w3auction

an online auction tool for wireless devices

Final Report

Teleseminar Nice-Mannheim

Winter Semester 2001 / 2002

Vanessa Bouchet, Lars Klose, Heiko Kopitzki,
Annabelle Le Sonn, Dorothée Robert, Gunnar Wiedenfels

CONTENT

Abstract

As handheld devices like personal digital assistants (PDAs) are becoming very popular, prices are dropping, and in addition an overall trend to wireless communications can be observed, this project tries to take advantage of these future developments. Its goal is to design and implement a tool to support wireless auctions on PDAs, either in a local scenario or on the Internet, worldwide. However, by making use of the system independent, Java Technology, the result isn't even limited to PDAs, but may also be used on notebooks or regular PCs, with or without wireless communication. and is therefore to be concerned a very flexible tool.

Concerning auctioning on the Internet, this tool tries to combine already existing and available technology, with the common image of what real-time auctioning is. We'd like to mention two examples: The widespread Internet company EBay, which provides a platform to buy and sell personal goods at auctions, and the famous English auctioneers Sotheby's. This solution combines these different approaches and makes it possible to participate in a real-time auction, with all the thrill and competition that comes with.

In addition this system bears in mind the general concept of auctioning, by providing different algorithms which will make it possible for the user to run specialized auctions which fit best his personal taste or the goods to be sold.

1 Introduction

This paper is part of the software project “W3A – World Wide Web Auctioning”. It gives an overview about this project, the development process and the resulting software. However, this is no complete reference and therefore should be read in combination with other documents which go further into detail.

This software project was part of a Joint Seminar and Telecollaboration Project On Electronic Commerce (in short: Teleseminar). Participating members are the Universities of Nice-Sophia Antipolis (France) and Mannheim (Germany), as well as Accenture as an industrial sponsor. It combines the efforts of students in both countries by providing the possibility of videoconferencing over the Internet, forcing both sides to communicate in English as a foreign language. It is a step forward to provide an international experience for students of both universities, as well as experience in a distributed team. See also [1].

Users of our software shall be able to participate in auctions in locals scenarios as well as in auctions world wide by connection through the Internet. The software is specialized for the use on portable handheld devices which are connected by wireless communication (several clients using one server which is connected to the Internet).

The main auction scenario is somewhere in the middle between a traditional “live” auction and Internet auctions, such as EBay: the participants are all in one room, but they submit their bids “live” using their PDAs (Personal Digital Assistants).

2 Architecture of the Auction System

2.1 Overall architecture

The system is based on a server client architecture: The server has the control over the system. It takes care of active elements like user connections and auction handling, and uses a relational database management system to keep track of user and auction data.There exist two types of clients, user clients and an administrator client. The user clients supports the role of auction participants, who may take part in auctions, but may also create their own auctions in case of an open Internet scenario.

The administrator client is a special client reserved for the administrator. It enables him to control the server from a remote location. This is the only way foreseen for the server to be managed. It’s true that a server console exists, but its functionality is limited to starting and stopping the server as well as loading auction algorithms and some other basic options.

Communication between the server and the clients is done through an appropriate proprietary protocol, which is based on the exchange of simple message blocks, so called PDUs (Protocol Data Units). It operates over TCP/IP. Figure 1 presents an overview for the system architecture. Grey blocks represent physical host systems.

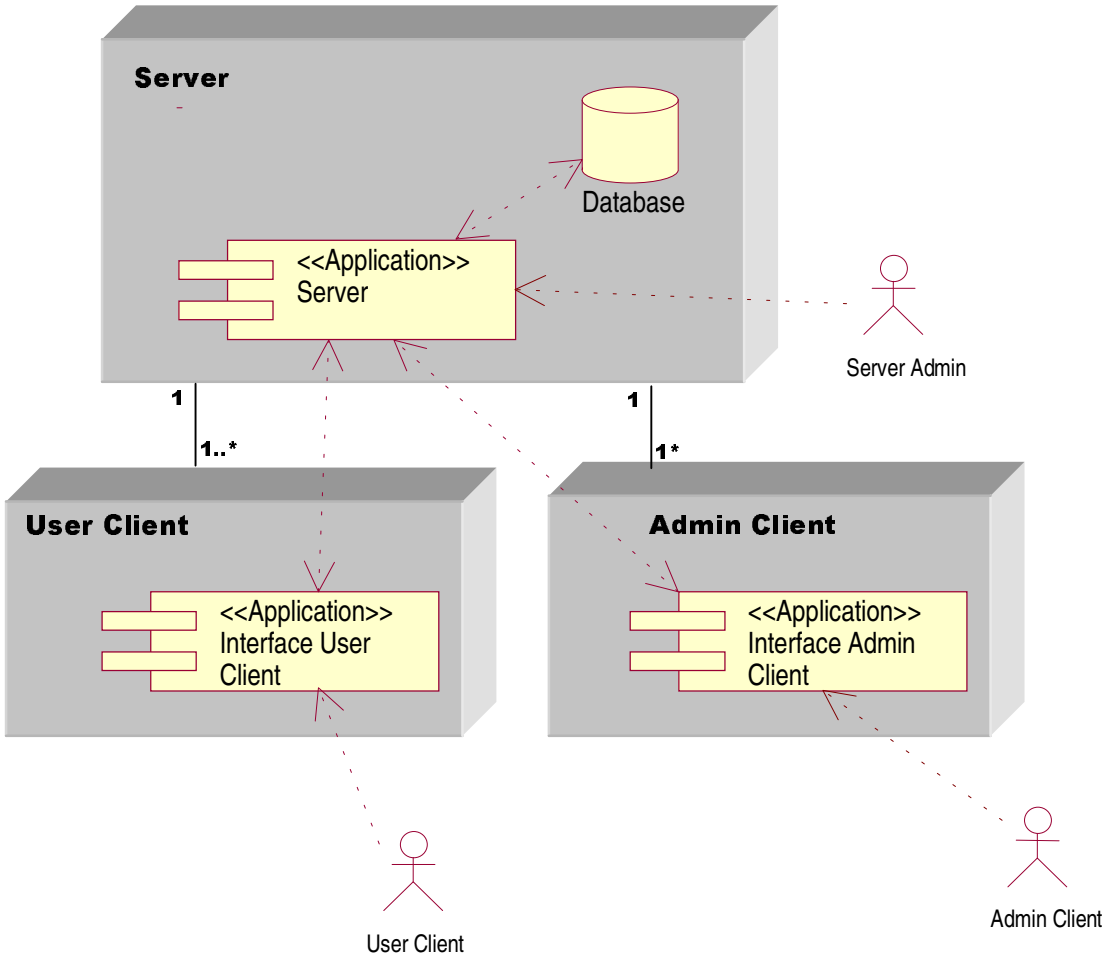

Figure 1 System Architecture Overview

2.2 Architecture of the Auction Server

The server architecture of the server is based on the following concepts: It exists a core module, that manages the cooperation of the other components:

