Object Oriented Analysis and Design of a Web-Enabled Auction House

Y. Narahari, K.N. Rajanikanth, and Sourav Sen
Electronic Enterprises Laboratory
Department of Computer Science and Automation
Indian Institute of Science
Bangalore - 560 012, **India**
hari, knrajani, sourav@csa.iisc.ernet.in

Abstract

This paper describes the object oriented analysis and design of a web application, WHAT (Web-enabled House of Auctions). WHAT is an Internet auction application that serves a multitude of web clients and conducts multiple auctions concurrently. We present the steps involved in the conceptualization, analysis, design, and implementation of this application. UML (Unified Modeling Language) is the modeling language used in this work. We show how the static, dynamic, and operational aspects of the system are captured using various UML diagrams. Design patterns have been used extensively in evolving the analysis model of WHAT into a design model.

1 Introduction

Object technology is a sound paradigm to economically produce enduring and resilient industrial-strength software systems [1, 10]. Object oriented modeling is based on real world concepts and objects lead to an accurate description of real-world processes.

The unified modeling language (UML) [2, 5] is a visual, highly expressive language for describing the constructs and relationships of a system. More recently, UML has emerged as a suitable modeling language for web-based applications [4, 3]. By combining the most useful aspects of object oriented methods and extending the notation to cover new aspects of system development, UML provides a comprehensive notation for the full life cycle of Object oriented development.

In this paper we describe the use of UML for specification, analysis, design, construction, and documentation of WHAT, a web application for Internet auctions.

1.1 WHAT: Web-enabled Auction House

WHAT is a web application for an on-line auction system. Internet auctions have emerged as an important business model for E-commerce and automated negotiations [8]. Primarily, WHAT is a web server with auction logic and a backend database, and services a multitude of web clients. It supports:

- **Open Cry Auction** : wherein each buyer may know the bid submitted by a competing buyer and can respond to it. When the bidding phase is over the bidder(s) with the highest bid(s) gets the item.
- **Dutch Auction**: wherein the seller attaches a high asking price for the item. Then he gradually decreases the asking price until buyers emerges with bids specifying how many items they will purchase at the current asking price.

See the excellent tutorial by Manoj Kumar and Feldman [8] for more details on these auctioning methods.

WHAT is required to have facilities for:

1. Registration of buyers and sellers.
2. Setting up the auction event: this involves describing the item(s) on auction and setting up the auction rules:
    - type of auction
    - negotiable parameters in auction
    - start time of auction
    - auction closing rules
    - penalties for defaulting

3. Conducting the bidding process by collecting bids from buyers and implementing bid control rules

4. Evaluation of bids and closure of auction

WHAT only matches buyers and sellers through auctions. It does not take care of payment or trade settlement; however, it makes provisions for those. WHAT should have support for defining of ”new” auctioning mechanisms and algorithms and deployment of agents.

1.2 Software Process

The Rational Unified Process (RUP) [7] is the recommended software process for object oriented product development. In developing WHAT, we have followed RUP in spirit. The following core workflows were used in the process.

1. **Requirements Analysis** to come up with all important use cases, their descriptions, and and use-case diagrams.

2. **Domain Analysis** to discover all important abstractions for the problem domain (classes, interfaces, collaborations, and relationships). Develop all relevant structural and behavioural models (UML structural diagrams and UML behavioral diagrams).

3. **Design**: This will involve architectural design and detailed design. Issues to be addressed here are: GUI design; database design; and need for technical classes. Design patterns [6, 9] can be used here to maximum effect.

4. **Implementation and Testing**: Some issues to look for here are web enabling and concurrency of transactions.

UML was used extensively in all the above workflows above in an iterative way.

1.3 Outline

This paper is organized as follows. Section 2 details object oriented analysis of WHAT, covering both use-case analysis and domain analysis. We begin by outlining the requirements. Use cases and use case diagrams capture the requirements precisely. Domain analysis which involves defining object structure and behavior is presented next using UML class, sequence and statechart diagrams. Section 3 describes the object oriented design of the application. Design consists of architectural, mechanistic, and detailed design. The overall architecture of the system is described. UML diagrams are presented for identifying the key strategies during the organization of the system. Section 4 is exclusively devoted to mechanistic design centred around the discovery and use of patterns of object collaboration. Section 5 provides some implementation details and presents a component diagram and a deployment diagram for WHAT. Finally Section 6 provides directions for future work and research.

2 Object Oriented Analysis of WHAT

2.1 Requirements Analysis

The textual requirements specification for WHAT is as follows:

```
-- Registration of Buyers and sellers:   This involves
    o    Buyers and sellers register into the system before any
         meaningful activity with the system.

-- System Login: This involves
    o   Some way of authenticating the participants.
    o   Seller login is before any setting
        up/canceling/closing/modifying of an auction.
    o   Buyer login is before bidding / querying an auction.

-- Setting up the auction event: this involves
    o   Describing the item(s) on auction.
    o   Setting up the auction rules.
    o   Setting up the auction type.
    o   Setting up Negotiable parameters of the auction.
```

```
-- Scheduling of the auction and conducting the bidding process:
   this involves
   o    Automatically starting an auction at the scheduled start
        time.
   o    Implementing the bid control rules.

-- Evaluation of bids and closure of auction: This involves
   o    Automatically closing an auction at the scheduled stop
        time.
   o    Evaluate bids and notify winners/participants.
   o    Permit seller to cancel/close auction based on rules.

-- Query support
   o    Facilities for obtaining information on auctions/bids

WHAT only matches buyers and sellers through auctions. It does not
take care of the payments or trade settlements. However it makes
provisions for those.
```

2.1.1 Use Case Analysis

We capture the functionality of the system through the use cases. Firstly we identify the actors of the system as: Buyers, Sellers, and Auction Monitor.

Secondly we identify the use cases for each of these actors by considering:

1. Functions the actors requires from the system.
2. Information retrieval/modification in the system (by the actor)
3. Events in the system to which the actor needs to respond / be notified about.

