Free University of Bolzano/Bozen
Faculty of Computer Science
Bachelor of Science in Applied Computer Science

ON LINE AUCTION SYSTEM

Borella Michele 1270
Thesis Supervisor: Dott. Artale Alessandro
Academic Year 2003/2004

INDEX

CHAPTER 1

INTRODUCTION

1.1 Aims

The purpose of this project is to build an "on-line auction management system", a place for buyers and sellers to come together and trade almost anything.
In fact, the system consists in a web-portal where registered users can propose new auctions, place bids in order to buy the items on auction, send messages to other users and receive automatically news via e-mail (when they receive new offers for the proposed auctions, when an auction is over etc.).
Registration of users is preceded by a "pre-registration": to check whether users insert their real e-mail address, they receive an e-mail with an auto-generated secret code that they will be asked to type in a second moment to confirm the data (name, address, phone number etc.) they entered. Without this confirmation, a user cannot access the functionality of the portal.
Auctions have a name, a description, possibly a photo (of the related item) uploaded by users and an end period: users cannot place bids when the auction interval (start - end period) ends, but, in case there were no offers for an item, there is the possibility to extend the interval.
Moreover, administrators have the possibility to accept or refuse auctions proposed by users, to view information about users and items and to create, modify and delete the categories of auctions (auctions regarding cars, books, music stuff etc.).
The system is realized with a 3-tier architecture: a relational database that store the information regarding items, users, auctions and categories of auction; an application server that cares about the business logic of the system and the presentation layer that consists in the web browser where users can interact with the system.
With such architecture, the database is never directly accessed: for example administrators can change the data stored in the database without connecting directly to it but using their own browser.

1.2 Outline

The second chapter talks about the state of the art: an investigation of the existing auction systems and a description of the technologies used. The third chapter is dedicated to the requirement collection (done with the collaboration of the customer, in this case the company's tutor), the design of the system, done using UML use cases and class diagrams, and of the relational database, done with EER (extended entity relationship diagram) as modeling technique. In the forth chapter we will describe some peculiar aspects of the coding phase and we will provide a manual for the system. The fifth chapter is dedicated to the conclusions and "future works".

CHAPTER 2

STATE OF THE ART

Before starting with the project, we have considered some already existing on line auction systems and their functionality. Then we have decided what kind of system architecture and software technology to use.

2.1 Existing on line auction system

The first part of the project is an investigation of already existing on-line auction systems around the net. We considered three of the most famous auction web sites: eBay.com, asteinrete.com and onsale.com.
The table below describes the functionality offered to the users by this three big auction systems:

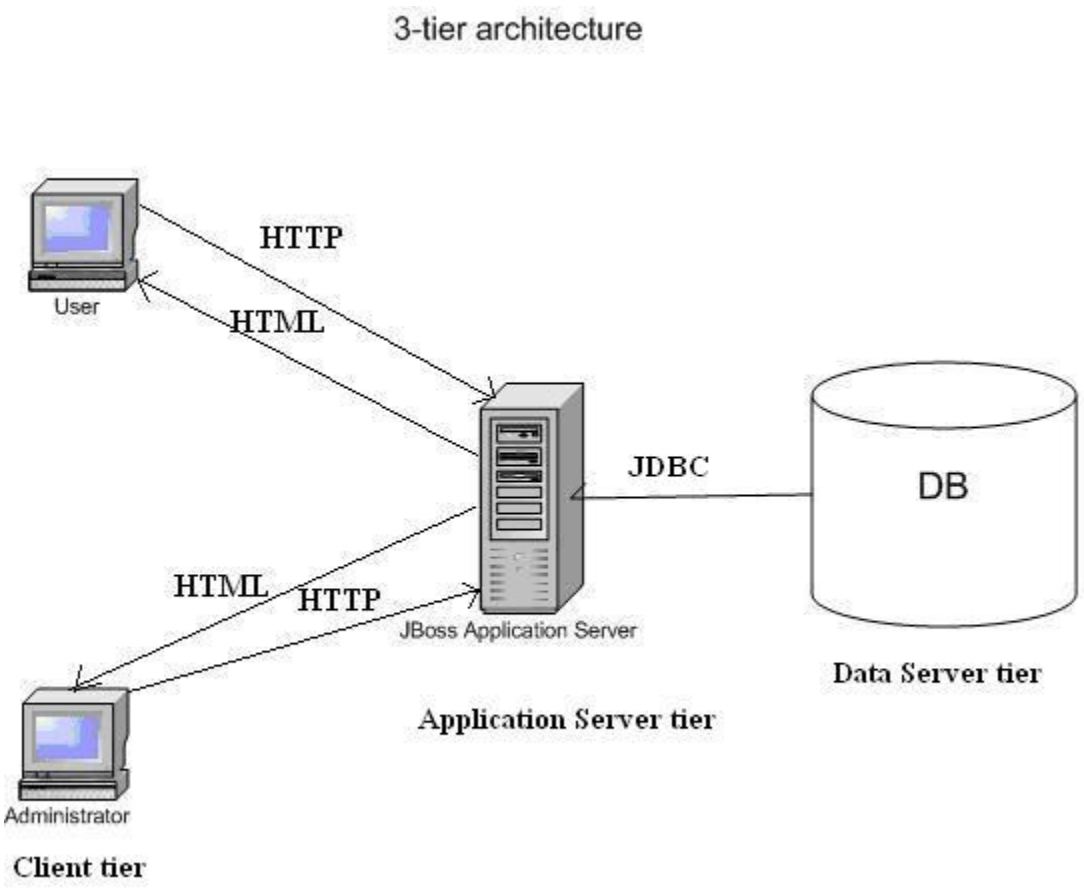
User Stories	**eBay.com**	**asteinrete.com**	**onsale.com**
Home Page	X	X	X
Registration	X	X	X
Login	X	X	X
Personal Page	X	X	X
Search	X	X	X
• Excluding a word	X		
• In a given category	X	X	X
• In a given city	X		
• Having price form € to €	X		X
Browse	X	X	X
Item Page	X	X	X
Bid	X	X	X
Post an auction	X	X	X
Help	X	X	X
Change Language			
Chat	X		
Administration	?	?	?

As shown in the table, all the three systems give the possibility to register, to login to the website and have a home page with a general description of the portal. They offer also a personal page, where each user can check the status of their auctions or of their offers.
Another characteristic of this portals is to have an item page, a page that describes each item on auction (with a textual description, a photo etc.).
The search functionality is also very important: in addiction to a normal keyword search, eBay offers also the possibility to search excluding a given word, search in a given category, search for auctions regarding a given city and to make price range (from € to €) search.
All the three systems give also the possibility to place a bid, to post an auction and have also some help pages that explains the aims of the portals and the functionality.

None of the portals has the possibility to change language (onsale.com is in English, asteinrete.com is in Italian while eBay.com is in English but there is also the Italian version at eBay.it) and only eBay.com has a chat room for the users.

2.2 Three-tier architecture

For the realization of the on line auction system we used a 3-tier system architecture as shown on this schema.



In such architecture, there are 3 main elements:

- The **client tier**, that is responsible for the presentation of data, receiving user elements and controlling the user interface.
- The **application server tier**, that is responsible for the business logic of the system. In fact, business-objects that implement the business rules "live" here, and are available to the client-tier. This tier protects the data from direct access by the clients. For the project, we used JBoss as application server.
- The **data server** tier, that is responsible for data storage. As data server, we used PostgreSQL, an open-source relational database.

2.3 JBoss and EJB

JBoss is a free, open source, application server that implements the complete Java 2 Enterprise Edition (J2EE) stack, including Java Server Pages (JSP), servlets and Enterprise JavaBeans (EJB) [4].
For the realization of the project we used the EJB technology, that provides an architecture for building distributed, component-based, enterprise-level J2EE applications.
EJB architecture handles the complexity of lower level services, such as component life cycle, state management, persistence, multithreading, connection pooling, transaction management and security [12].
EJB components, called enterprise beans, are scalable, transactional and secure. An EJB is made up of four parts:

- An **home interface**, that provides methods that control EJB lifecycle operations.
- A **component interface**, that defines the business methods exposed to EJB clients.
- The **bean implementation**, that contains the methods that perform business logic.
- A **deployment descriptor**, that specifies the declarative semantics that describe an EJB and the services it requires.

