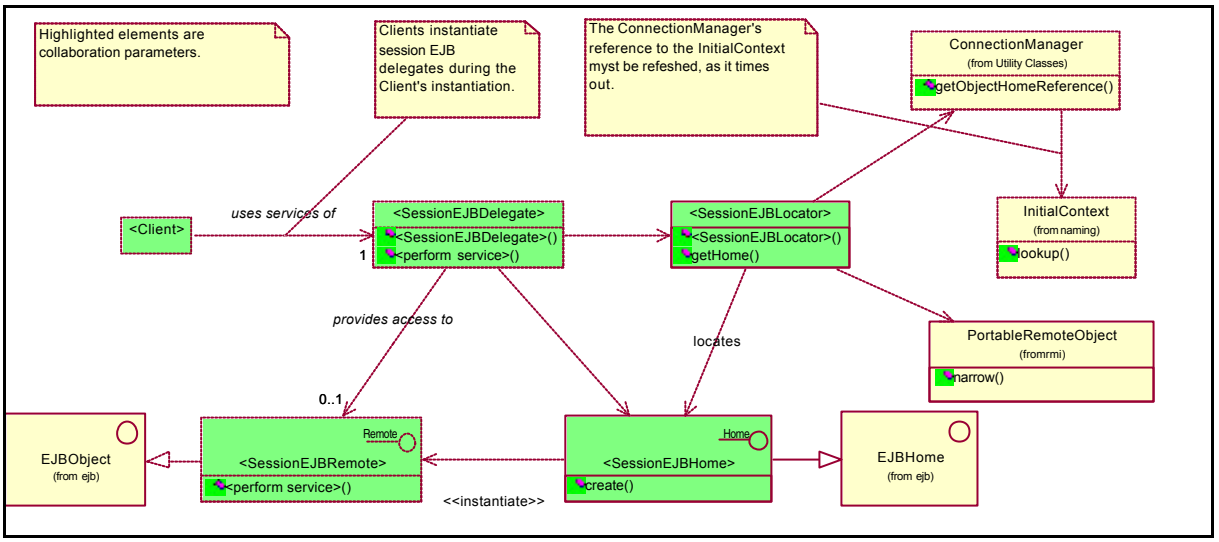
The resource access table used by the authorization component and the front controller to map requests to dispatchers is stored in an XML file and internalized on system startup (see <Resource Map> parameter in the mechanism).

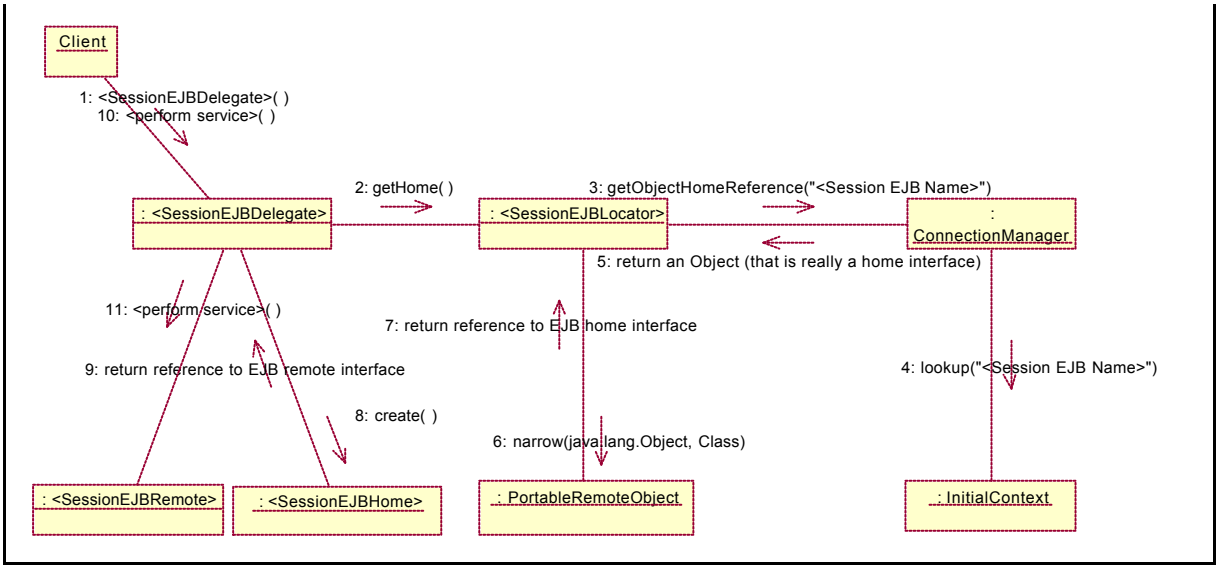
The second mechanism, Session EJB Access, combines implementation strategies of two Sun J2EE patterns:

1. Business Delegate and
2. Service Locator.

The participant diagram and the collaboration diagram of the mechanism are shown below.

Session EJB Access





The mechanism is a "blueprint" for the organization of access to the application server components. The architecture does not allow any presentation layer components (dispatchers in particular) to communicate directly with entity EJBs. Hence, the only beans that can be accessed remotely are session EJBs and the mechanism is used for that. The mechanism captures the following architecture decisions:

- All business service components are implemented as Session EJBs or have Session EJB façades
- A client (in most cases it is a dispatcher) that requires access to a business service component creates an instance of a session EJB delegate (see <SessionEJBDelegate parameter, which is usually bound to the same concrete elements as the respective business delegate in the related PresentationRequestController

mechanism[1])

- The constructor of the session EJB delegate contacts a session EJB locator (see <SessionEJBLocator> parameter) which returns the session EJB home interface that in turn provides the delegate with the remote interface to the service.

As a consequence of using the two mechanism of the framework, the application has a distinct separation boundary between the presentation logic and the business logic. That boundary is the business delegates that are smart proxies to the business services of the application. This separation is very desirable for a few of reasons:

- It provides an explicit contract between designers and developers of the presentation tier and the business tier of the system
- It allows the business tier to change independently from the presentation tier
- It allows for concurrent development of the two tiers.

6.2.3 Common Elements & Services

Common Elements and Services is a grouping (a package) of support elements and services that don't belong to any of the business components, but belong to the framework that underlines the application. The package contains the following elements and services:

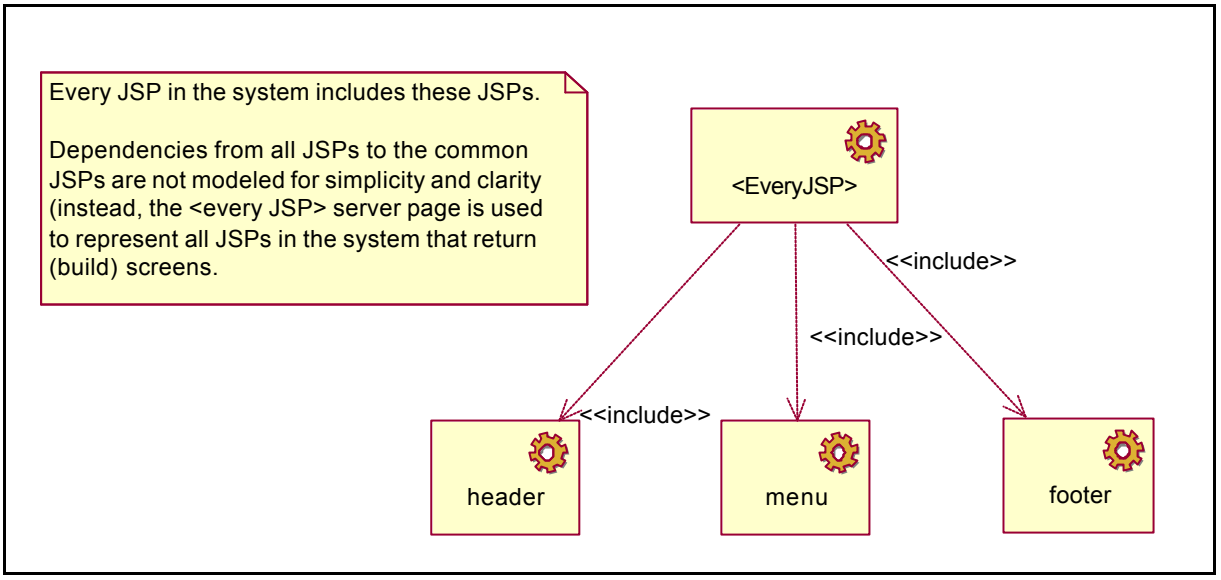
- Base Presentation Elements
- Credit Bureau Service
- Email Service
- Presentation Request Processing
- Sign In Log
- Utility Classes, and
- XML Parser

6.2.3.1 Base Presentation Element

The package contains elements of a very simple pattern which states, that every JSP producing a screen includes a standard header, menu, and a footer. As the comment in the diagram below explains, the pattern is not explicitly expended in the design model for all screen-building JSPs, but it is used at the code level.

Base Presentation Elements

[1] In use case realization the two mechanisms are combined. The combination is done by biniding the same JaveBean to the <BusinessDelegate> parameter in the PresentationRequestprocessin mechanism and to <SessionEJBDelegate> parameter in the SessionEJBAccess mechanism.



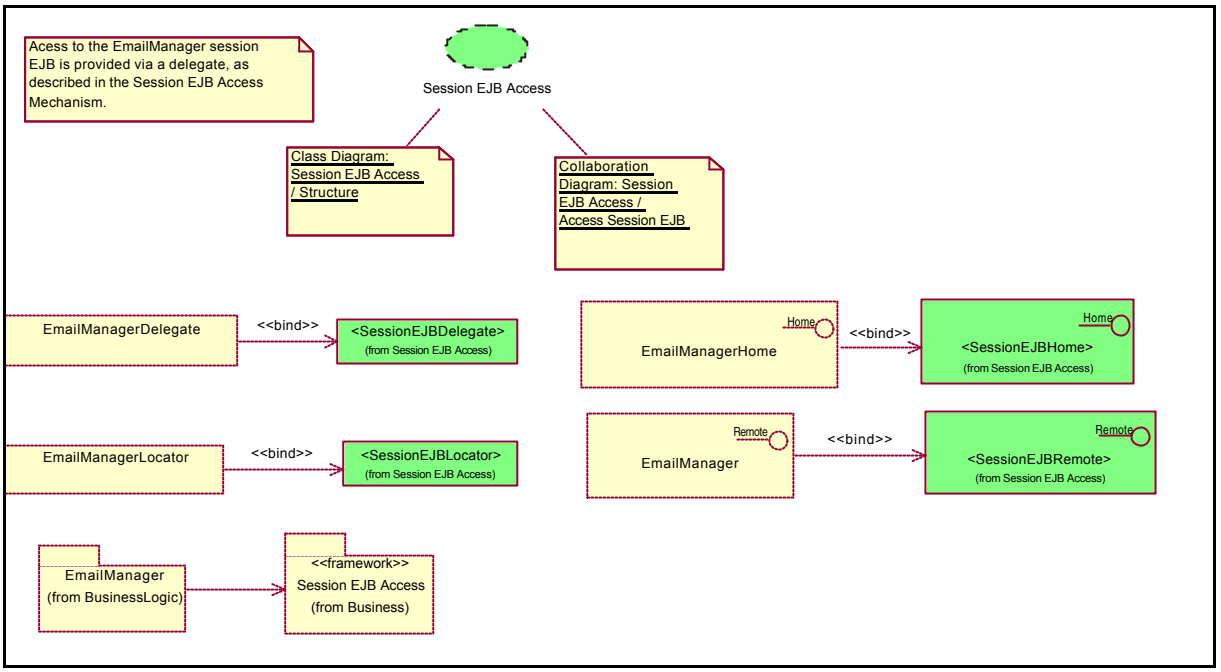
6.2.3.2 Credit Bureau Services

The Credit Bureau Services is represented by a stubbed-out delegate of two services (1) verification of credit cards, and (2) credit transaction for the fee charged to the sellers who sell their items.