- the DataStore which keeps the connection with the database and provides a general database interface which depends in its structure on the actual database. This allows us to use any kind of database by simply reimplementing this part.
- an auction spooler which manages the upcoming and current auctions, by constantly checking the database for scheduled auctions. If an auction is to be started, the spooler will start a new instance of the appropriate algorithm, which from there on manages the auction.
- a communication thread which is always waiting for new connections from users logging on. It will create a new independent connection thread for every new incoming user. It takes the socket of the client as parameter and from thereon is responsible for the sending and receiving of message packets.
- a UserInput which takes care of possible manual input by the server administrator. This input is restricted to basic operations like to start or stop the server.

Figure 2 gives an overview of the architecture of the auction server.

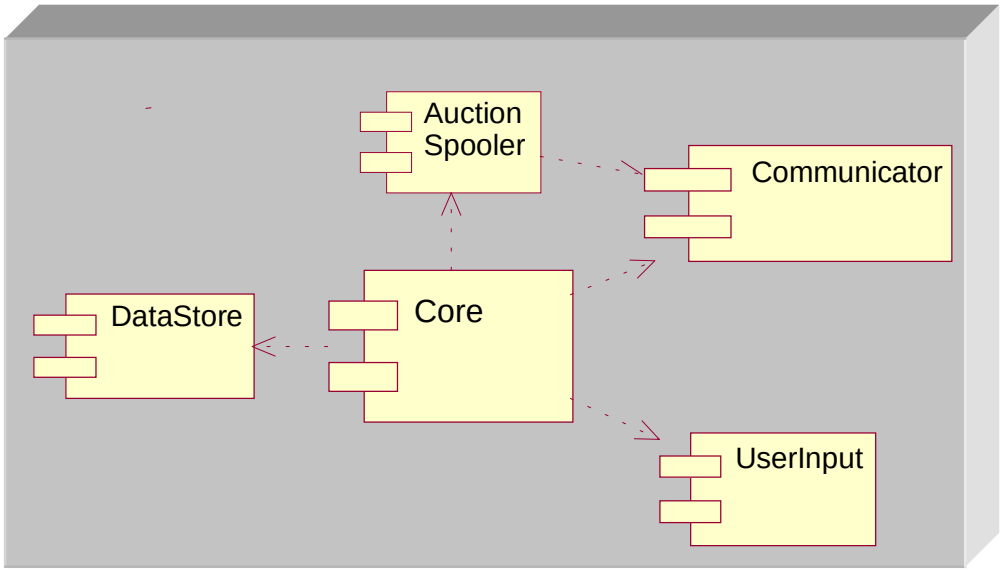

Figure 2 Server Architecture

2.3 Architecture of the Auction Clients

Similar to the server, the clients are based on a core module which manages the communication between edge modules. The user and the administration client bare the same architecture. The client may be seen as a simple translator between the server and the human user.
The edge modules are:

- the Communicator, which sends and receives messages to and from the server.
- the ClientGUI, the interface for receiving input by the user and presenting the data received by the server.

Figure 3 presents the client architecture.

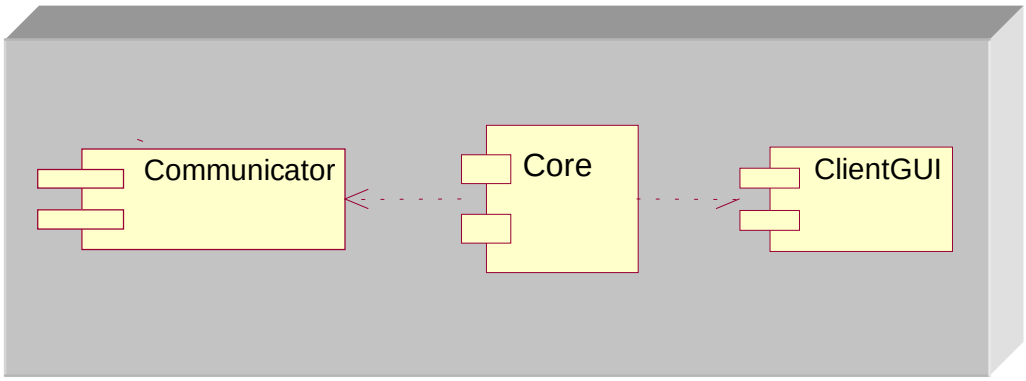

Figure 3 Client Architecture

2.4 UML Use Cases

Modern software engineering technology was used throughout this project: Object-oriented analysis and design, object-oriented programming, and UML as the specification language.
In Figure 4, an overview the use cases for this software project are presented. For reasons of place and time, only an overview is given.