The use cases identified are: Register, Register Buyer, Register Seller, Log-in, Log-out, Query, Setup Auction, Setup Opencry auction, Setup Dutch Auction, Bid, Notify, Cancel Auction, Modify Auction, Close Auction, and Autoclose Auction. Figure 1 shows a UML use case diagram depicting the relationships among these usecases. The diagram is self-explanatory. We provide the descriptions for two use cases: Setup Auction and Cancel Auction.

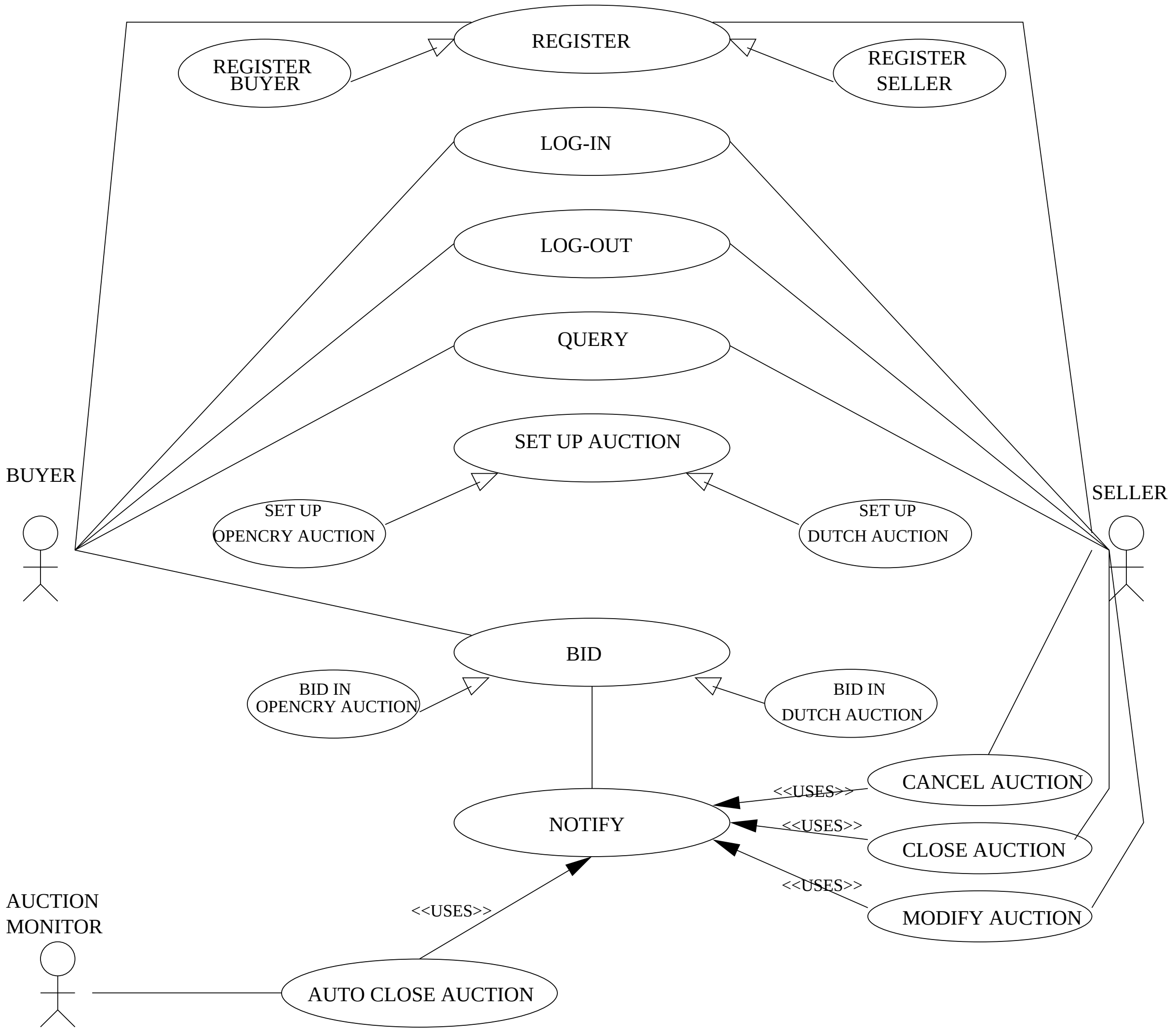

Figure 1: A use case diagram for WHAT

SETUP AUCTION Use case

- Main Flow

  1. The seller chooses to set up an auction.
  2. The system queries the auction type and product category.
  3. The seller chooses Dutch/Opencry auction and the product category.
  4. The system provides the Dutch/Opencry auction setup page.
  5. The seller provides the detailed information and also provides the auction rules. (**E1**).
  6. The system checks whether the auction can be hosted. (**E2**).
  7. The auction is set up.

- Alternate Flows

  1. **E1**: (1) The input parameters are incorrect. (2) The system provides the setup page to input the parameters again.
  2. **E2**: (1) The system checks and finds that resources are not available to host the auction. (2) System displays a message to the user that the auction cannot be hosted.

BID Use case

- Main Flow:

  1. The buyer chooses to bid in an opencry/Dutch auction.
  2. The system provides the opencry/Dutch auction bidding page.
  3. The buyer provides the necessary details like bid amount, auction id, etc.
  4. The system validates the bid . (**E1**).
  5. A valid bid is registered in the system.

- Alternate flows (**E1**)

1. The bid is invalid.
2. The system displays a message that an invalid bid has been submitted and prompts the user to submit a new bid.

2.2 Domain Analysis

2.2.1 Class Diagram

The information gathered during the construction of the use cases was used to perform object decomposition and build the object structure of the system.

- Object identification: The key objects identified were: Auction, Bid, Participant, Rules, Buyer, Auction House, Seller, Product.
- Identification of relationships: It was realized that the relationships between major classes may be classified as follows:
  - Buyer and seller both are kind of participants. Dutch bid and open cry bid are kind of bids. Dutch rules and opencry rules are kind of rules. So these fall into the category of *is a* relationship.
  - Individual Auctions and Traders are part of auction house. So they fall into the category of aggregation relationship.
- Identification of object attributes: We used the following strategies to discover attributes. (1) Information to define the object. (2) information the methods of the object act on. (3) Information necessary to fulfill the responsibilities of the object.

Thus we arrive at the class diagram shown in Figure 2. The basic structure of this diagram is similar to the one presented in [8].