6.2.3.3 Email Service

The Email Service supports the capability of the system to send email messages to the user of the auction system. The service is a small, service business component that only has a business layer. The component has no user interface, as it is intended to be used only via its delegate. The Email Service is also an example of an instantiation of the Session EJB Access mechanism. The first diagram below shows the bindings applied to the mechanism, while the second diagram shows the resulting view of the participating classes.

Email Service: Mechanism Binding and Instance



The Email Service is also an example of how we represent EJBs in the PearlCircle OnLine Auction system. We follow an approach somewhat similar to the JSR-26 recommended approach. The EJBs are described in a separate package that has the same name as the EJB itself and is stereotyped «SessionEJB» or «EntityEJB». The remote interface of the respective bean and its home are defined outside of the package.

6.2.3.4 Presentation Request Processing

The package contains the concrete elements of the front-end of the application framework. In particular, it contain:

- The Presentation Request Controller (see the <Common Elements and Services:Presentation Request Processing:Presentation Logic: <<Http_Servlet>> PresentationRequestController> servlet) and
- The Resource Map service (see the < Common Elements and Services:Presentation Request Processing:Business Logic:ResourceMap> package)

The Presentation Request Controller is a singleton defined in the Presentation Request Processing mechanism. In other words, it "comes with" the mechanism as a concrete element.

The Resource Map is a service used by the Presentation Request Controller to:

- Map the first user request parameter to a use case (and hence to the use case dispatcher)
- Verify if the user needs to be authenticated, and
- If the user is authenticated, verify if the user needs to be authorized to use the particular system function (the use case).

The Resource Map service is structurally very similar to the Email Service, as it is an instantiation of the Session EJB Access mechanism (see the biding and participants class diagrams in < Common Elements and Services:Presentation Request Processing:Business Logic:ResourceMap> package). The service is comprised of

- A Session EJB (<<SessionEJB>> ResourceMapEJB) with its home and remote interface
- A delegate (ResourceMapDelegate Java Bean), and
- A locator (ResourceMapLocator Java Bean).

6.2.3.5 Sign In Log

Sign In Log has two elements:

- An Entity Bean (see <Common Elements and Services:Sign In Log:Business Logic:SignInLogger: <<EntityEJB>> SignInLoggerEJB> package) that is used to store the sign-in attempts/actions of the users, and
- A sign-in service implemented as a Session Bean (see <Common Elements and Services:Sign In Log:Business Logic: SignInLoggerManager> package).

The SignInLoggerManager is once again an instantiation of the Session EJB Access mechanism and hence can be accessed via a delegate. The two key attributes of a sign-in attempt are userId and actionTime.

6.2.3.6 Utility Classes and XML Parser

The Utility Classes and the XML Parser packages contain support elements. As the name implies, the XML Parser contain a utility Java class that loads and parses XML files.

The Utility Classes package contains:

- Value classes (classes used to pass state of Entity Beans by value)
- Exception classes, and
- Helper classes used by, for example, the locators to encapsulate connection with the EJB Container.

6.3 User-Experience Model

The user experience model captured in the <Logical View:User-Experience Model> package is a very important part of the specification of the PearlCircle Online Auction application. It captures the screens presented to the user and the transitions between screens resulting from the user-generated events such as clicking on a URL or a submit button of a form. Descriptions of the screens also contain descriptions of the dynamic content that the system must produce and display to the user.

The user experience model consists of:

- Navigational maps, showing allowable transitions (navigations) between system screens
- Storyboards describing navigations inside the scope of individual use cases, and
- Static "lists" of user experience model elements.

These three elements are briefly described in the following subsections.

Complementing the user experience model are mappings between its elements and the design elements that contribute to their "creation". In our online auction system the screens and forms of the user experience model are mapped to the JSPs and static pages that provide the HTML for the screens. This mapping is a contract between the designer of the systems interface and the designer of the business logic of the system.

6.3.1 Architecturally Significant Navigation Map(s)

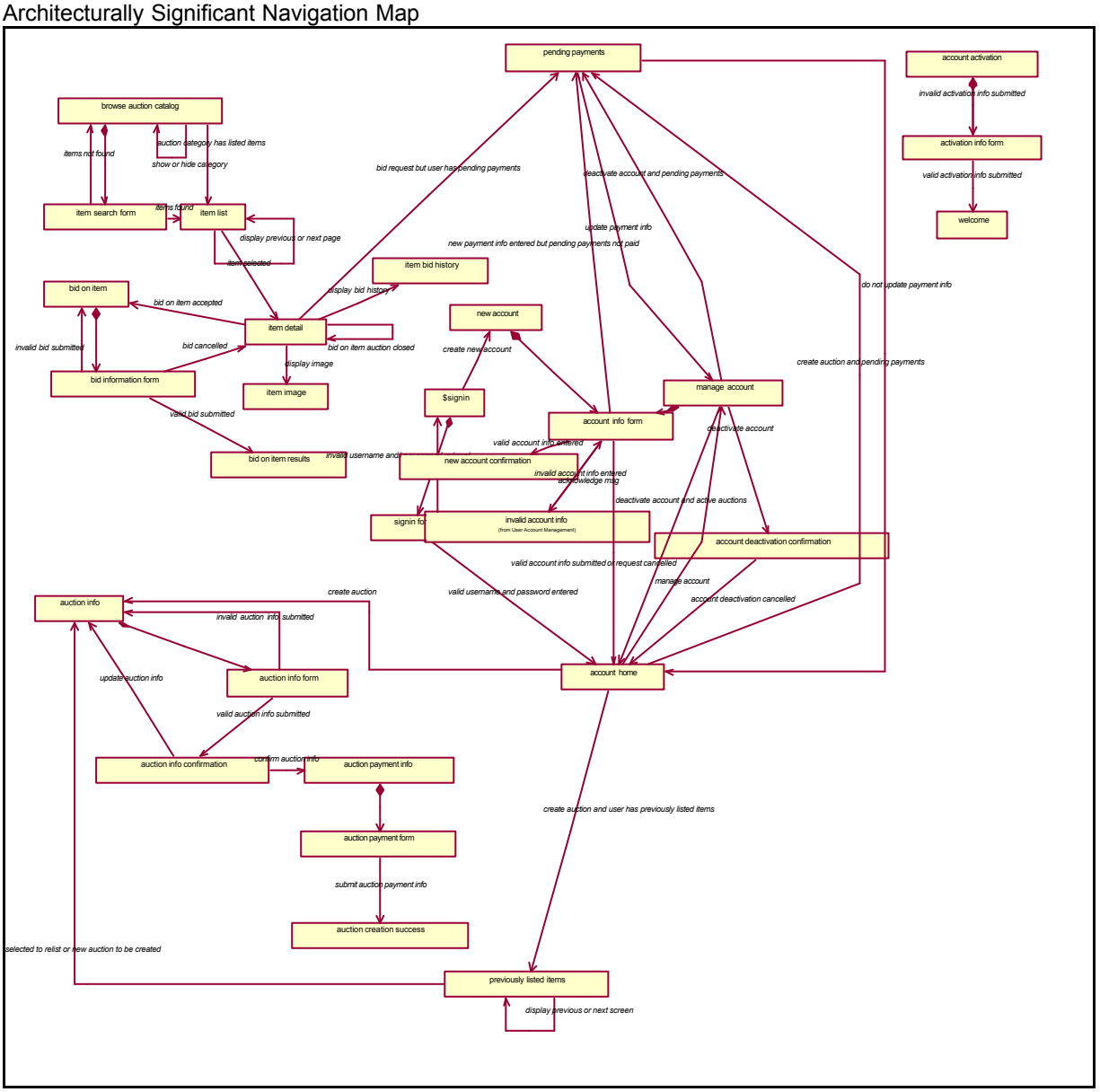
The diagram below shows the PearlCircle Online Auction navigation map for the architecturally significant use cases. This means, that the diagram shows all the screens and screen-to-screen transitions for the key use cases of the system.

The classes on the diagram represent (1) screens (stereotype «screen»), (2) forms (stereotype «form») or (3) value objects whose content is displayed on the screens or in the forms.(no stereotype).

The relationships between classes represent transitions between screens initiated by the user's actions. Transitions

can be paired with the methods on the forms or on the screens, where the methods represent allowable user-generated events.

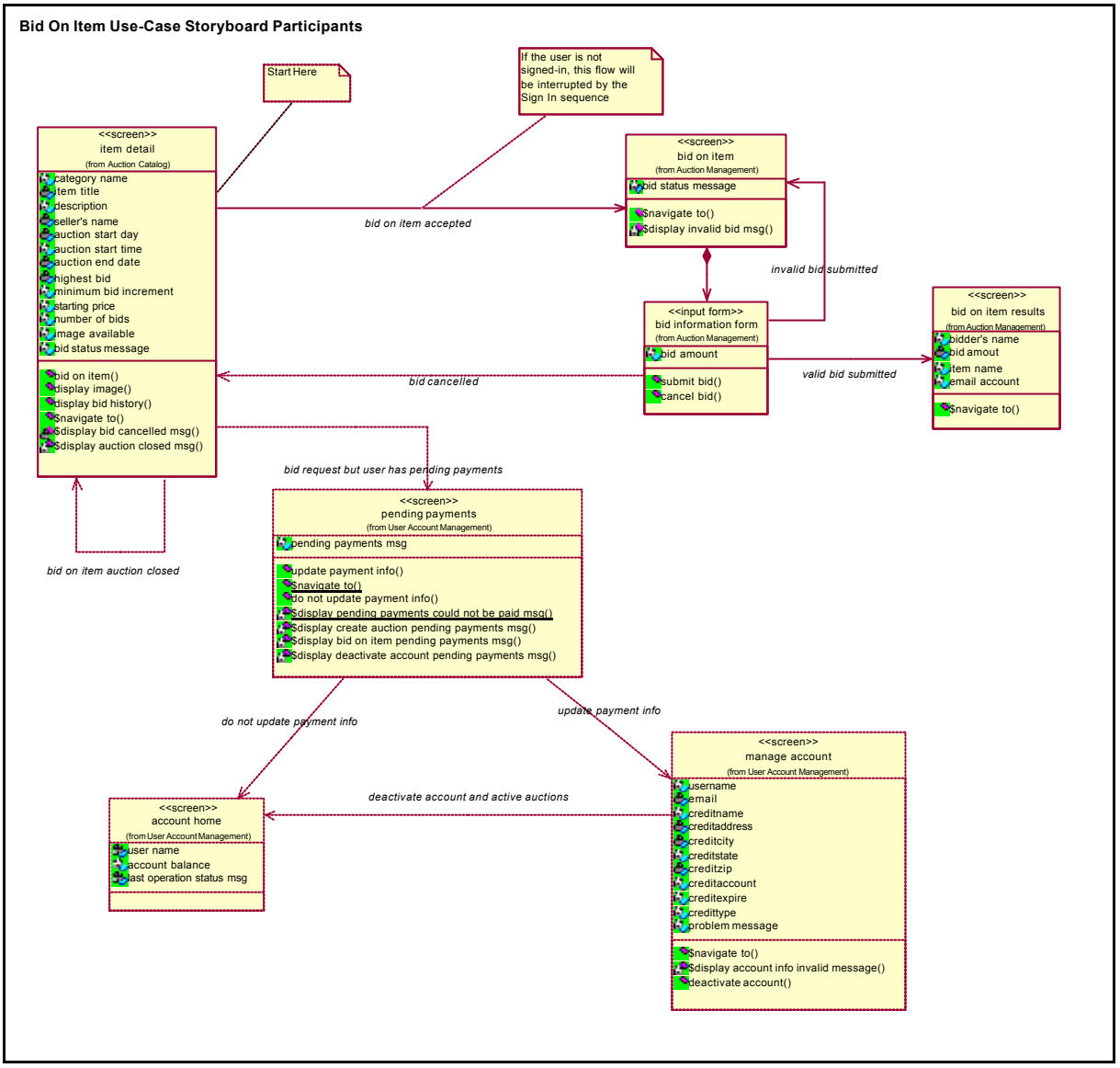
Architecturally Significant Navigation Map