Moreover, there are two types of EJB: session beans, that are used to represent oprerations on persistent data but not the data, and entity beans that represent persistent data (usually records in a database).
The persistence of the entity beans is managed in two ways:

- **CMP** (container managed persistence) in which the Container uses the information in the deployment descriptor to automatically manage the mapping of EJB to a storage system reducing the amount of code in the bean.
- **BMP** (bean managed persistence) in which the Bean manages the mapping directly, usually with some JDBC code.

For the realization of this project, we used the CMP management technique.

2.4 Database and JDBC

As database for the project we used PostgreSQL, that is an open-source relational database. A relational database is a type of database management system (DBMS) that stores data in the form of related tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in many different ways.
The most common options when choosing a DBMS are Oracle [9], MySQL [7] and PostgreSQL [10]. Here is a table that summarizes the pros and cons of these database systems:

	Oracle	MySQL	PostreSQL
Pros	- Fastest commercial DBMS -Writers never blocks readers - Transactions, rollbacks and subselects support.	- Open source - Low machine requirements - Easy to setup	- Open source - Easy to administer - Transactions and rollbacks support
Cons	- Not open source (Oracle licenses are expensive)	- No transactions, rollbacks and subselects.	- No support to fault tolerant installations.

The main difference between these three systems is that MySQL and PostgreSQL are open source while Oracle is not. Oracle offers more advanced functionality than the other two systems: it is the fastest, supports transactions (sets of basic operations considered as single operations) and has enterprise-level data protection and distribution capabilities, such as full-scale clustered replication. On the other hand, of all open-source database solutions available, PostgreSQL is the most impressively complete, boasting transaction support, ANSI SQL92 compliance, a query optimizer and essential data integrity controls etc. [5]. As database system for the on-line auction portal we choose PostreSQL: it has not support to fault tolerant installations, but, in our environment and for our needs, there are no disadvantages in using it.
JBoss can "communicate" with the data server tier using the JDBC API that is the industry standard for database-independent connectivity between the Java programming language and a wide range of database. In fact, the JDBC API provides a call-level API for SQL-based database access and makes possible to do three things:

- Establish a connection with a database or access any tabular data source.
- Send SQL statements.
- Process the results [12].

2.5 JSP

For the realization of the web pages of the on-line auction portal, we used the JSP (Java Server Pages) technology, which is also implemented by the JBoss application server.
JSPs allow developers to combine HTML and Java code in the same document using special tags that identify Java code.
JSP applies no restrictions to the Java language and its development is very simple. In fact, JSP pages are simply HTML pages with special tags including Java code performing elaborations and providing HTML code to be included into the page.
The first execution of a JSP includes three steps:

- Translation of the JSP code into a JavaServlet
- Compilation of the Servlet
- Execution of the Servlet.

The following executions include only the last step. Moreover, it is possible to pre-compile all JSP in order to reduce the time required by the first execution [12].

2.6 Eclipse and MyEclipse

For the programming part of the project we used Eclipse, that is a very powerful open-source integrated development environment (IDE). This IDE offers several services to the developer: it has compiler aware editing; syntax errors are highlighted when they are made, as are simple semantic errors such as missing declarations. Eclipse supports method completion, shows class interfaces concisely in graphical notation and supports interactive exploration of a program, through features such as fly-over name resolution. In addition, Eclipse supports Software Engineering principles such as packaging, debugging, testing, refactoring and versioning [2]. Another very important advantage of using Eclipse is that several plug-ins can be installed on it, like MyEclipse, that is particularly useful when working with Enterprise JavaBeans.
In fact, MyEclipse offers affordable tools for Java and J2EE (Java 2 Enterprise Edition) developers. Features include:

- Web development tools, such as editors with code completion and syntax coloring for JSP, HTML, XML and CSS. They include also JSP syntax validation and native JSP debugging.
- Productivity wizards, such as EJB wizards, archive based deployment (EAR and WAR) and Sync-On Demand automated deployment of applications to integrated servers.
- Application Server integration, including integrated controls for starting and stopping application servers and full hot swap debugging support for deployed applications [6].