2.2.2 Sequence Diagram

A scenario is a sequence of events flowing between actors and the system for some purpose. The order dependent view of how the system is expected

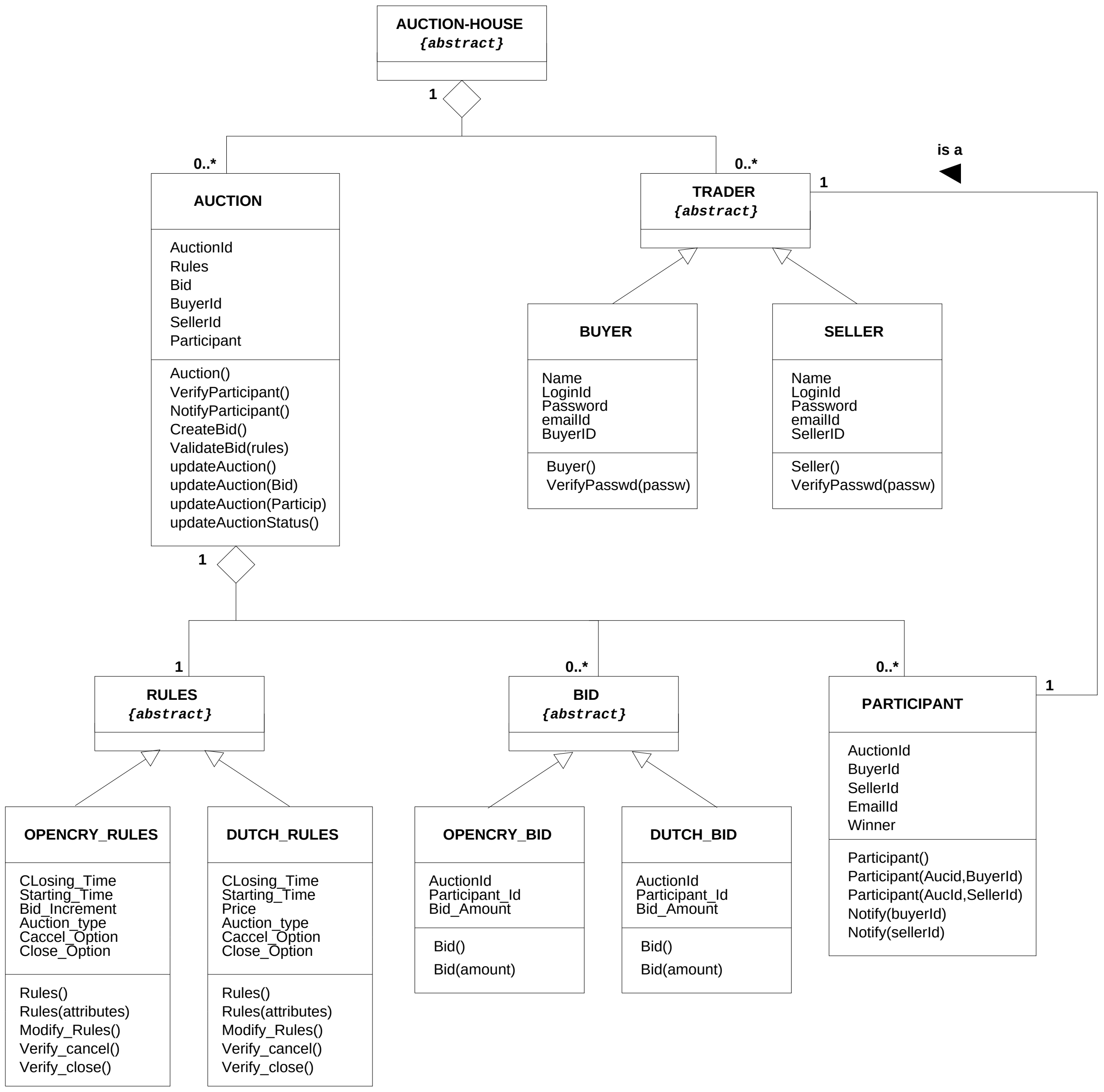

Figure 2: Domain class diagram for WHAT

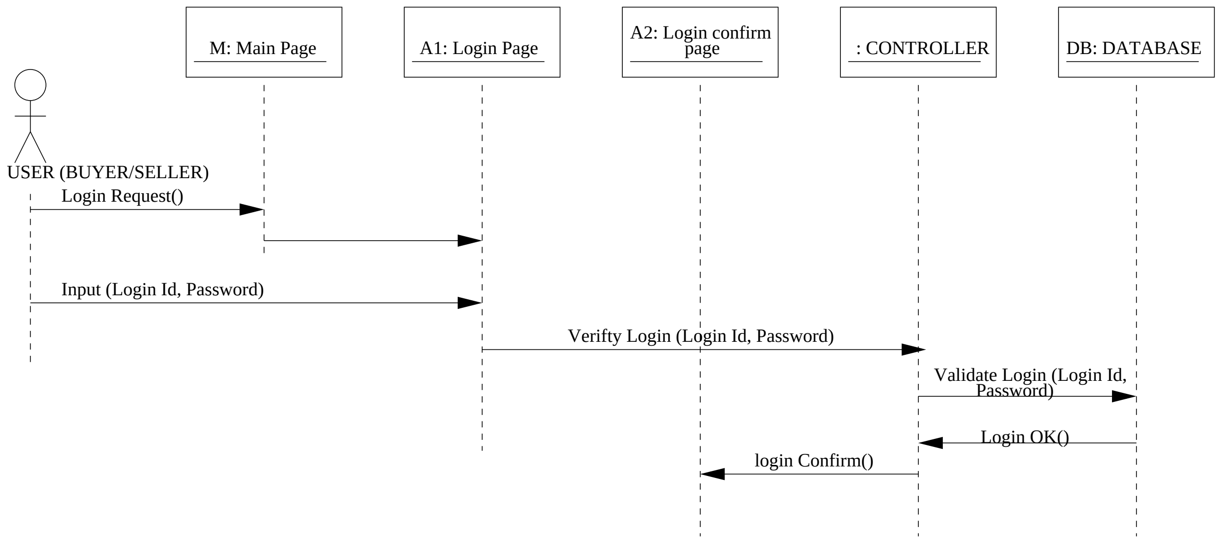

Figure 3: Sequence diagram for login use case

to behave when actually used, is captured by sequence diagrams. We present some of the sequence diagrams pertaining to WHAT (Figures 3 and 4). Figure 3 depicts a sequence diagram that describes the interactions comprising the login usecase. The sequence of interactions in the cancel auction usecase is shown in Figure 4. Such sequence diagrams can be written down for all the use cases.