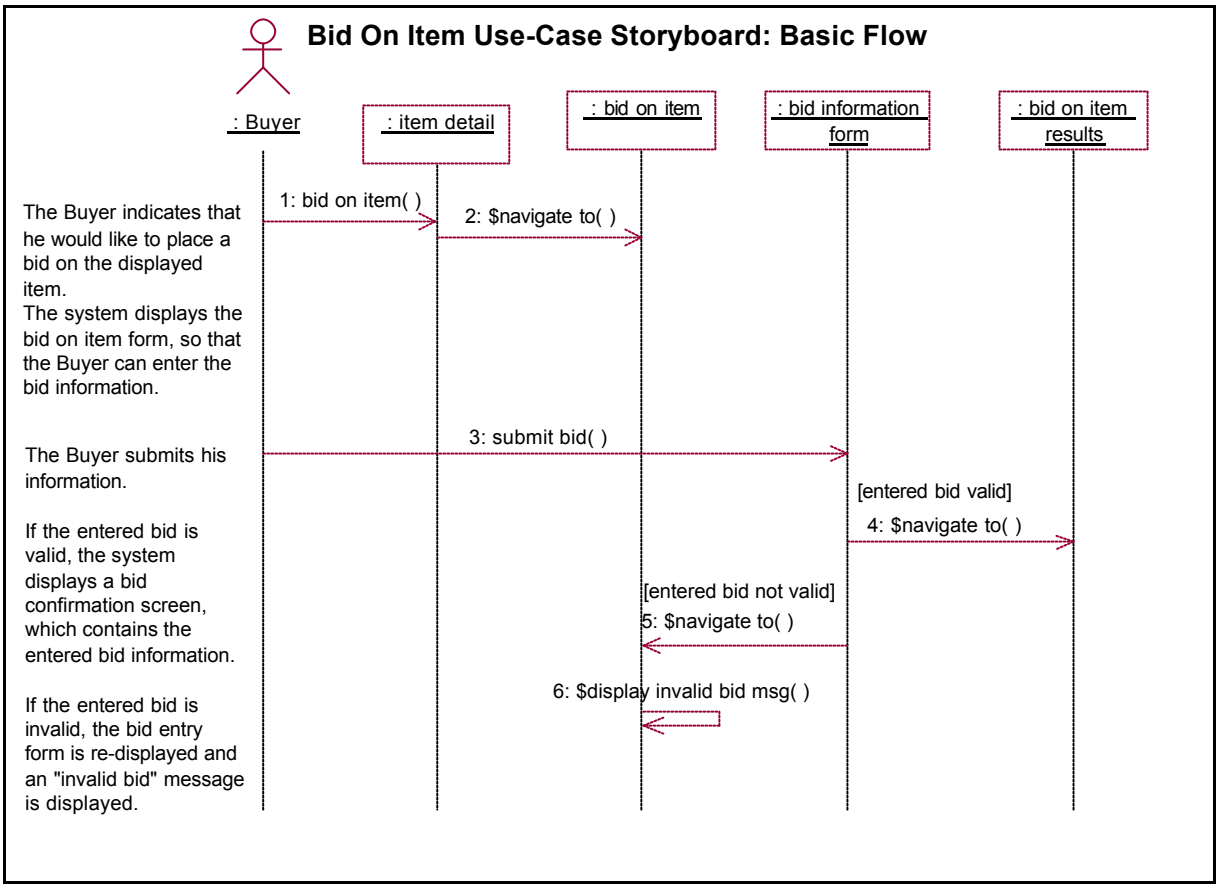


6.3.2 *Architecturally Significant Use-Case Storyboards*

Each use case has its own so-called storyboard. A storyboard is a collaboration of user experience model elements that participate in the realization of a use case. In other words, a storyboard shows system behavior from the user experience model perspective. Below, as an example, we show the participants and the basic flow of the Bid on Item use case storyboard.

Bid on Item Use Case Storyboard Participants and Basic Flow


Bid On Item Use-Case Storyboard Participants
Start Here
If the user is not signed-in, this flow will be interrupted by the Sign In sequence
<<screen>>
item detail
(from Auction Catalog)
category name
item title
description
seller's name
auction start day
auction start time
auction end date
highest bid
minimum bid increment
starting price
number of bids
image available
bid status message
bid on item()
display image()
display bid history()
$navigate to()
$display bid cancelled msg()
$display auction closed msg()
bid on item accepted
<<screen>>
bid on item
(from Auction Management)
bid status message
$navigate to()
$display invalid bid msg()
invalid bid submitted
<<input form>>
bid information form
(from Auction Management)
bid amount
submit bid()
cancel bid()
bid cancelled
valid bid submitted
<<screen>>
bid on item results
(from Auction Management)
bidder's name
bid amout
item name
email account
$navigate to()
bid on item auction closed
bid request but user has pending payments
<<screen>>
pending payments
(from User Account Management)
pending payments msg
update payment info()
$navigate to()
do not update payment info()
$display pending payments could not be paid msg()
$display create auction pending payments msg()
$display bid on item pending payments msg()
$display deactivate account pending payments msg()
do not update payment info
update payment info
<<screen>>
manage account
(from User Account Management)
username
email
creditname
creditaddress
creditcity
creditstate
creditzip
creditaccount
creditexpire
credittype
problem message
$navigate to()
$display account info invalid message()
deactivate account()
deactivate account and active auctions
<<screen>>
account home
(from User Account Management)
user name
account balance
last operation status msg



The basic flow of the storyboard can be summarized as follows:

- The user is presented with the item detail screen that contains bid-on-item URL
- When the user clicks on the URL s/he will be presented with the bid on item screen for bidding or, if the user owes money to the site, s/he will be presented with the pending payment screen
- The bid on item screen contains a form to make a bid. The user fills out the form to submit the bid
- If the bid increment is correct, the user is presented with the bid on item results screen.

6.3.3 User Experience Model Elements

The model elements of the user experience model (see <Logical View:User-Experience Model:Ux Model Elements> package) have been grouped along the business components boundaries, that is:

- Auction Catalog Ux Model Elements
- Auction Management Ux Model Elements, and
- User Account Management Ux Model Elements.

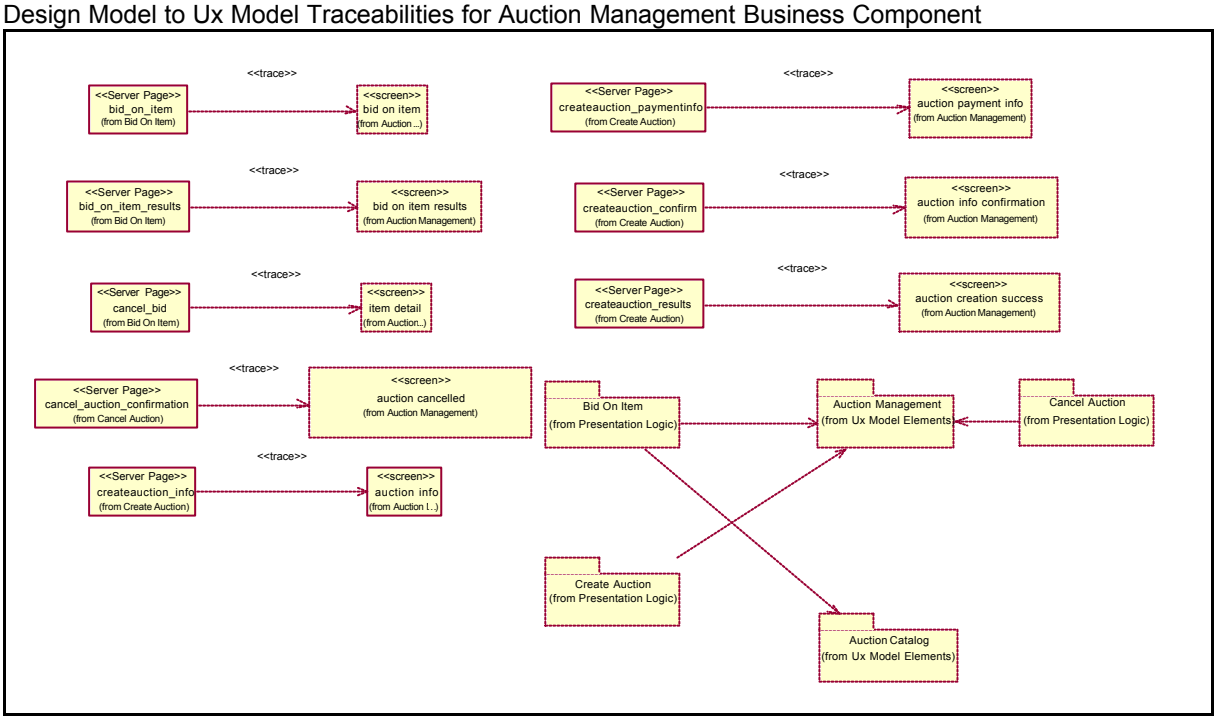
6.3.4 Mappings to Designs Elements

The mappings between Ux model elements and the design elements (in particular the JSPs) are very important. They provide the "bridge" between the "creative" designs of the system and the technical design of the system. The mappings are captured in <Logical View:Design Model:Design Elements:"business component package":Presentation Logic:Design Model to Ux Model Traceabilities> class diagrams.

For example, the diagram below shows the mapping between the design model elements and the Ux model elements in the Auction Management business components. These mappings demonstrate what JSPs produce the screens

presented to the user during execution of auction management use cases. One of the use cases is Bid on Item discussed in 6.3.2.