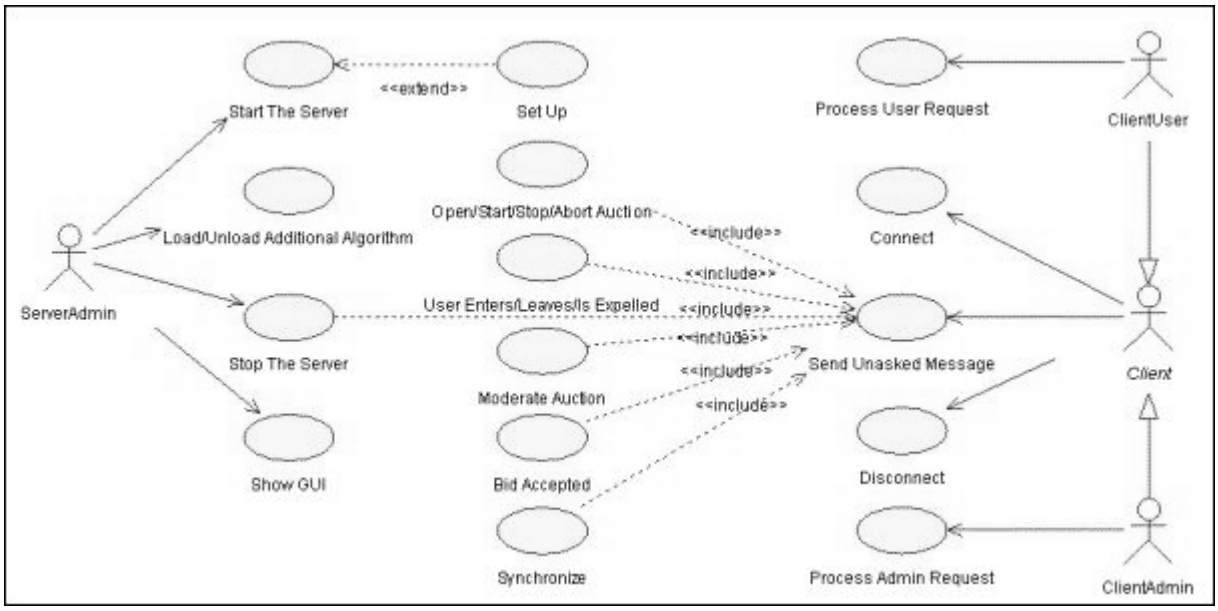

Figure 4 Use Case Diagrams – All Users

2.4.1 User Client

Before having the possibility to interact with the system, the *user client* needs to connect to the server, by specifying an IP address and a port (socket). If the connection succeeds, it is registered in the list of active connections by the server.

After logging in, the user can execute the following basic functions:

- He/she may change his personal settings: the current data is retrieved from the database and presented to the user. Then he/she can modify the entries, which are then stored again in the database. He/she may create auctions: The system provides several different auctioning algorithms from which the user can choose one. Depending on the algorithm, he/she must provide specific options. For example for an English auction, one must provide
    - an auction name
    - a category (e.g., “cars” for a Mercedes)
    - the start price (the minimum bid)
    - the step (defines the minimum difference in money between two bids)
    - a start date and time

  In addition he/she can provide optional data:
    - a description of the item
    - an image of the object (by providing the URL of a file containing the photo)
    - further information which may concern the type of payment etc.
- auctions can also be searched for, by specifying one or several of the following options:
    - name or the part of the name of an item
    - a category
    - the status (upcoming, current, open, etc.)
    - start date
    - the buyer or seller.

  After having searched for auctions, all auctions can be consulted with their details, and auctions which are already ongoing can immediately be entered.
  After having entered an auction, the user may participate by bidding and looking at the result until the auction is finished.

2.4.2 Admin Client

The administrator connects to the server by a specialized client, the *administrator client*. He/she needs to give an administrator password to authenticate himself as a valid administrator client. As can be seen in Figure 4 he/she has the choice to manage all major fields of the server, including:

- *auction management*: auctions can be deleted (the decision whether an ongoing auction needs to be deleted is based solely on the opinion of the administrator)
- *user management*: user accounts can be looked up in the database, modified and even deleted
- *error log management*: viewing and changing the log file the system maintains to keep track of errors and other necessary information
- *connection management*: to manage all users who are connected at that moment to the system.

2.4.3 Server Admin

The Server Admin is the person who is physically present at the site of the server. He is not necessarily different from an administrator who connects from distance. In contrast to an administrator client, this interface is generally not intended to manage the system itself, but to fulfil basic operations like to start the server, stop the server and to load or unload auction algorithms.

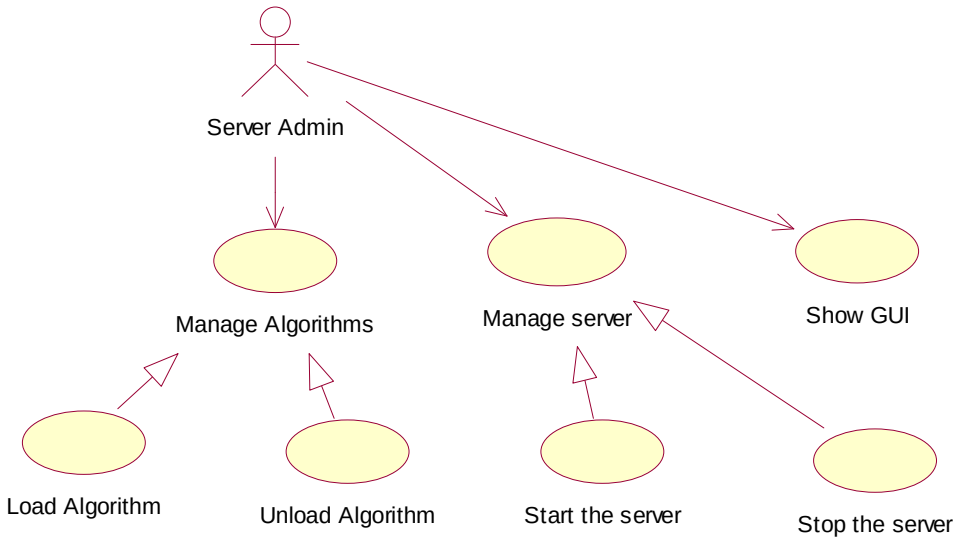

Figure 5 Use Cases – Server Admin

When the server is stopped, it is shutdown safely to prevent data loss or inconsistency: auctions are closed, connections are detached, the communication with the database is stopped and safely disconnected.

When the server is started, various options must be entered:

- *Scenario*: It must be specified if the server is intended to manage a local or an Internet scenario, influencing the possibility for all users to create auctions or only one given user.
- *Port*: the TCP port for incoming connections.
- *Maximum number of simultaneous connections*: The number of connections the server will allow in parallel
- *Link to the database*: The administrator must specify a file that contains all necessary information to connect to the JDBC database on the (possibly remote) host
- *Debug log file*: A file that takes all error outputs produced by the system
- *Additional algorithms*: The system disposes of several built-in auctioning algorithms. The administrator may add new algorithms at start-up time.
- *Optional GUI*: Although not further supported, an optional GUI may be started with the server to display debugging information directly on the server machine.