2.2.3 Statechart Diagram

State chart diagram shows complete behavioral model of an object. The following state chart diagram shows the various states an auction object can be in (Figure 5). Many domain objects in WHAT have interesting state chart diagrams and one can write down these state chart diagrams.

2.2.4 Activity Diagram

We present an activity diagram for placing a bid. Figure 6 shows the diagram. The diagram has with three swimlanes: the Client, Server and Backend lanes. This diagram can be used in forward engineering (that is, producing the code for the activity). As usual, activity diagrams can be written down for all non-trivial activities in the system.

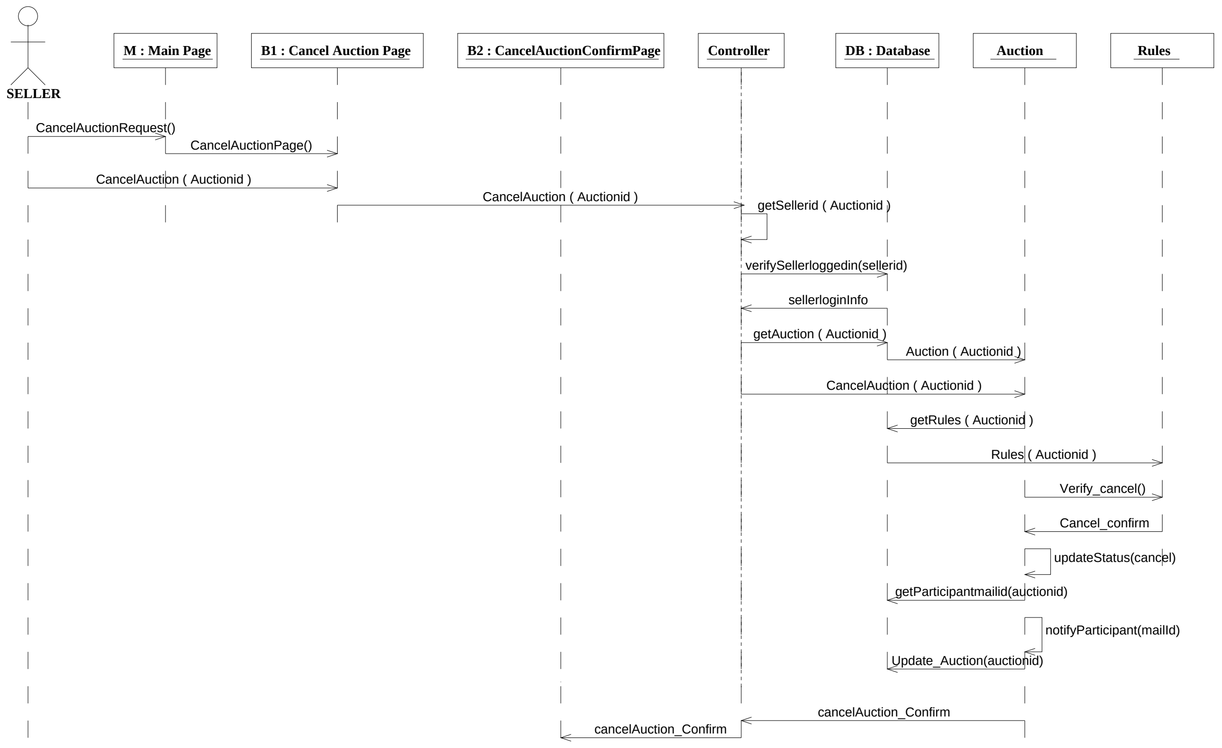

Figure 4: Sequence diagram for cancel auction usecase

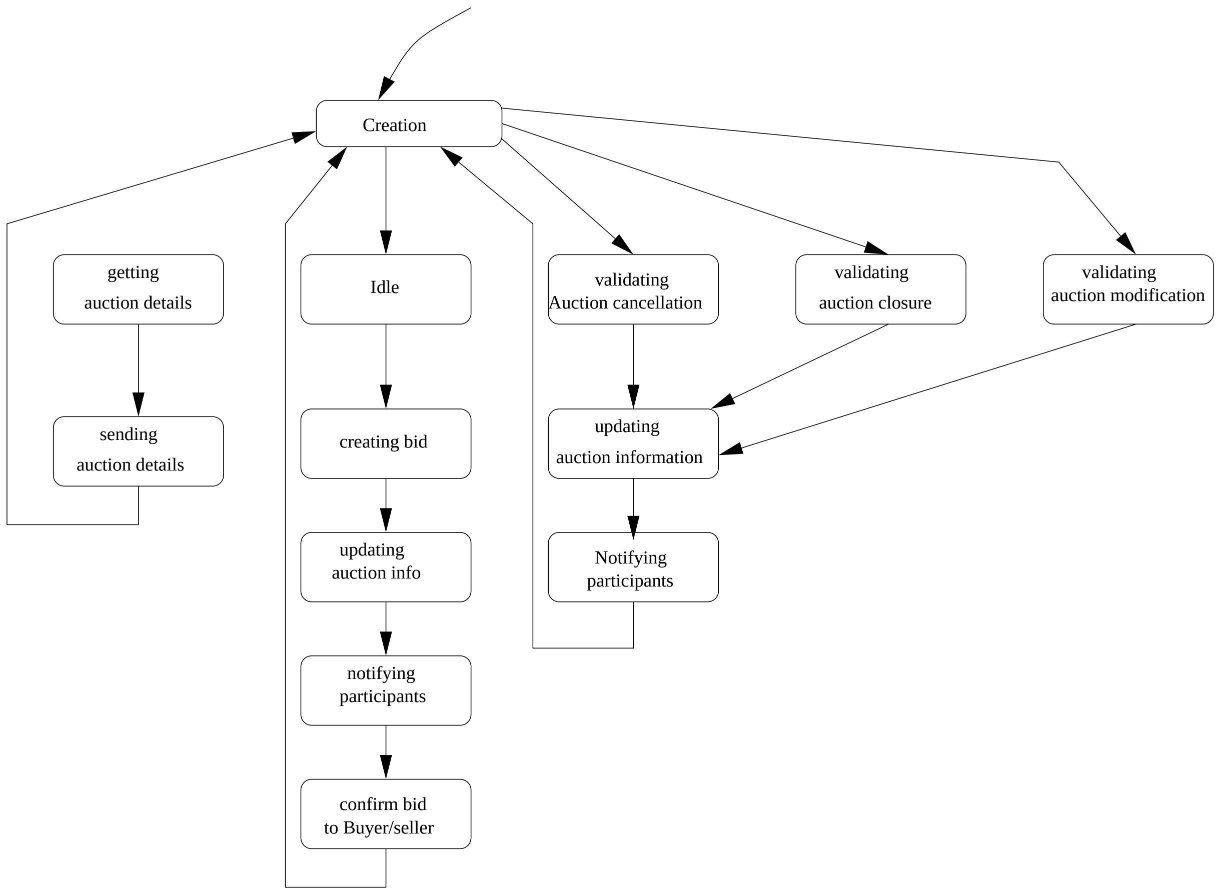
Creation
getting auction details
sending auction details
Idle
creating bid
updating auction info
notifying participants
confirm bid to Buyer/seller
validating Auction cancellation
validating auction closure
validating auction modification
updating auction information
Notifying participants