Design Model to Ux Model Traceabilities for Auction Management Business Component



7. Architecturally-Significant Use-Case Realizations

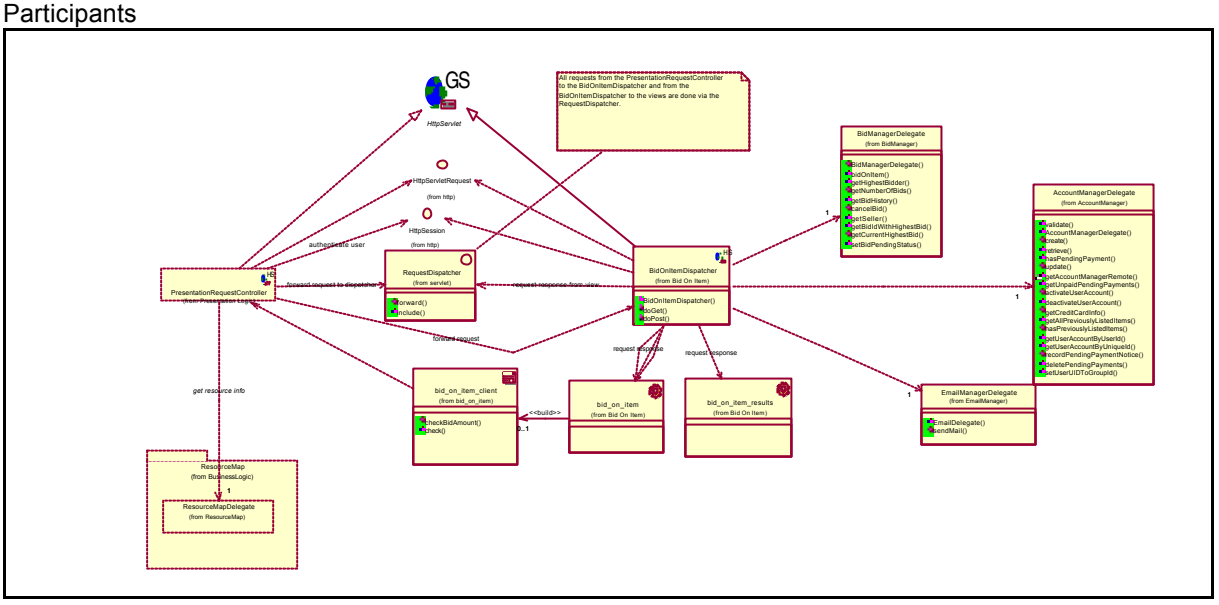
This section shows the architecturally significant use-case realizations. Each use case realization is a collaboration of design elements from the business components and common elements and services. For each use case we show:

- Use-case static view, that is the participants of the use case, and
- Use case dynamic view, that is the collaboration between the use case participants.

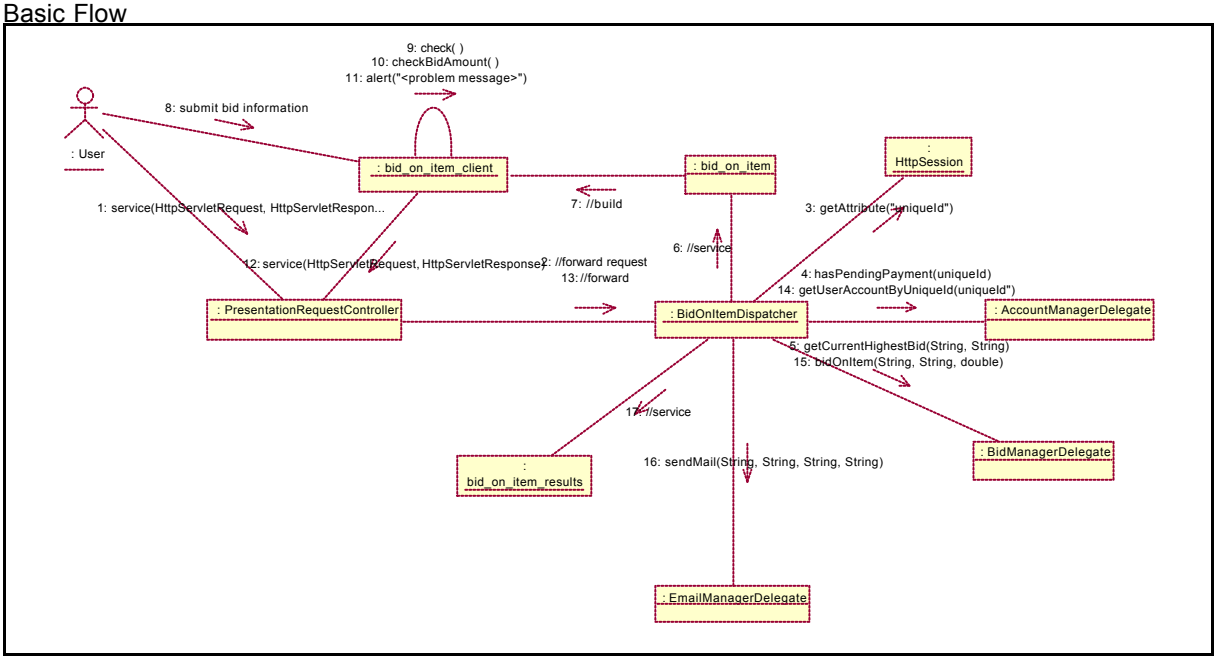
It is important to notice, that all use case realizations have a very similar structure. This is because all use cases have been derived from the two architecture mechanisms described in Paragraph 6.2.2. Complete descriptions of the use cases and scenarios of these realizations can be found in the package <Logical View : Design Model : Design Use-Case Realizations>.