3. Implementation

3.1 System Environment

3.1.1 Personal Digital Assistant

Personal Handheld Computers (PDAs, Personal Digital Assistants) vary in form and size, they run different operation systems (PalmOS, Windows CE) and are more or less powerful devices. In our case, the PDA of choice is the iPAQ by Compaq, which represents a perfect platform for the Java Client to run. It can also be easily equipped with a PC card for wireless communication. It disposes of a colour display, allowing to display images in high colour and usually 64 Mb of RAM.

3.1.2 Java

Java is a modern programming language, which provides the ability to run on different operating systems. We considered this to be a major advantage for our project. Concerning the client program (User as well as Admin Client), the only constraint is the limitation to AWT as graphical base component, as on PDAs only a limited version of Java is available. Compared to the new graphics library SWING, AWT has a rather limited pool of graphical components, though lacking no crucial functionality.

For the server, the multithreading operability of Java provides a perfect base to manage the client connections as well as the different auctions and the database connectivity in real-time. The disadvantage of a rather weak performance compared to other programming languages was not critical in our system, neither for the server nor for the clients.

The use of Java on both the client and the server system makes it possible to easily implement a proprietary protocol to manage the communication. This protocol is based on the exchange of PDUs containing different message objects. They support all necessary tasks like login and bid requests, the appropriate confirmations and error handling.

3.1.3 Wireless communication

With the upcoming trend to smaller and mixed devices an ever growing number of cables limit functionality and fun. The direction in computer interconnectivity is predictable: wireless connectivity. It makes it possible to stay connected to networks while moving around in a local area. This is a new dimension in comfort while being a low cost solution.

Whereas normal network connections are becoming better in quality, giving the guarantee of reliable links, wireless connections are vulnerable to interruptions. For this reason, we chose TCP as our protocol for the PDUs: It assures the correct transmission of data by built-in error detection and flow control. This is considered to be the best compromise between faster protocols like UDP and reliability. Connection to the server is therefore done by providing an IP Address coupled with a port number (Socket).

3.1.4 Internet Connectivity

Despite the fact that this project first was designed for local scenarios, later on, the goal was extended to manage worldwide auctioning. Reasons for this were that the local scenario could easily be adapted to the Internet, where TCP/IP is the general standard anyway for communication between terminals of any kind.

This scenario is characterized by a certain number of possibilities concerning how the clients can connect to the server. In general they connect to their local Internet Service Provider (ISP) and from there they reach the auction server by IP. Problems can occur when data packets need too much time on their way and timeouts happen.

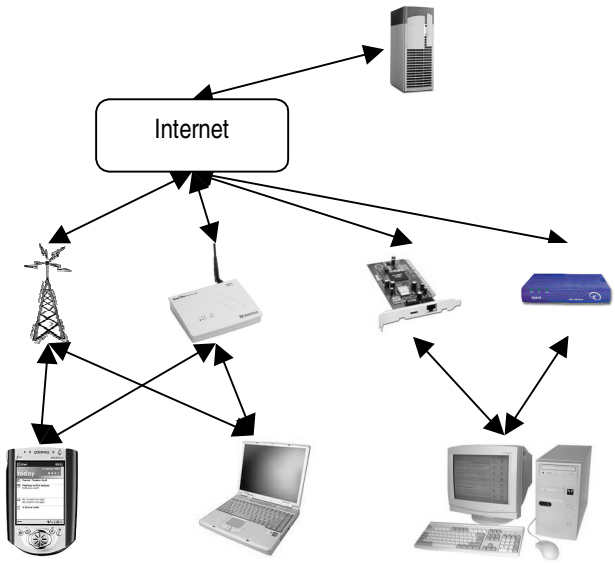

Figure 6 Internet Scenario

3.1.5 Operating Systems

As the server and the client both use Java, the underlying operating system is of minor importance. Usually, different behaviour of Java on different operating systems can be observed, but most of the problems concern the graphical user interface only.In our case, we have chosen Windows 2000 as operating system, rather because of the need of development software, than on a technical basis.

3.2 Software structure

Figure 7 and Figure 8 will give an overview of the classes used to implement the system.

3.2.1 Server

Figure 7 gives an overview about the server software structure. In the following, the most important classes are further described:

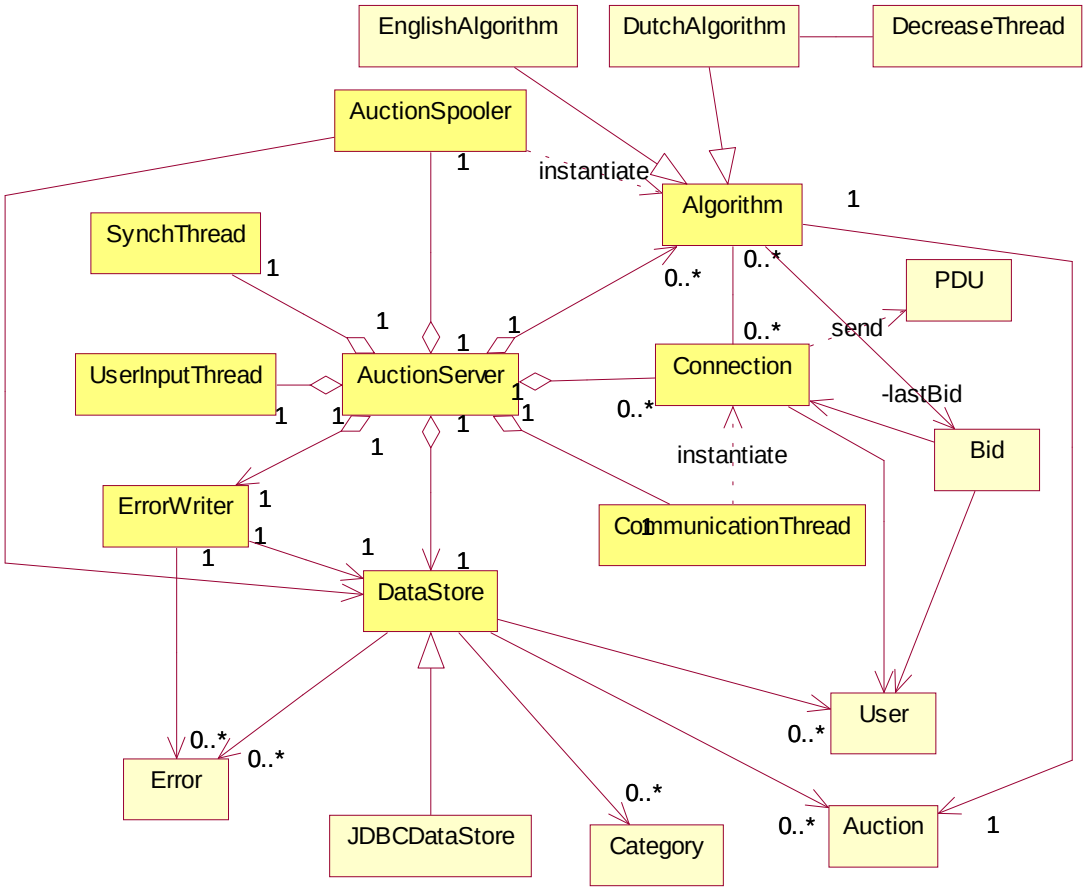

Figure 7 Server Software Structure Overview

3.2.1.1 Auction Server

The *AuctionServer* class is the main part of the server system that holds together the whole framework of the server architecture. It includes the *main* method to start the application and invokes all other necessary threads. It manages the active connections, the dynamically loadable *Algorithms* and the association between *Auctions* and *Algorithm* instances.

3.2.1.2 DataStore
The *DataStore* serves as a central place to store data. The different types of data are *Users*, *Auctions*, *Categories* and *Errors*. As an abstract class *DataStore* encapsulates the technical storage of the data. Currently, there is only one implementation of the *DataStore* class, the *JDBCDataStore*.

3.2.1.3 SynchThread
Several issues require synchronization between server and client. First of all, both systems may be located in different time zones or have at least different system times. Users should neither wait unnecessary long nor miss an auction caused by inaccurate time information, especially when conducting real time auctions in our case. A difference of few minutes or even seconds can be relevant. On the other hand we have to detect connection loss to allow fast reconnection and to inform the user. The *SynchThread* solves these problems by periodically sending synchronization messages to all connected clients. Clients can be sure to receive at least one message every x seconds. If not they can react accordingly by informing the user or automatically establishing a new connection.

3.2.1.4 UserInputThread
This component allows direct access to control the server via console. The thread waits for user input to shutdown the server, load new *Algorithm* classes or show the rudimentary graphical user interface. It runs as a daemon thread in order to allow the program to terminate while the "read" operation is still blocking.

3.2.1.5 Algorithms
The *Algorithm* class defines a kind of auctioneer that performs the logic processing related to a certain auction. Some administrative methods are implemented in the abstract *Algorithm* class: enter, leave, open, start and stop is identical for each *Algorithm*. *Algorithms* can access a list of all participants. Every participant is informed about other users entering or leaving. Changes in the runtime state when opening, starting or stopping are stored in the *DataStore* for search purposes. An auction is only started if enough users entered the auction. In case of an error or auction abortion users can be expelled from auctions.

3.2.1.6 AuctionSpooler
As auctions can be created with a start time in the future there needs to be a process that automatically starts auctions at the desired time. The *AuctionSpooler* performs this scheduling function. It permanently searches the *DataStore* for upcoming auctions. Auctions that could not be started due to server down time - those that are marked as upcoming but have a start time in the past - are rescheduled to a future time.

3.2.1.7 CommunicationThread
Communication is based on TCP/IP sockets. Java allows to establish connections using *Sockets* for each client. These *Sockets* can be created using a *ServerSocket* that listens on a specific IP port. Accepting connect requests results in a new *Socket* for each connection. These *Sockets* can be used in the *Connection* class to communicate with the client.

3.2.1.8 Connections
For each client a separate connection is necessary. The *Connection* class is based on Java's TCP/IP *Sockets*, which allow point-to-point communication in both directions. Protocol Data Units (PDUs) can be sent using an *ObjectOutputStream* or are received by an *ObjectInputStream*.

3.2.1.9 ErrorWriter

Server side errors should be stored in a central place. This class provides a simple method to log messages in the *DataStore*. If a file for logging purposes is specified as a start parameter of the application, messages are printed to a file or the screen. Messages can be of type error, warning, info or debug. As debug messages occur very often, they are never logged in the *DataStore* but can be printed to the file or screen. The administrative client allows viewing the stored messages.

3.2.2 Clients

A simplified class diagram of the client can be seen in Figure 8. In the following, the main functionalities of the classes are described in short.

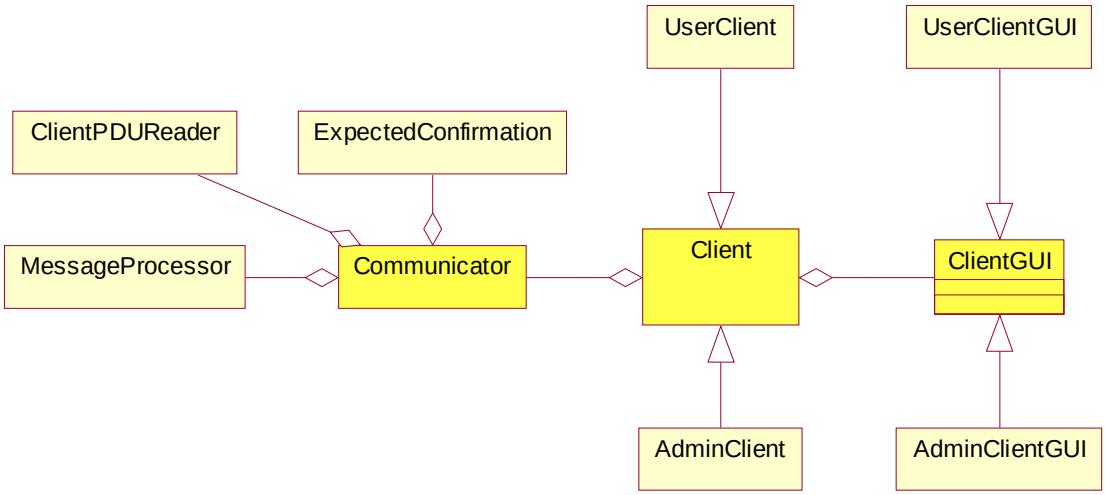

Figure 8 Client Software Structure Overview

3.2.2.1 Class Client

Class client is the central core of the client applications. Within this class all model data is maintained. That is –for example- user data, auctions, available algorithms and the like. The central client module communicates on the one hand with the user via the ClientGUI and on the other hand with the server using the capabilities of class Communicator. Abstract class Client is the super class for the two specialized Client classes UserClient and AdminClient. All methods that need to be implemented by a certain Client class are declared abstract.