Figure 5: State chart diagram for auction object

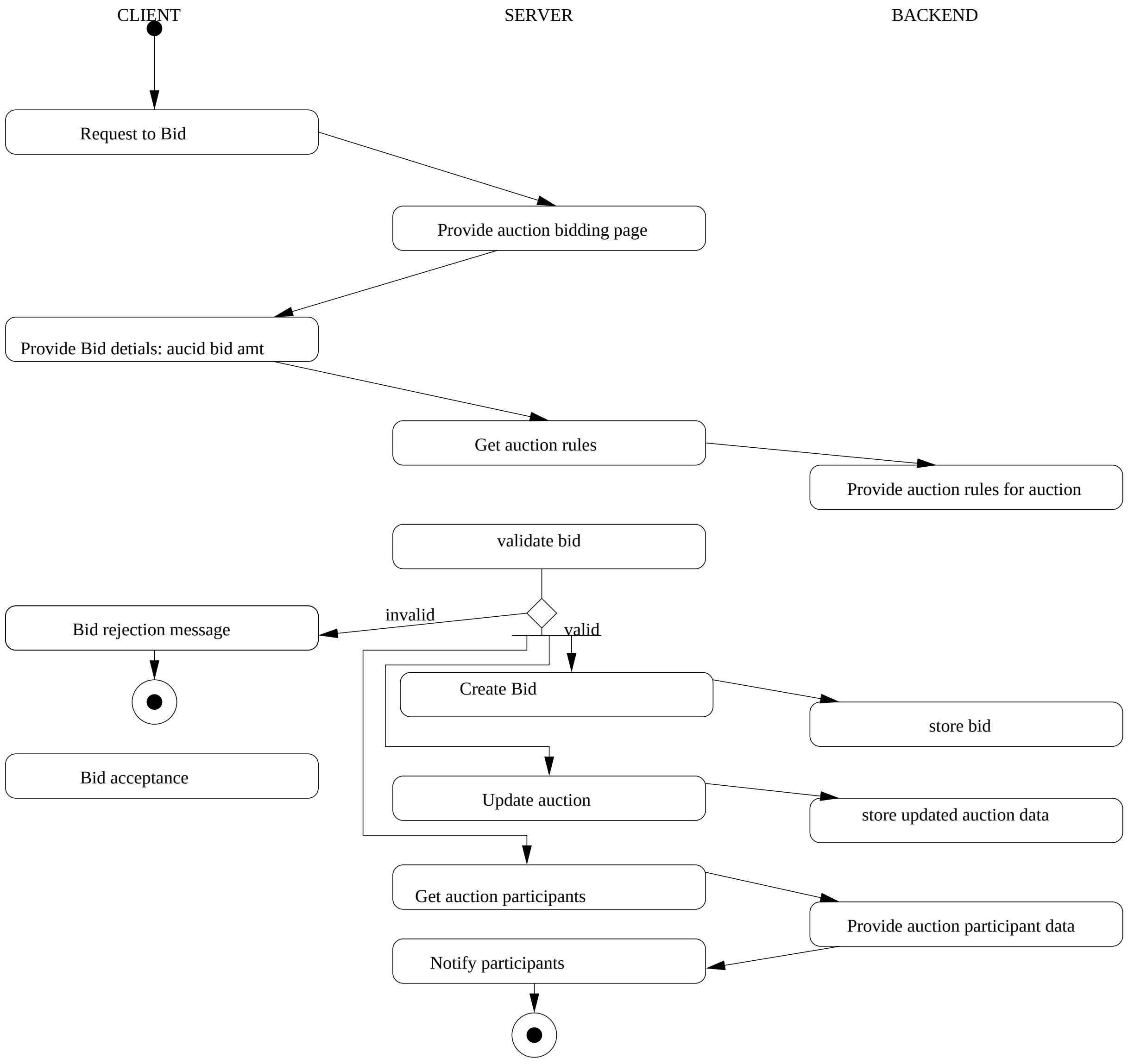

Figure 6: Activity diagram for placing a bid

3 Object Oriented Design

Design is the process of specifying an implementation that is consistent with the analysis model.

3.1 Architecture Design

A well-designed architecture is the foundation for an extensible and changeable system In this phase, we come up with a high level design where subsystems (packages) are defined, including dependencies and communication mechanisms between between the packages. In the context of WHAT, we take into account the following technical implementation issues in the formulation of an architecture:

- We introduce the concept of a consortium of auction houses.
- The consortium handles the issues related to
  - Web enabling
  - Concurrency control
  - Load balancing
- The consortium maintains the following information
  - Registry of traders. Traders only communicate with the consortium and have no idea of the auction servers.
  - Registry of servers
  - Information about individual auction houses

Figure 7 shows a logical view of the architecture comprising web clients, consortium, auction servers, consortium database, and auction server databases. We propose a four tier architecture as shown in Figure 8 for the system. Following are the packages in the proposed architecture:

1. **UI Package**: Consists of HTML files which enable the user to enter data and thus interact with the system.