7.1 Bid on Item

Participants

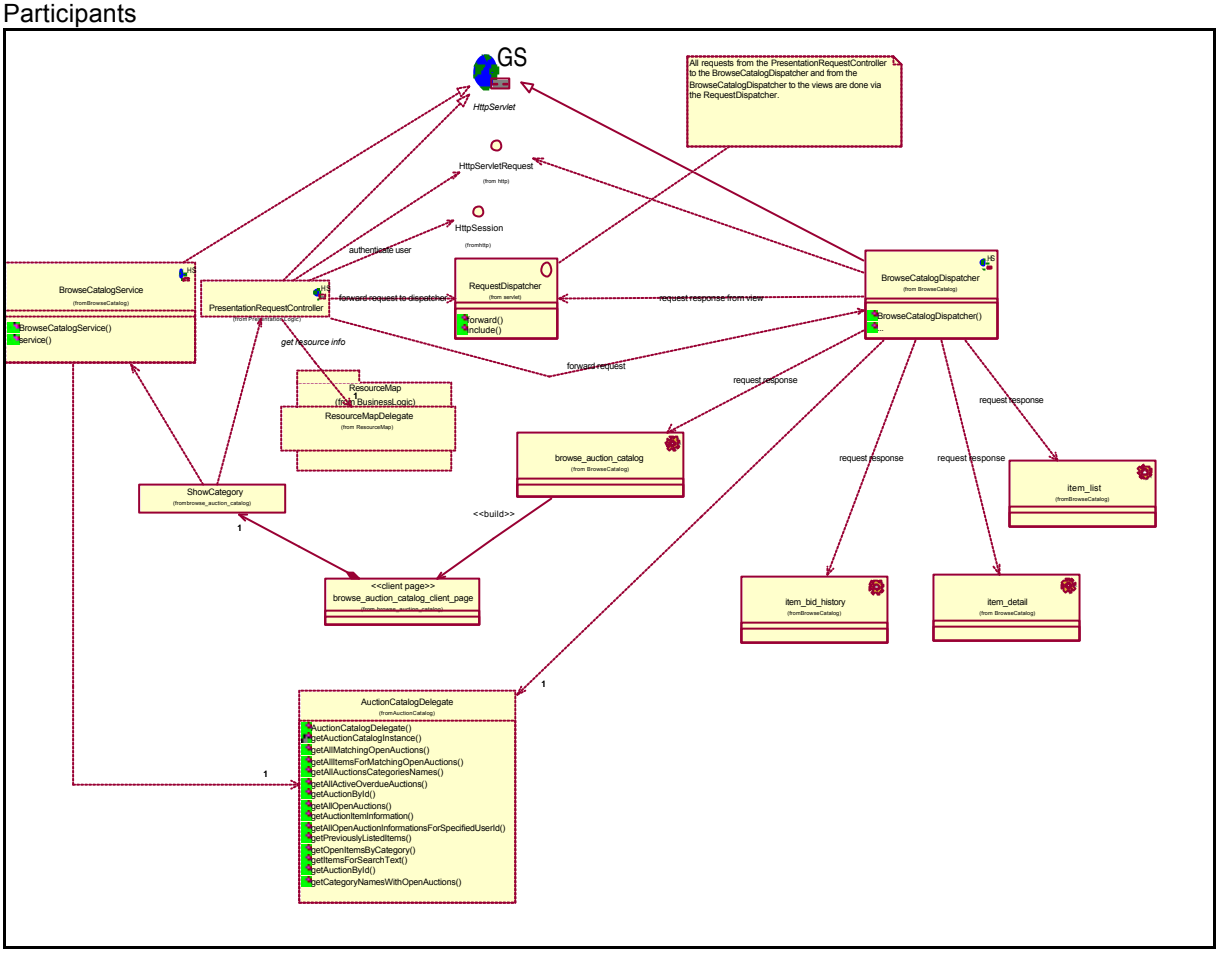


Basic Flow

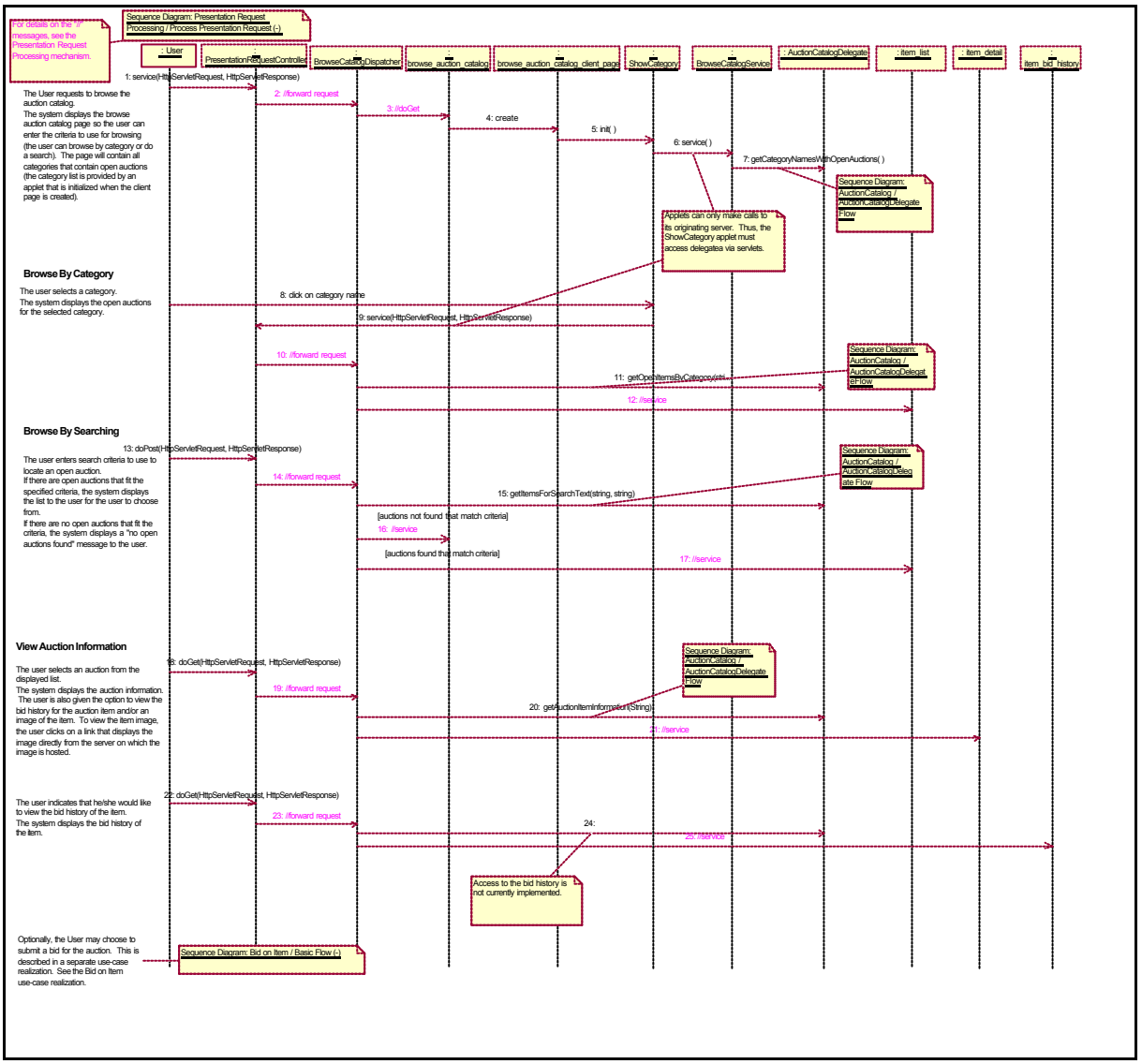


7.2 Browse Auction Catalog

Participants

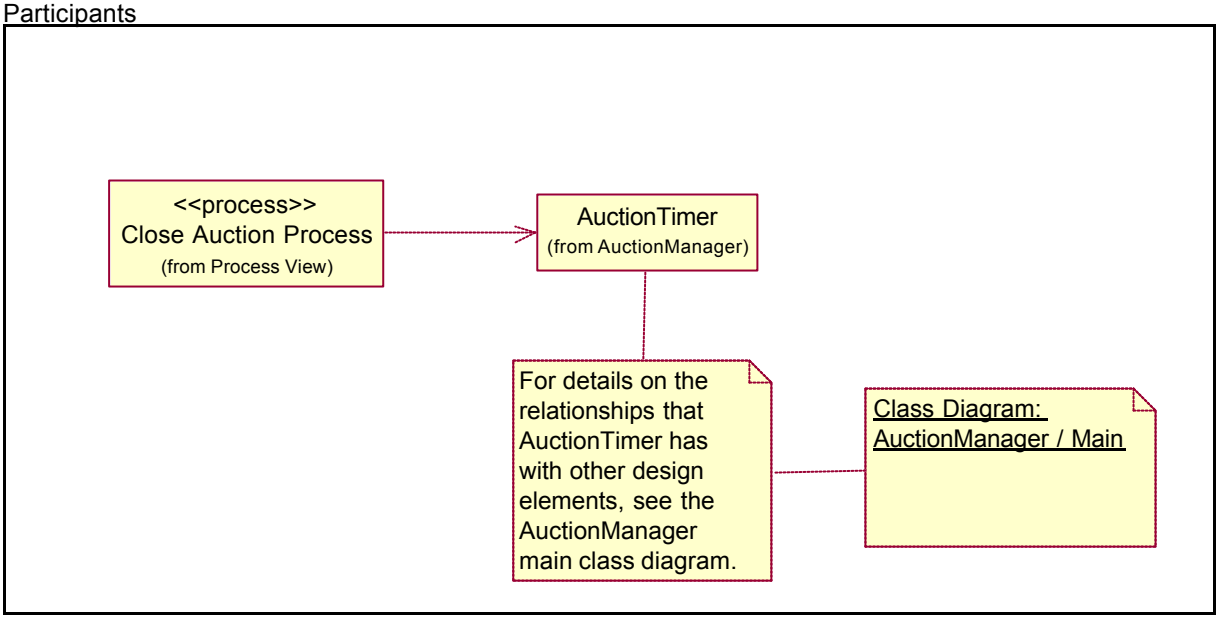


Basic Flow

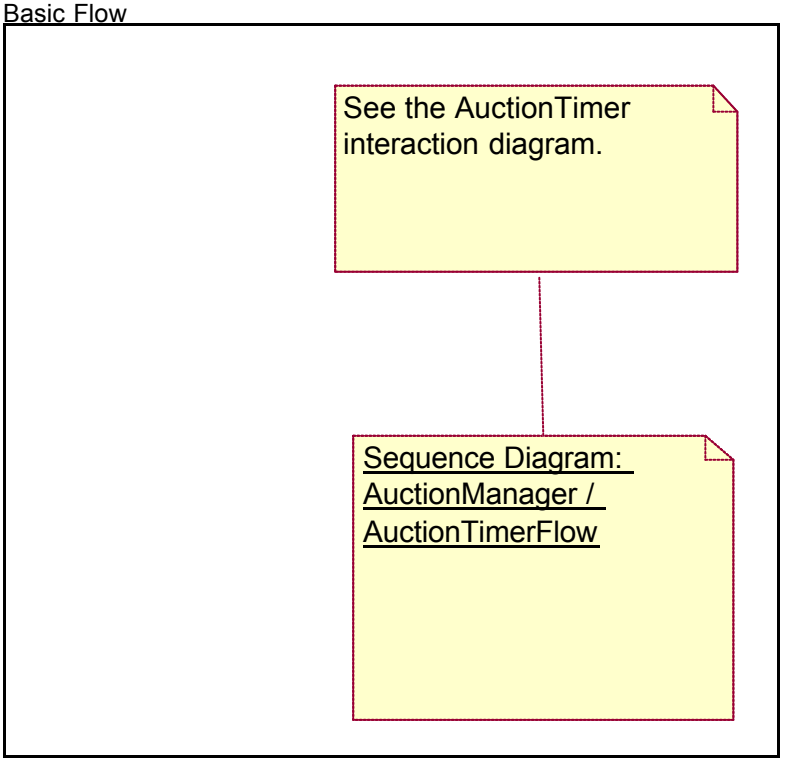

For details on the 'i' messages, see the Presentation Request Processing mechanism.
Sequence Diagram: Presentation Request Processing / Process Presentation Request (-)
: User
PresentationRequestController
BrowseCatalogDispatcher
browse_auction_catalog
browse_auction_catalog_client_page
ShowCategory
BrowseCatalogService
: AuctionCatalogDelegate
: item_list
: item_detail
item_bid_history
1: service(HttpServletRequest, HttpServletResponse)
The User requests to browse the auction catalog.
The system displays the browse auction catalog page so the user can enter the criteria to use for browsing (the user can browse by category or do a search). The page will contain all categories that contain open auctions (the category list is provided by an applet that is initialized when the client page is created).
2: //forward request
3: //doGet
4: create
5: init( )
6: service( )
7: getCategoryNamesWithOpenAuctions( )
Sequence Diagram: AuctionCatalog / AuctionCatalogDelegate Flow
Applets can only make calls to its originating server. Thus, the ShowCategory applet must access delegates via servlets.
Browse By Category
The user selects a category.
The system displays the open auctions for the selected category.
8: click on category name
9: service(HttpServletRequest, HttpServletResponse)
10: //forward request
11: getOpenItemsByCategory(str...
Sequence Diagram: AuctionCatalog / AuctionCatalogDelegate Flow
12: //service
Browse By Searching
13: doPost(HttpServletRequest, HttpServletResponse)
The user enters search criteria to use to locate an open auction.
If there are open auctions that fit the specified criteria, the system displays the list to the user for the user to choose from.
If there are no open auctions that fit the criteria, the system displays a "no open auctions found" message to the user.
14: //forward request
15: getItemsForSearchText(string, string)
Sequence Diagram: AuctionCatalog / AuctionCatalogDelegate Flow
[auctions not found that match criteria]
16: //service
[auctions found that match criteria]
17: //service
View Auction Information
18: doGet(HttpServletRequest, HttpServletResponse)
The user selects an auction from the displayed list.
The system displays the auction information. The user is also given the option to view the bid history for the auction item and/or an image of the item. To view the item image, the user clicks on a link that displays the image directly from the server on which the image is hosted.
19: //forward request
20: getAuctionItemInformation(String)
Sequence Diagram: AuctionCatalog / AuctionCatalogDelegate Flow
21: //service
22: doGet(HttpServletRequest, HttpServletResponse)
The user indicates that he/she would like to view the bid history of the item.
The system displays the bid history of the item.
23: //forward request
24:
25: //service
Access to the bid history is not currently implemented.
Optionally, the User may choose to submit a bid for the auction. This is described in a separate use-case realization. See the Bid on Item use-case realization.
Sequence Diagram: Bid on Item / Basic Flow (-)