3.2.2.2 Class UserClient

UserClient is an actual implementation of abstract class Client. It provides the functionality necessary to support the use of the client application by a regular user. That is creating auctions, participation including bidding, editing user data and the like.

3.2.2.3 Class AdminClient

AdminClient is the Client version used for administrative client application. It provides the methods used for administrative purposes. That includes viewing and maintaining the error log, user accounts, connections and so on.

3.2.2.4 Class ClientGUI

Analogue to the Client module, the GUI module depends on an inheritance relationship with super class ClientGUI. It declares all methods that have to be implemented by a GUI for a w3auction client application. The basic GUI is designed so it can run both on an iPAQ and on regular desktop or notebook PCs.

3.2.2.5 Class UserClientGUI

UserClientGUI is the sub class of ClientGUI that is to be used with the UserClient version of our application. It consists of a Frame and many screens in the form of Panels that are managed by a CardLayout layout manager. This is a perfect fit for the use of the GUI on an iPAQ. The screens include display screens for auction details, algorithm explanations and auction lists as well as input screens for user data, auction creation, login, connection and so on.

3.2.2.6 Class AdminClientGUI

AdminClientGUI is the GUI version for the administrator client application. It contains some features similar to the user client version and some additional screens for error, connection, category and user account management.

3.2.2.7 Class Communicator

Class Communicator is the part of the client application that is responsible for server communication according to the w3auction communication protocol. It can set up a connection to a specified server and port and monitor that connection in order to detect possible connection losses. It is able to permanently listen for independent server messages by the use of the ClientPDUReader and to send messages from the client to the server. For this case the Communicator contains a simple data structure called ExpectedConfirmation that is used to link a request (client) to a received confirmation (server).

3.2.2.8 Class ClientPDUReader

This class is the part of the communication module that is permanently reading messages from the network. After receiving a message it determines whether the message has been expected and if so notifies the Thread waiting for the confirmation. If not, the ClientPDUReader enqueues the message for the MessageProcessor and listens to the network again. The second purpose of the ClientPDUReader is the detection of network disturbances or connection losses. The reader thread “knows” that a server side synchronization message is sent at a constant rate. Therefore the client can assume a connection loss after not receiving a synch message for a certain time. In this case a separate Thread, ConnectionLossHandler, is started that interacts with the user to ask for further instructions and can reconnect to the server. Reconnecting includes restoring the current state of the client, i.e. logging in, re-entering the auctions the client was participating in and the like.

3.2.2.9 Class MessageProcessor

The MessageProcessor permanently looks up new messages in its queue and processes them. This is done by determining the type of message and afterwards calling the specific receiving method of the client.

3.2.3 Communication Protocol

The communication protocol is the essential part of the design that is responsible for the link between clients and server. Client and server architecture have to be designed around the protocol. The protocol defines the encoding of messages and the communication sequences.