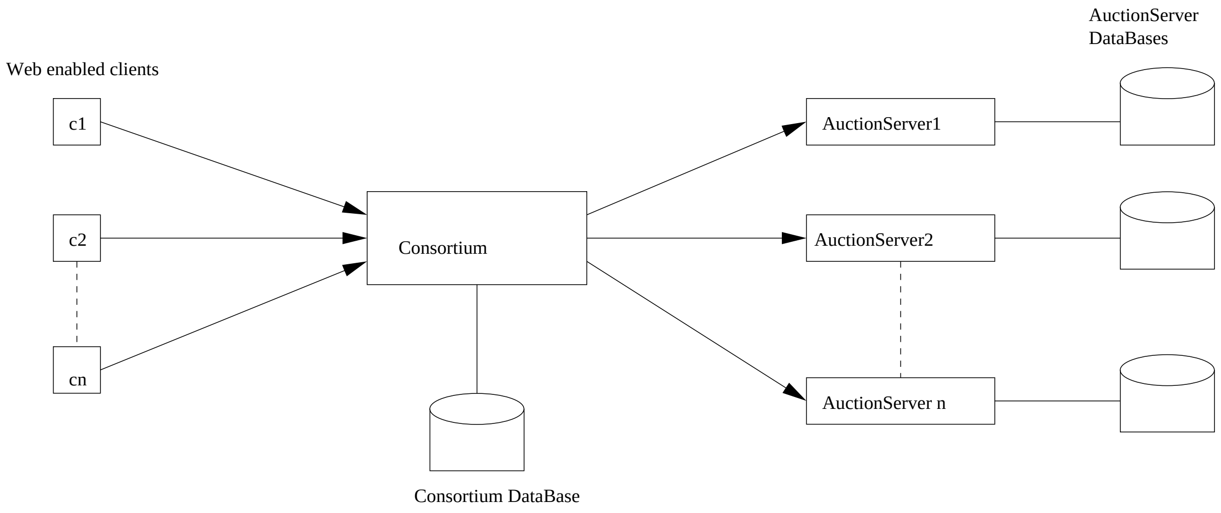

Figure 7: Architecture of WHAT

2. **Middleware Package**: Consists of master servlet which invokes appropriate threads to handle different requests .The code run in the threads in turn invokes appropriate local/remote methods. The communication between local and remote objects are done through CORBA. Both consortium and the auction servers have databases integrated with them and the business logic layer both at the consortium and auction server end uses the services of the database layer.

3. **Business Logic Package**: Consists of worker classes which implement the auction logic in part. These package also uses the services of database package. The classes in these package implement auction starting and ending and sending mail to various parties at appropriate points in time using utility package classes. They are also responsible for maintaining consistency of information at the consortium and the server end.

4. **Database Package**: Consists of database initialization, connectivity and database manipulation components.

5. **Utility Package**: Consists of utility classes like mail sending and other specific information gathering classes (Like those that passively take user statistics).

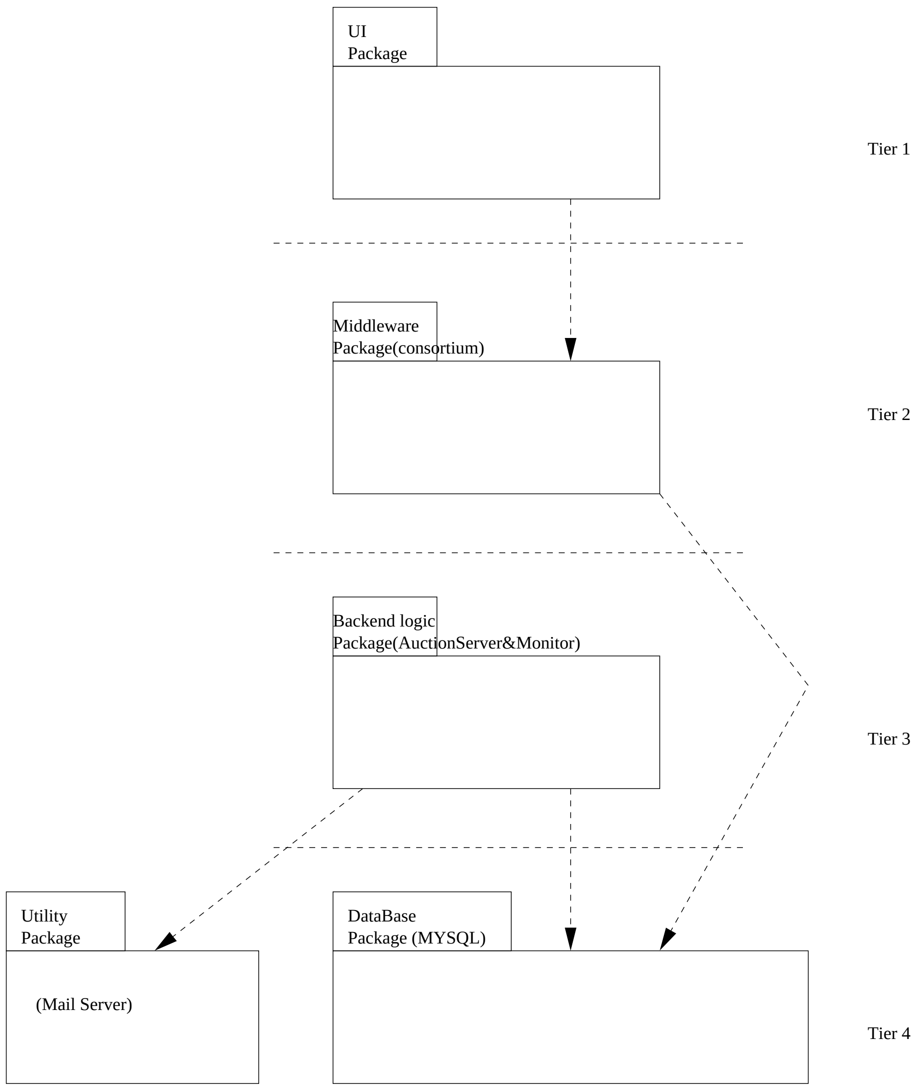

Figure 8: A four tier architecture for WHAT

4 Design Patterns in WHAT

Design patterns are simple and elegant solutions to specific, commonly occurring problems in software design [6, 9]. It is well known that design patterns make it easier to reuse successful designs and architectures of experienced and professional designers. The following design patterns have been used in the design of WHAT.