7.3 Close Auction

Participants

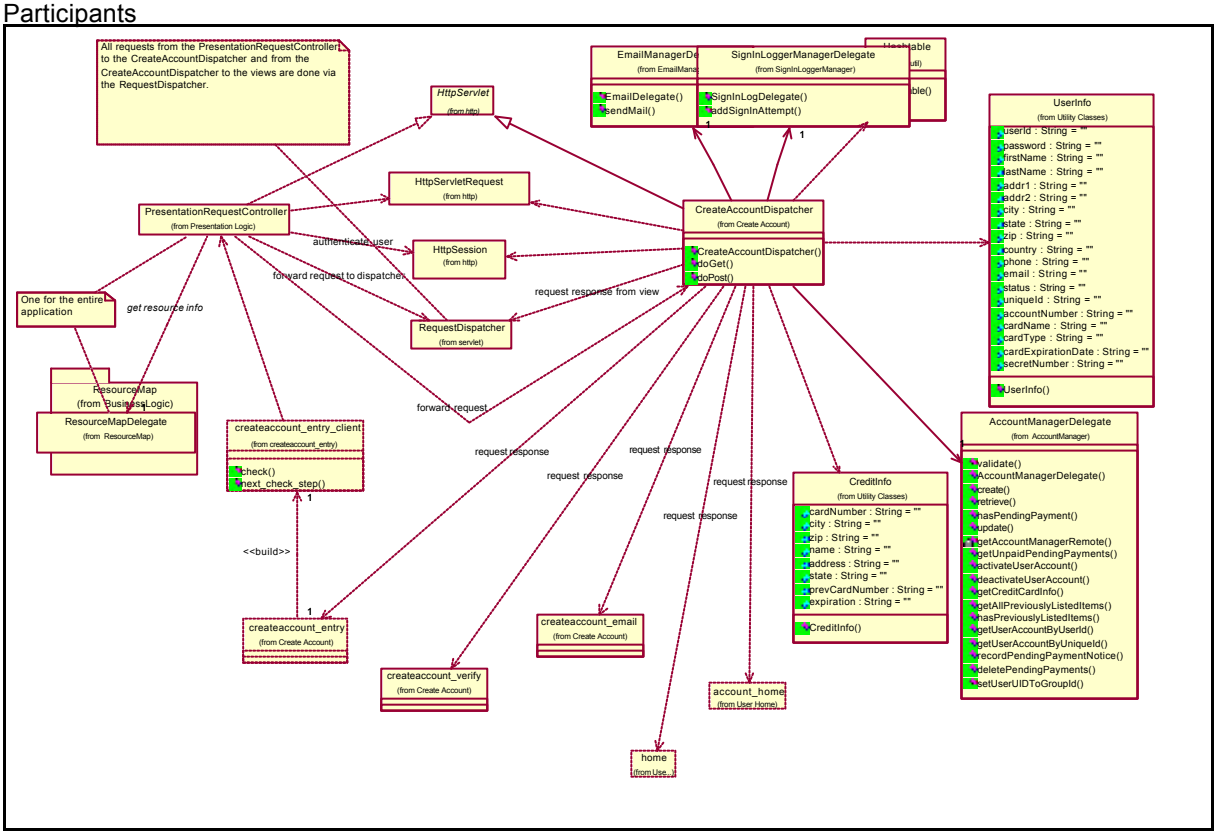


Basic Flow

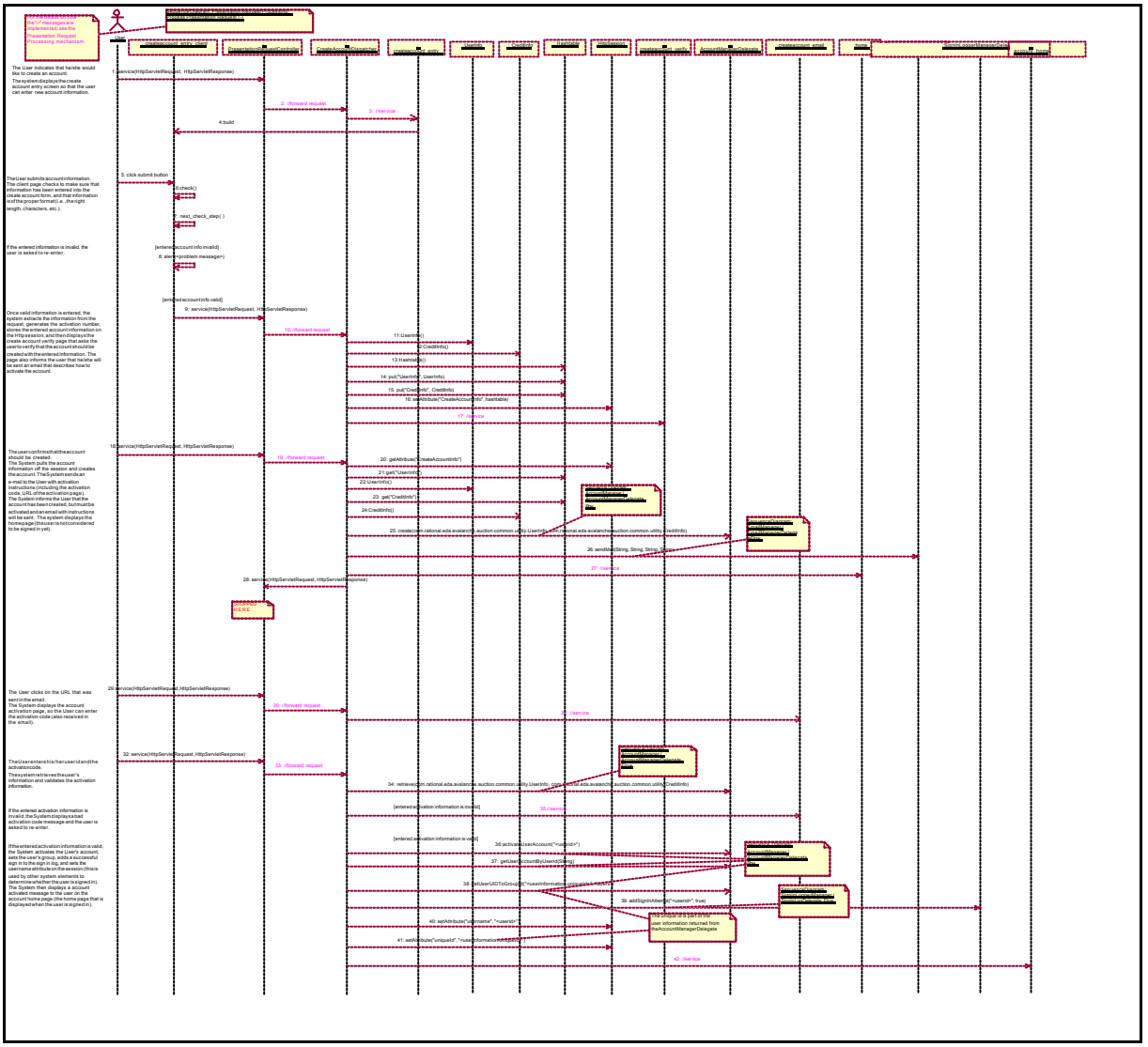


7.4 Create Account

Participants

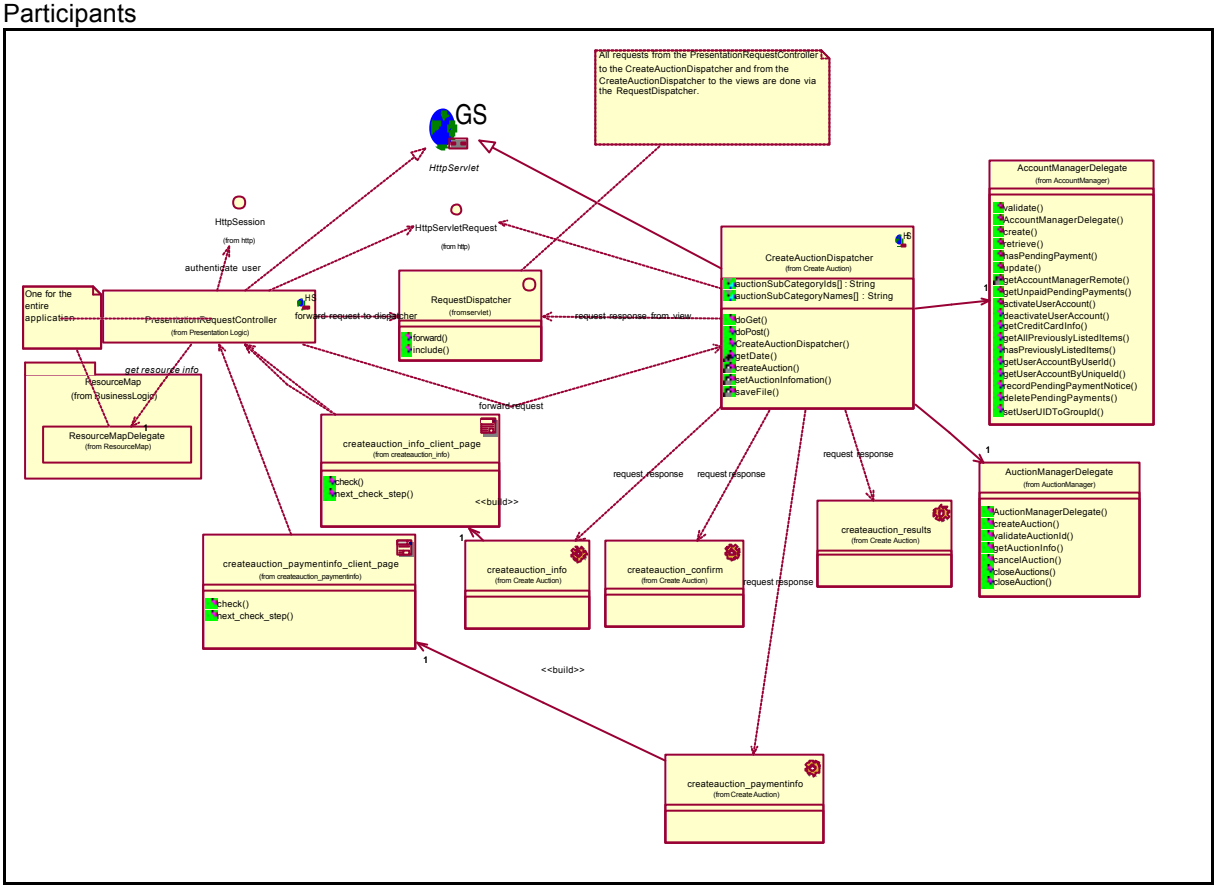


Basic Flow

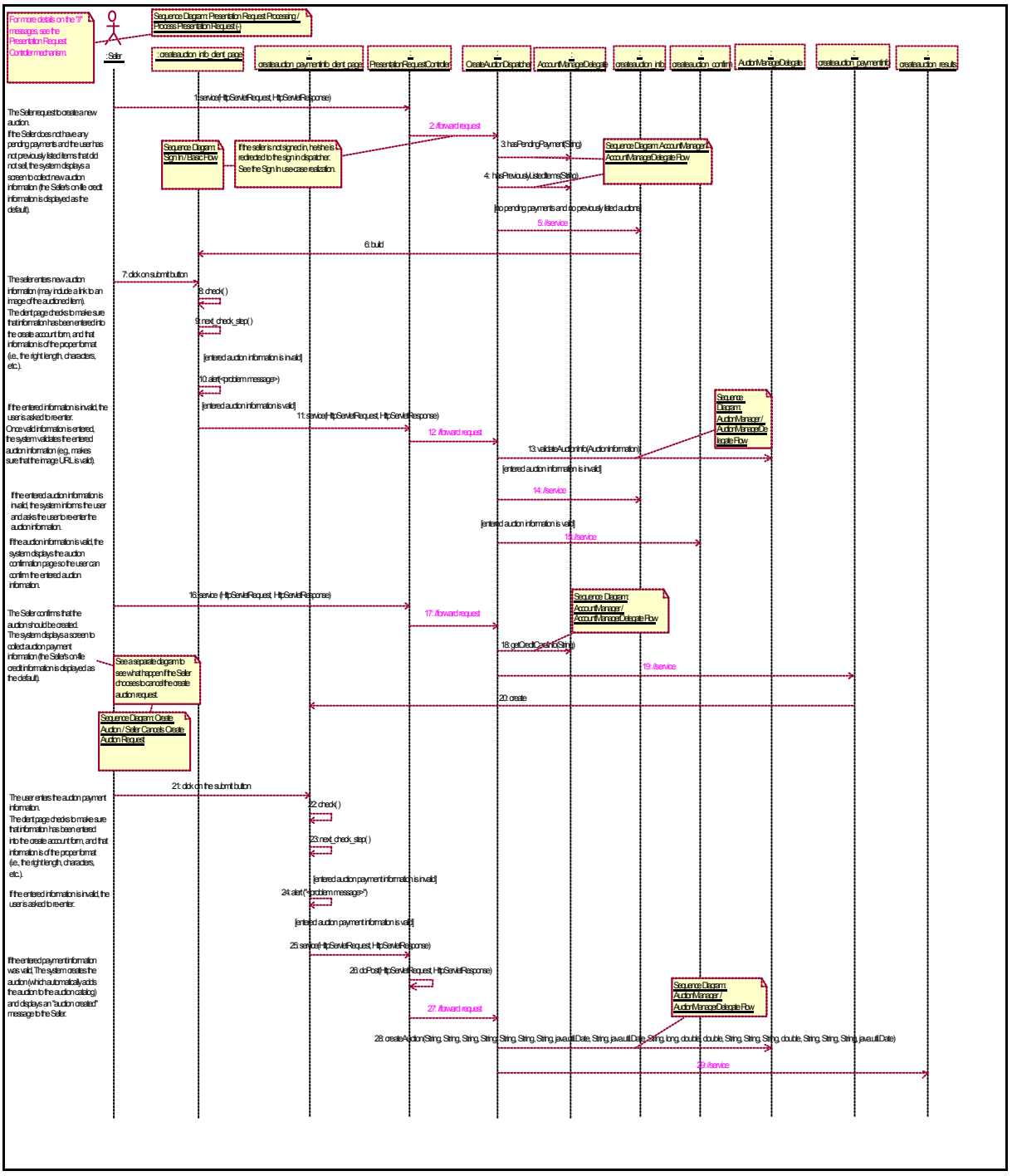


7.5 Create Auction

Participants

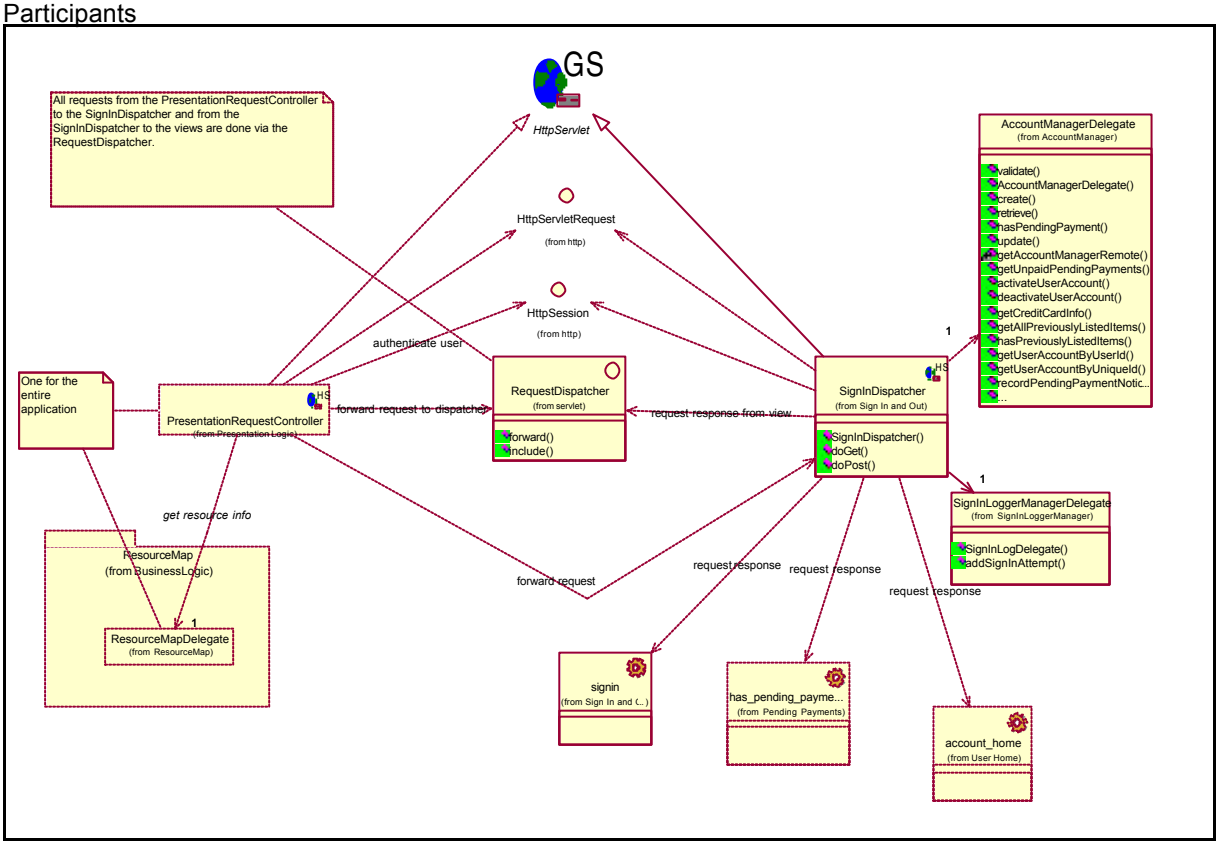


Basic Flow

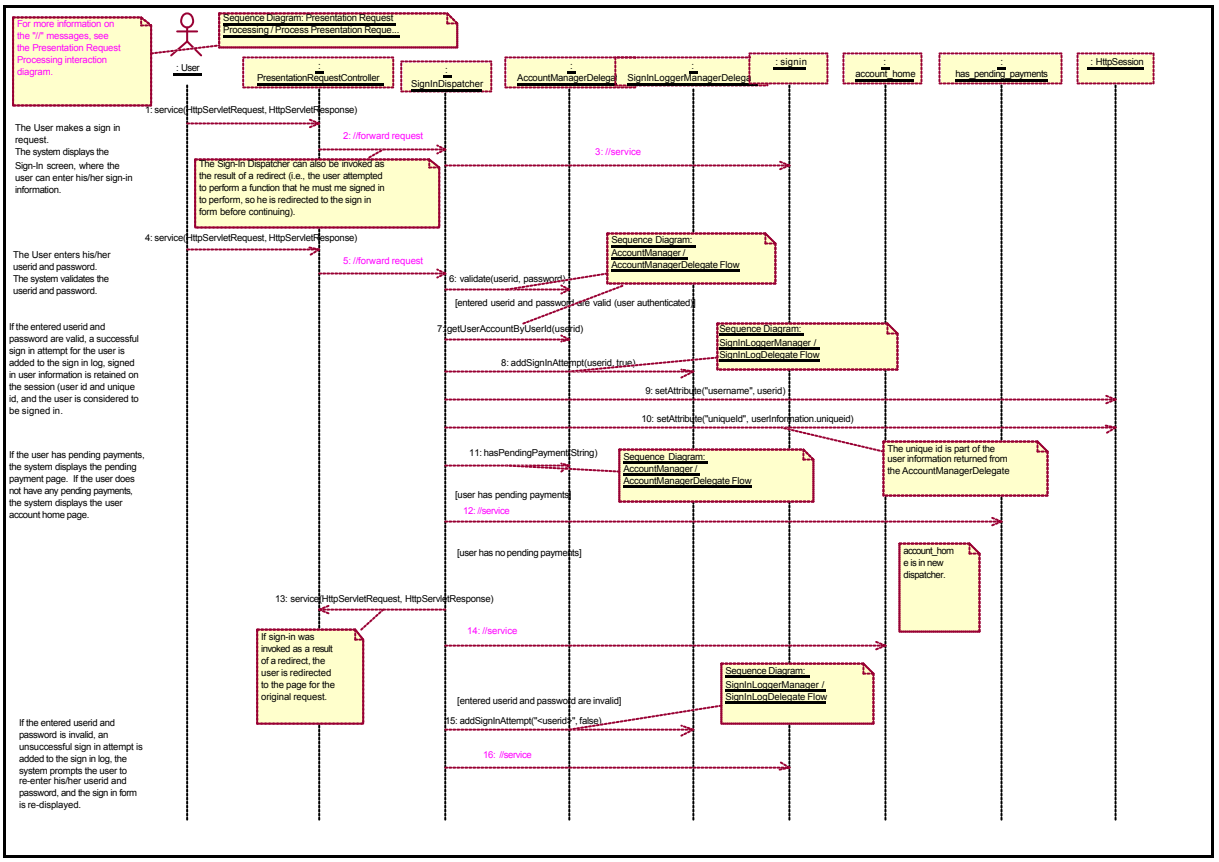

For more details on the "#" messages, see the Presentation Request Controller mechanism.
Sequence Diagram: Presentation Request Processing / Process Presentation Request ()
:Seller
:createauction_info_client_page
createauction_paymentinfo_client_page
PresentationRequestController
CreateAuctionDispatcher
AccountManageDelegate
createauction_info
createauction_confirm
AuctionManageDelegate
createauction_paymentinfo
createauction_results
1:service(HttpServletRequest, HttpServletResponse)
The Seller request to create a new auction.
If the Seller does not have any pending payments and the user has not previously listed items that did not sell, the system displays a screen to collect new auction information (the Seller's on file credit information is displayed as the default).
2:/forward request
Sequence Diagram: Sign In / Basic Flow
If the seller is not signed in, he/she is redirected to the sign in dispatcher. See the Sign In use-case realization.
3:hasPendingPayment(String)
Sequence Diagram: AccountManager / AccountManageDelegate Flow
4: hasPreviouslyListedItems(String)
[no pending payments and no previously listed auctions]
5:/service
6:build
The seller enters new auction information (may include a link to an image of the auctioned item).
The client page checks to make sure that information has been entered into the create account form, and that information is of the proper format (i.e., the right length, characters, etc.).
7:click on submit button
8:check()
9:next_check_step()
[entered auction information is invalid]
10:alert(<problem message>)
If the entered information is invalid, the user is asked to re-enter.
Once valid information is entered, the system validates the entered auction information (e.g., makes sure that the image URL is valid).
[entered auction information is valid]