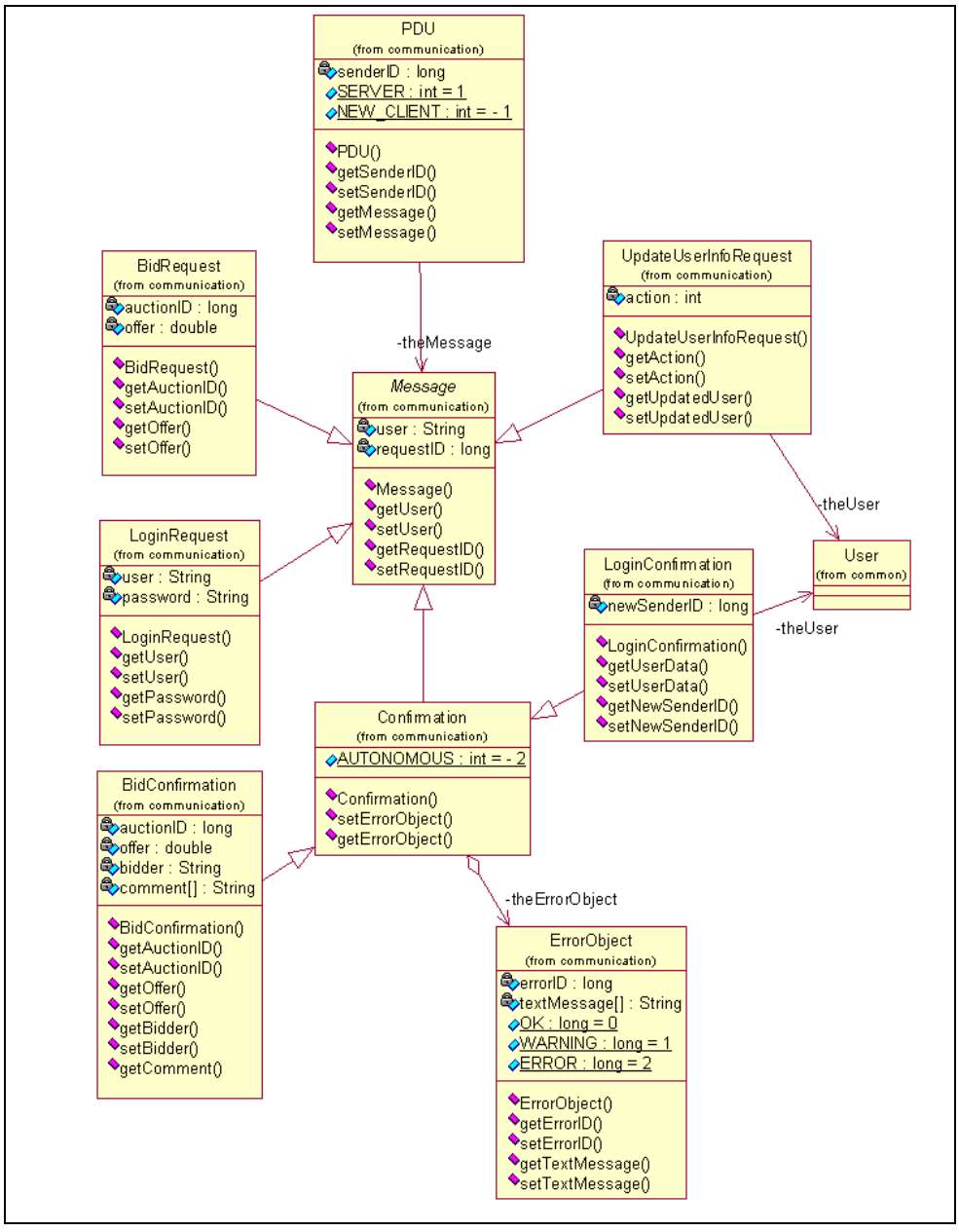

Figure 9: Excerpt from the communication protocol design

Figure 9 shows an excerpt of the communication protocol design. As communication is based on serialized objects, every class to be transferred implements the *Java.io.Serializable* interface.

The protocol allows only *PDUs* to be sent. These *PDUs* always include one *Message*. Client and Server try to read incoming *PDUs*, extract the included *Message*, and check which particular message type has been received. Classes that are defined in the communication protocol do not provide methods for activities on the client or the server side. They are only used passively to transfer data. This strong separation has the advantage that changes on one side do not affect the other.

Clients are intended to send only *PDUs* containing request messages, for example *LoginRequests*, *BidRequests* or *UpdateUserInfoRequests*. As a response to such a message, the server will reply with a confirmation or response. A request always belongs to a user and contains a *requestID* to identify the correct response message.
The *Confirmation* class defines that an additional object (i.e., the *ErrorObject*) is included to inform the user about success or failure by supplying multilingual text messages and an *errorID*, which indicates the success status. The *requestID* of the outgoing reply is set to the same value as of the incoming request.

Furthermore the server is allowed to send unsolicited messages to the client. For example a *BidConfirmation* for example is sent to all participants of an auction for every bid. The *requestID* is set to *AUTONOMOUS* then.

Some server messages contain additional objects such as the complete *User* record in a *LoginConfirmation*.

3.3 Graphical User Interfaces

In this section we describe the graphical user interfaces (GUIs) of the w3auction system.

3.3.1 User Client and Admin Client GUI

For the GUIs of the User and the Admin Client, we had to live with several major constraints. As mentioned before, because of the limited versions of Java on most of the PDAs, the graphical components had to rely on Java's much older graphical library AWT which provides fewer graphical components with a totally different appearance than the new library SWING.

In addition, the screen size of the *iPAQ* is rather small. The actual area itself where components can be displayed has a size of 240x280 pixels. Figure 10 presents a final version of the GUI for the user client. It is set for the final size, images have been added, an components have been grouped by functionality. It presents a scenario when the client is participating in an auction. The object to be sold is presented with an optional picture. The type of auction with its parameters is presented on the left and other participants are displayed in the box on the right.

Figure 10 Client GUI Final Version

3.3.2 Server GUI

The optional server GUI is mainly intended to support the debugging and testing of the server. Figure 11 shows an example of active connections and occurring errors. In addition, the GUI allows to search the database for users and categories. They can also be deleted interactively.

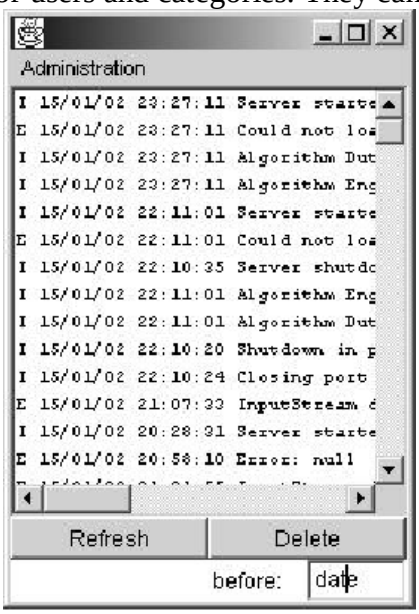

Figure 11 Server GUI

3.4 Database Management System

For our project we have chosen a MySQL database management system for the following reasons:

- this software is free,
- it delivers a very fast, multi-threaded, multi-user and robust SQL database server,

- it is compatible with various programming languages, MySQL client tools and APIs,
- it works on many different platforms,
- it supports many column types (FLOAT, DOUBLE, CHAR, VARCHAR,...)

Our database is composed by 4 tables : Users, Auctions, Categories and Errors.