- **Abstract Factory**: This has been used for generating objects for different auction protocols. Different auction protocols require different sets of individual objects to be created and used together. The use of Abstract Factory automatically enforces this constraint.
- **Builder**: WHAT clients may be presented with different kinds of web pages (HTML pages, forms, etc.). To separate construction of different kinds of web pages from representation, the Builder pattern has been used.
- **Bridge**: Bridge pattern has been used in several ways:
  - Avoid permanent binding between auction type and effective price
  - Determining winners in auctions
  - Different types of notification mechanisms
- **Composite**: The bid structure in complex auctions (such as combinatorial auctions) can be complex since it includes arbitrary combinations of products as part of a bid. If the Composite pattern is used to represent bids, then WHAT will be easily extensible to any type of auction.
- **Decorator**: Decorate auction object with multiple individual value added services such as can be provided to sellers and buyers. Buyers and sellers may requisition these value added services dynamically and the Decorator pattern can be used to add/delete these services at run time.

- **Facade**: Auction house provides a unified interface to the whole system and uses the Facade pattern.
- **Iterator**: To access information from the database about individual auctions without knowing the type of the auction. For example, information about a Dutch auction may be stored in one way and information about an Opencry auction may be stored in a totally different way, but by using the Iterator pattern, it is possible to extract information without knowing the type of auction whose information is being sought.
- **Proxy**: To provide detailed description of item. Some items on auction may have complex images, VRML files describing them. These may be loaded only on demand.
- **Observer**: Automatic generation of services, notifications, etc. when designated events happen during the auction process. The occurrence of such events triggers the notifications to be sent to all subscribers to the event.
- **Strategy**: Different types of auction protocols have different ways of determining winners, settling the trade, etc. Strategy can be used in these situations.
- **Chain of Responsibility**: Consortium forwards requests to a chain of auction houses to handle new auction requests from sellers.

The use of the above design patterns has resulted in an extensible design model for WHAT. The following factors enable a high level of reusability and extensibility of the WHAT design model.

- Create objects indirectly and avoid creating and object by explicitly specifying a class
- Reduce dependence on object representation or implementation by hiding the internals from clients

- Reduce algorithmic dependencies by isolating algorithms that are likely to change
- Promote loosely coupled classes
- Avoid subclassing as a means to extend functionality; use composition instead
- Comprises abstract classes from which application-specific subclasses can be created to build a particular customized application
- Design reuse and flexibility are important

5 Implementation

WHAT has been implemented as a four tier architecture as shown in Figure 8. The UI package comprises HTML pages and forms to be used by the web clients. The middleware package (consortium) has been implemented using CORBA. The Business logic package and web services employ the Java servlet technology. MySQL database system has been used for the consortium database and also the databases of individual auction houses. The utility package is basically the mail server.

Figures 9 and 10 show the component and deployment diagram, respectively for the implementation of WHAT.

6 Future Work

WHAT is an experimental, prototypical product and in its present state of implementation, cannot scale to high levels. However, the design model of WHAT has been created to support highly scalable and powerful implementations. Some features that future versions of WHAT will have to support include:

- complex auctioning mechanisms and algorithms
- deployment of agents

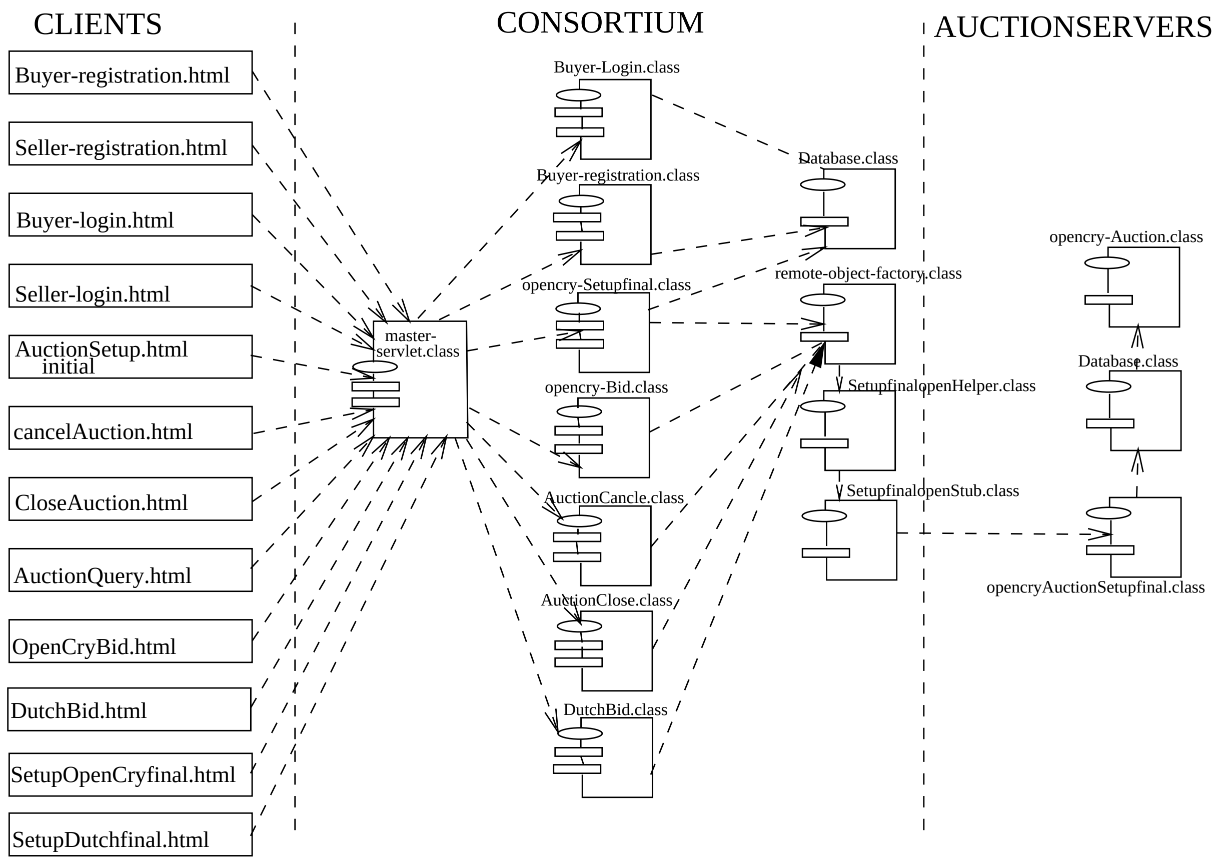

Figure 9: UML component diagram for WHAT

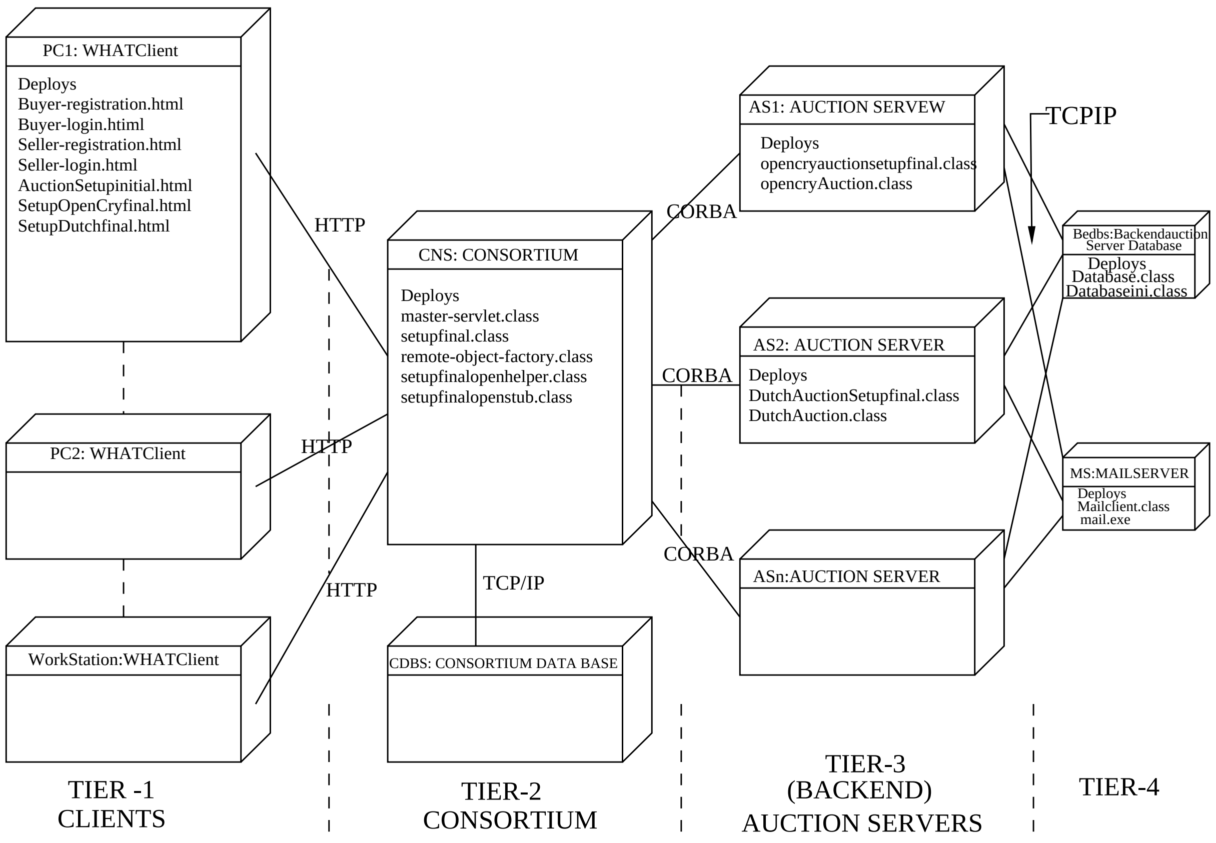

Figure 10: UML deployment diagram for WHAT

- security of data and transactions
- maintaining anonymity of buyers and sellers
- notification mechanisms to indicate the status and progress of auctions
- participation in multiple auctions
- searching through the ongoing auctions for desired information (search engine)

Due to paucity of space, this article could not cover all details of analysis and design of WHAT. Interested readers should contact any of the authors by email for more details.

Acknowledgments

We would like to thank Mr Chandrasekhar for preparing excellent figures for this paper.

References

[1] G. Booch. Object-Oriented Analysis and Design with Applications. Second Edition. Benjamin Cummings, 1994

[2] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide. Addison Wesley Longman, 1999.

[3] Jim Conallen. Modelling Web Application Architectures with UML. *Communications of the ACM*, October 1999, pp.63-70

[4] Jim Conallen. Building Web Applications with UML. Addison Wesley, 1999.

[5] H.E. Eriksson and M. Penker. UML Toolkit. Wiley Computer Publishing, 1998.

[6] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. 1995. Addison-Wesley Longman, Second ISE Reprint, 1999.

[7] I. Jacobson, G. Booch, and J. Rumbaugh. The Unified Software Development Process. Addison Wesley Longman, 1999.

[8] Manoj Kumar and Feldman. Internet Auctions. http://www.ibm.com/iac

[9] Patterns Home Page. http://www.hillside.net/patterns/patterns.html

[10] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. Object-oriented Modeling and Design. Prentice Hall, Englewood Cliffs, 1991. Reprinted by Prentice Hall of India, Eastern Economy Edition, 1997

[11] Mary Shaw and David Garlan. Software Architecture - Perspectives on an Emerging Discipline. PHI Eastern Economy Edition, 1999.