11:service(HttpServletRequest, HttpServletResponse)
12:/forward request
Sequence Diagram: AuctionManager / AuctionManageDelegate Flow
13:validateAuctionInfo(AuctionInformation)
[entered auction information is invalid]
If the entered auction information is invalid, the system informs the user and asks the user to re-enter the auction information.
14:/service
[entered auction information is valid]
If the auction information is valid, the system displays the auction confirmation page so the user can confirm the entered auction information.
15:/service
16:service (HttpServletRequest, HttpServletResponse)
The Seller confirms that the auction should be created.
The system displays a screen to collect auction payment information (the Seller's on file credit information is displayed as the default).
17:/forward request
Sequence Diagram: AccountManager / AccountManageDelegate Flow
18:getCreditCardInfo(String)
See a separate diagram to see what happen if the Seller chooses to cancel the create auction request.
19:/service
20:create
Sequence Diagram: Create Auction / Seller Cancels Create Auction Request
21: click on the submit button
The user enters the auction payment information.
The client page checks to make sure that information has been entered into the create account form, and that information is of the proper format (i.e., the right length, characters, etc.).
22:check()
23:next_check_step()
[entered auction payment information is invalid]
If the entered information is invalid, the user is asked to re-enter.
24:alert(*<problem message>*)
[entered auction payment information is valid]
25:service(HttpServletRequest, HttpServletResponse)
If the entered payment information was valid, The system creates the auction (which automatically adds the auction to the auction catalog) and displays an "auction created" message to the Seller.
26:doPost(HttpServletRequest, HttpServletResponse)
Sequence Diagram: AuctionManager / AuctionManageDelegate Flow
27:/forward request
28:createAuction(String, String, String, String, String, String, String, java.util.Date, String, java.util.Date, String, long, double, double, String, String, String, double, String, String, String, java.util.Date)
29:/service