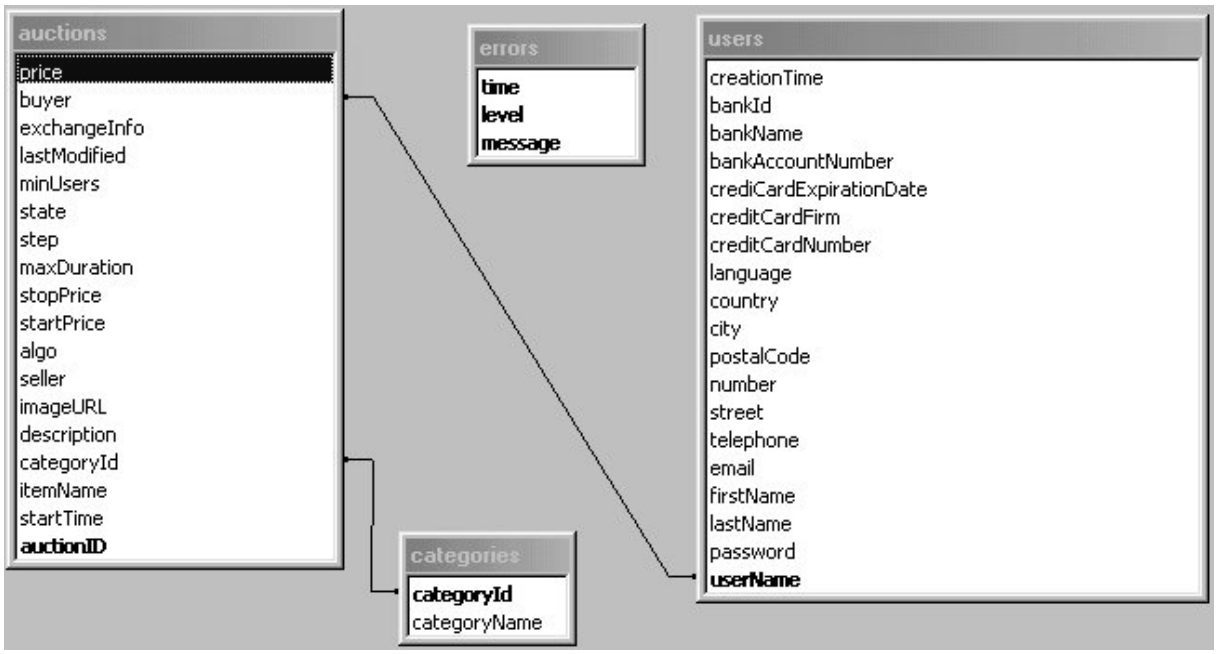

Figure 12 Database Architecture

To connect our application to the database we make use of a JDBC class. In this class we implement the JDBC MySQL, where we connect to the database.
In this class we have specified the path of the JDBC driver, the user name, the password (which permit to connect the database), the database name as a valid JDBC data source, the table names and all the methods who permit to initialize the database, insert new data into relations (e.g., when a user client creates an auction, item information and parameters are entered and the insert auction method is used), select data with different criteria (e.g. a user looking for particular data, select methods with parameters are used) and delete data from relations.

3.5 Wireless LAN

The wireless LAN environment is based on the IEEE 802.11 standard of the International Standards Organisation (ISO). It concerns the layers 1 and 2 of the OSI Model, as shown in Figure 13

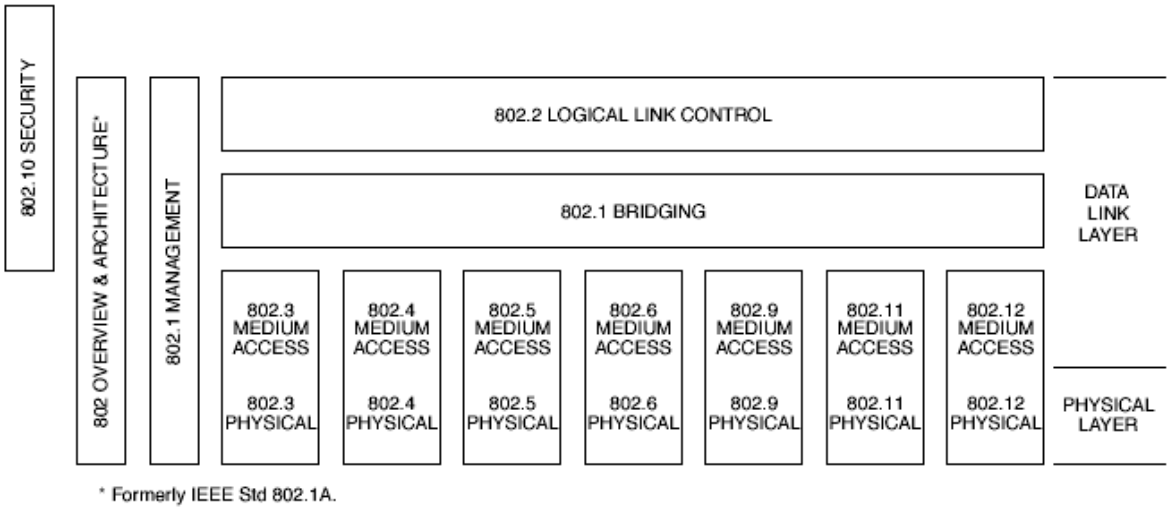

Figure 13 ISO Standard 802 for local area networks

It defines one base station, a so-called access point. Access points transmit network signals to the clients. The range varies and depends on floors and walls around, usually from 50 up to 300 meters.

An interesting feature is the possibility to connect several access points together in order to allow users to roam about without loosing their network connection. In this way, assuming there are enough base stations provided, a virtually unlimited area can be covered, so that there are no restraints regarding the size of a local scenario.

In general, the access points are themselves connected to a wired network through an Ethernet connection: they may also connect to an Internet Service Provider (ISP) through a built-in modem.

In terms of network technology, wireless access points act usually like layer-2 bridges, connecting different parts of a network. These parts have to be of the same kind, so clients are forced to use the same network protocols. Bridges perform a simple routing mechanism on the level 2 of the OSI standard model.

4. Experience

4.1 Experience with early users

At this moment, we don't dispose of enough experience with early users to draw any conclusions. Nevertheless it can be stated, that the system still lacks user friendly interfaces and coherent error messages and information throughout the program.

4.2 Implementer's experience

For the developers, this project was an incomparable experience regarding the use of new technologies, like modern handheld devices and wireless LAN. It provided the possibility to work closely with modern approaches of software design, like the description language UML and an object oriented approach in programming.

It was for most of us the first time to experience the complete life-cycle of a software project, with all aspects longing from analysis and implementation to database management, tests and a complete documentation.

5. Conclusion and Outlook

Our software system is operational and all necessary functions have been implemented. We have managed to provide easy to use user and administration clients.

Concerning the future development of this project, there is certainly a need to include appropriate add-ons concerning the aspect of security, as our software is at this time not yet protected against possible attacks over the net.

In order to make our system operational on a realistic real-world basis, it is necessary to provide an integrate settlement. User data must be verified, and checked for possible cheating, double accounts, invalid entries. After an auction has finished, the contact between the participants has to be established, the validation of auctions must be considered.

Electronic payment must be integrated into the system, which requires secure transfer and protection of banking information.

References

[1] Project Homepage
http://clio.unice.fr/~lesonna/site2/

[2] Teleseminar. Online Resource:
http://www.informatik.uni-mannheim.de/informatik/pi4/stud/veranstaltungen/ws200102/seminar_MA_Nice/

[3] Sun. **The Java Online Documentation**:
http://Java.sun.com/

[4] Nicolai Scheele: **Interaktive Lehre durch Einsatz mobiler Endgeräte. Diplomarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 2001**

[5] EBay Online Auctioning.
http://www.ebay.de