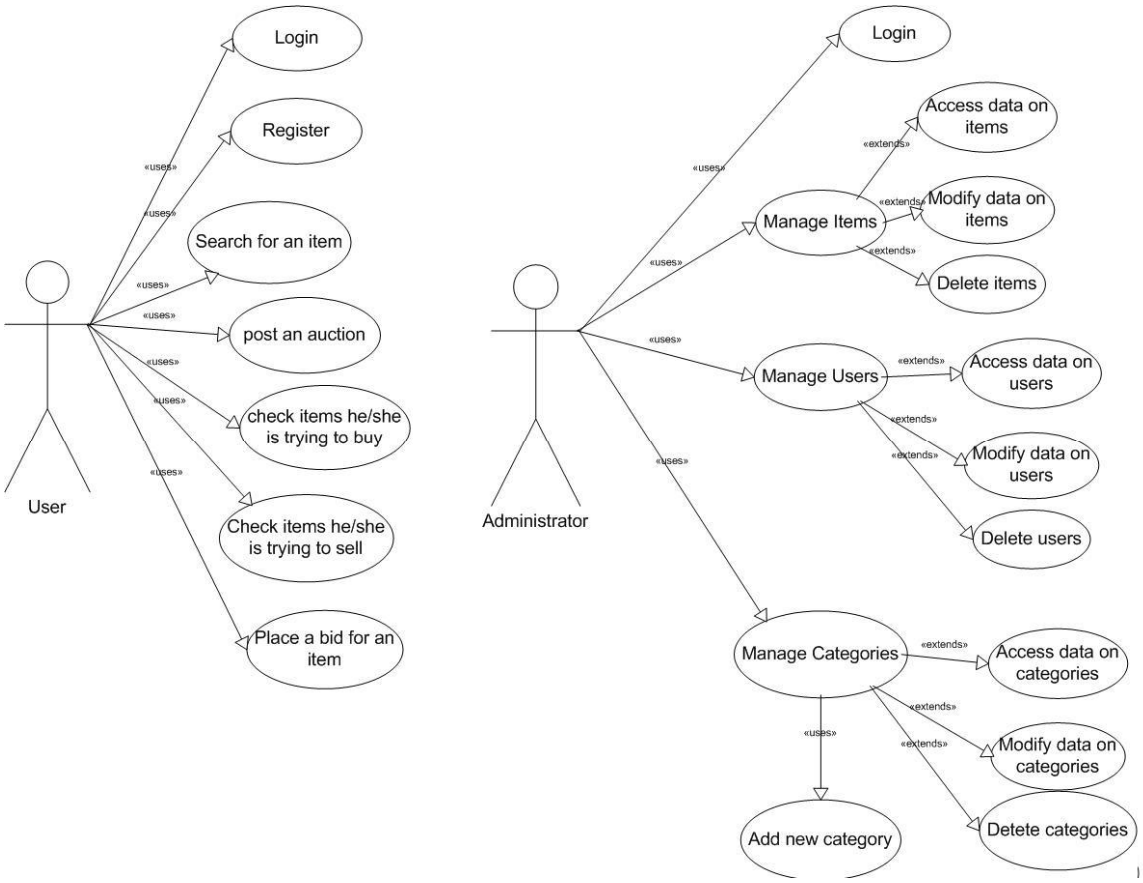
CHAPTER 3

FUNCTIONAL REQUIREMENTS AND SYSTEM DESIGN

The first step of the project is to find the functional requirement of the on line auction portal with the help of techniques like Use Cases and User Stories. After having found the functional requirements, the project goes on with the system design using the following techniques: the UML class diagram, the EER diagram for the database design and the page flow diagram.