7.6 Sign In

Participants



Basic Flow

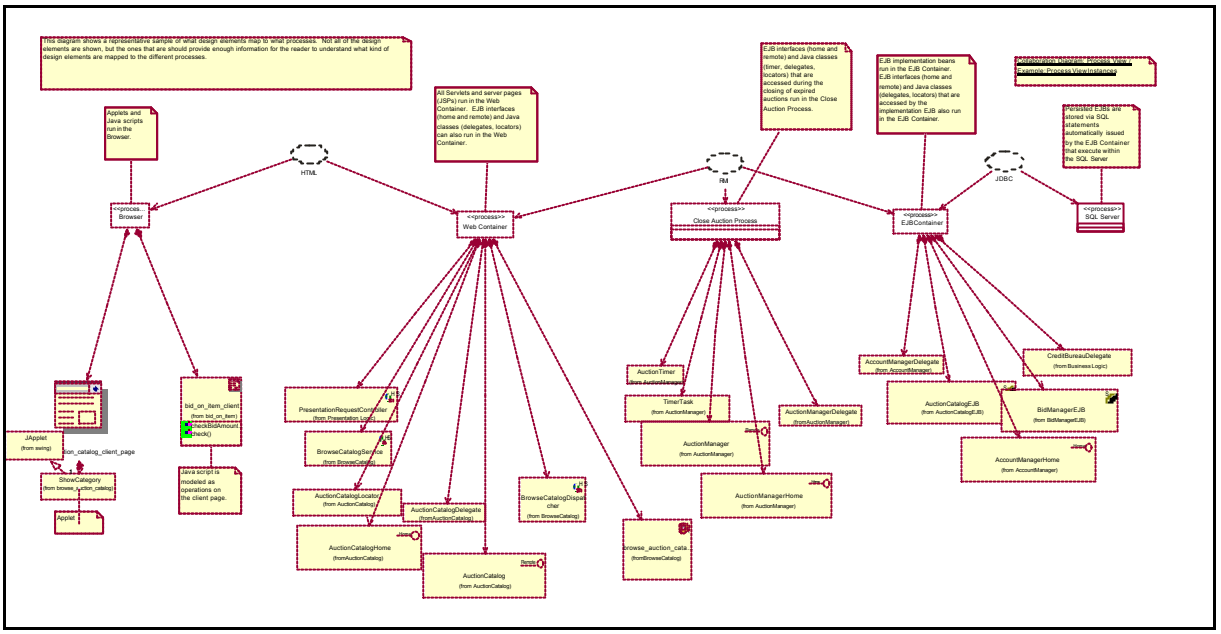


8. Process View

The process view of a system shows the assignment of active classes (classes that must run in their independent threads of control) to the operating systems processes and threads. In the case of J2EE deployment platform, the operating system resources are “hidden” underneath the container services. In other words, the infrastructure manages the operating system resources. In particular, the containers are placed in operating system processes or JVMs and the containers in turn manage thread-pools and assign the threads to active objects.

In the J2EE architecture JSPs, Servlets, and EJBs are assumed to be active. Hence, the process view of a J2EE-based system is rather straightforward and shows the communication mechanisms between the containers and what types of design/implementation elements run in what container. The process view of the PearlCircle Online Auction is shown below.

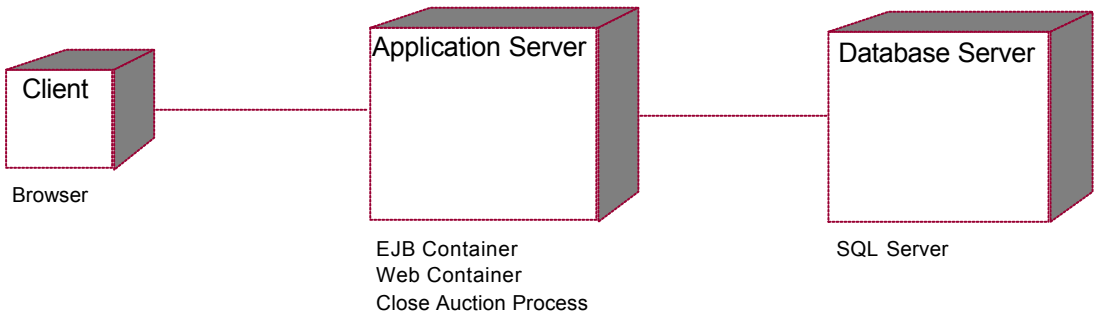
Process View of the PearlCircle Online Auction



The Close Auction Process is a standalone program that runs in a dedicated JVM. The process can be located on the same or different node as the web container or the EJB container. The process contains a scheduled thread that "wakes up" at regular intervals. The classes running on this thread communicate with the EJB container, determine which auctions should be closed, and close them.

9. Deployment View

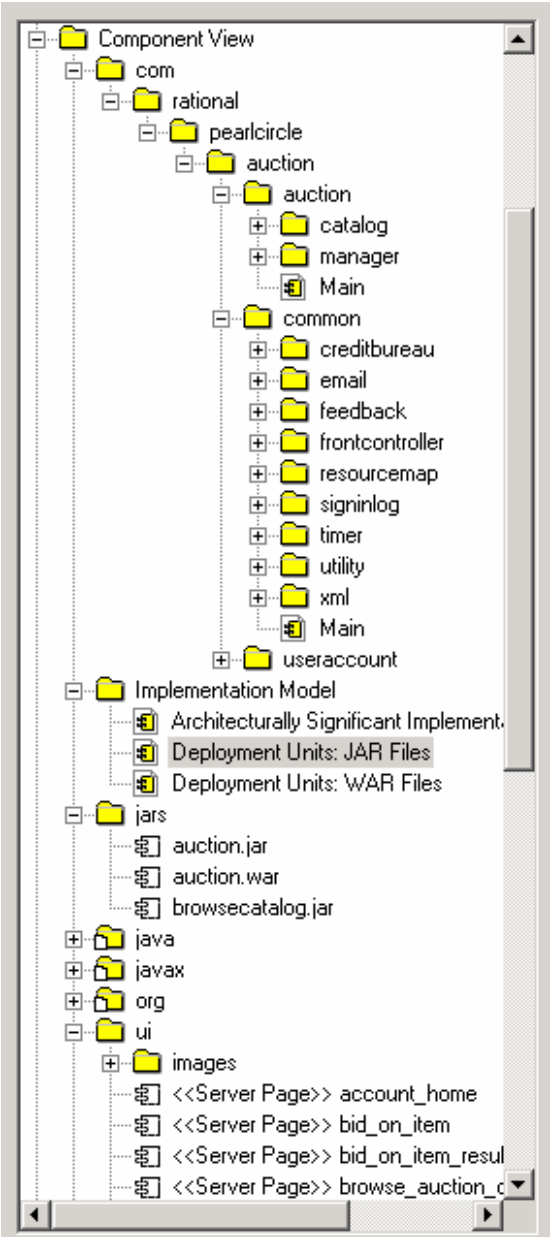
The deployment view of a system shows the physical nodes on which the system executes and the assignment of the system processed to the nodes. The PearlCircle Online Auction reference application can be deployed on different hardware configurations. For example, it can be deployed on a single NT workstation. The diagram below shows the most typical deployment configuration used by the development team.



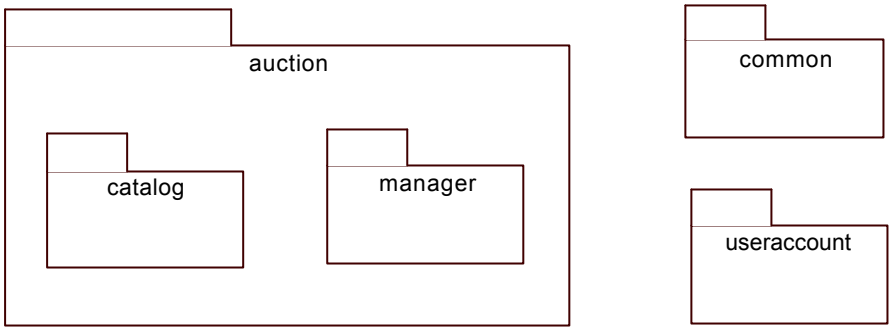
10. Implementation View

10.1 Source-code Components Organization

The structure of the system components representing the Java source-code units reflects the logical structure of the system. It can be best seen in the Rose browser and its image shown below.



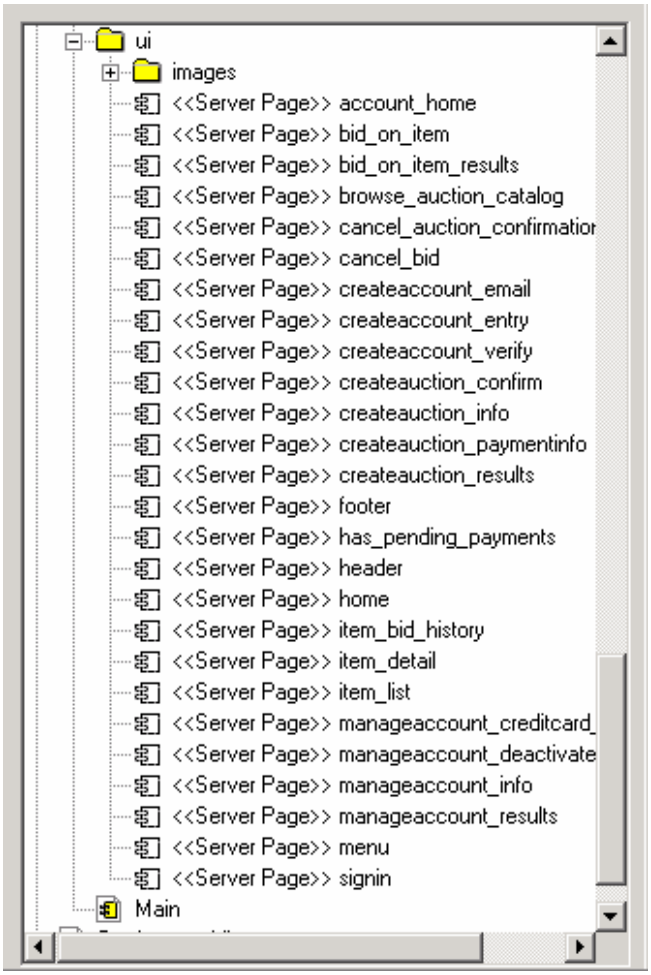
The structure also follows the recommended, URL-reversed nesting of Java namespaces that is identical to the source-code directory structure. In our case, since the reference application has been developed by Rational, all source-code is stored in a subdirectory X/com/rational/auction in some project directory X. Inside that directory the components have been grouped along the lines of business components. This is shown in the component diagrams below and can also be seen in the browser window image above:



The components on the diagram above trace to the business components as follows:

- Auction::catalog t races to Auction Catalog business component
- Auction::manager traces to Auction Management business component
- Useaccout traces to User Account Management business component, and
- Commontraces to Common Elements and Servicesw package (the internal structure o f the namespace/directory follows the decomposition of the package).

The server pages (implemented as JSPs) and the images are all grouped in the UI package that maps into subdirectory Xui in the project directory X. The structure of the package is shown in the figure below.



10.2 Deployment Components

The Online Auction application has three deployment units:

1. auction.jar which contains all classes in the ../com/rational/auction/* subdirectories
2. browsecatelog.jar which contains the applet, and
3. auction.war which contains the server pages

The deployment components are in the Component View::jars package that maps into a subdirectory X/jars in the project directory X.

11. Systems Size

The Online Auction application's size is described with the following items:

- Labor months: 18
- Business components: 3
- Dependencies on external components: 5
- Lines of Java code: approximately 10,000
- Java source files: 149
- JSP files: 54
- Implemented use cases: 7