3.1 Use Cases

The first step for the functional requirement collection are the use cases. Use cases are "a description of set of sequences of actions, including variants, that a system performs that yield an observable result of value to an actor". They are used in order to: design system from user's perspective, communicate system behaviour in user's term and enumerate all externally visible behaviour [11].
Here are the use cases for the on line auction system project (there are two actors for the system: a normal user and an administrator).



As shown on the schema, a normal user can register to the system, browse the available items, post his/her bid and post a new auction. The administrator, on the other hand, can insert and modify available data about items, users and categories of items.

3.2 User Stories

After collecting all use cases, user stories can be written. A user story "is the smallest amount of information (a step) necessary to allow the customer to define (and steer) a path through the system" [8].
The user stories are divided into 2 main categories: user side (stories for the general users) and administration side (the stories for the administrators of the system).

User Side Stories:

Home Page:
The home page is the entry point to access the main services:

- Register
- My personal page
- Search
- Help

The home page shows also a list of categories to simplify items searching and the latest auctions.

Registration:
The registration page allows user to provide his/her personal data (name, address, date of birth, fiscal code, email address, phone number, userID, password) and receive a userID and a password. UserID and password allow the user to access to his/her personal page, to take part to the auction and to post a new auction.
It performs basic checks on entered data and provides user registration or an error message if the userID and/or user fiscal code are already present in the system.

Login
Every time the user tries to access to non-public areas (personal page, bid, post an auction…), he/she is asked to provide his/her personal ID and password. These are entered through a form. If userID and password are correct, the user is logged in and is no more asked to login throughout the session. Otherwise an error message is raised.

Personal page:
To access the personal page the user is asked to login, or to register. The personal page keeps track of all the items the user is presently trying to buy and has bought in the recent past and of all items he/she is trying to sell. From this page it is also possible to post a new auction.

Browse:
The user can browse the auctions selecting among several categories of items (e.g. cars, books etc.). The results will be shown in a table and the user can sort them by price, by auction interval (by lasting period of the auction).

Search:
The user can search for items on auction providing a key word by different criterions:

- Excluding a word
- In a given category
- In a given city
- Auctions having price from a given value in Euro to another value

Both registered and unregistered users can access to this service.

Item page:
Item characteristics are shown in the item page. From this page the user can place a bid pushing the button “PLACE A BID” and view the chronology of the bids.

Bid:
The user that makes a bid is asked to login if not already logged. If the bid is accepted by the system, the item is listed in the user personal page. Bids can only be placed during the auction interval and they must be at least one minimum increment bid above the current price.

Post an auction:
From his/her personal page, the user can post an auction from a specific form, providing the characteristics of the item he/she is willing to sell. If the auction is accepted by the system, the item is listed in the user personal page and other users can place a bid for it.

Help:
The system must provide help pages in order to explain how to perform all possible actions, such as registering, searching items, posting an auction etc.

Change language:
The user can change from every page the language in which he/she wants to read the pages with a combo box (Italian/English).

Chat:
The user can enter in a chat room using as nickname his/her userID and as password his/her personal password. In the chat room can send message to all users or send private messages to only one user.

Messages:
The user can send message to other users with a text and a topic and the message will be sent via email to the receiver. It is also possible to view the received messages on the website and answer to them.

Administrator Side Stories:

Administrator Page:
From the login page, providing his/her administratorID and password, he/she can access the administrator page, that shows the administrator menu to access all the administration activities (manage items, manage users, manage categories).

Manage Items:
The administrator can access all data about items stored in the database and also delete them, but not modify the characteristics of the items (initial price, description etc.).

Manage Users:
The administrator can access and modify all data about users stored in the database and also delete them.

Manage Categories:
The administrator can access and modify all data about categories stored in the database, add a new category and also delete them. The administrator can delete a category if and only if no items are associated to that category; otherwise an error message is raised.

Accept / refuse auction:
The administrator can control the new auctions posted by the user and decide whether to accept or to refuse them.

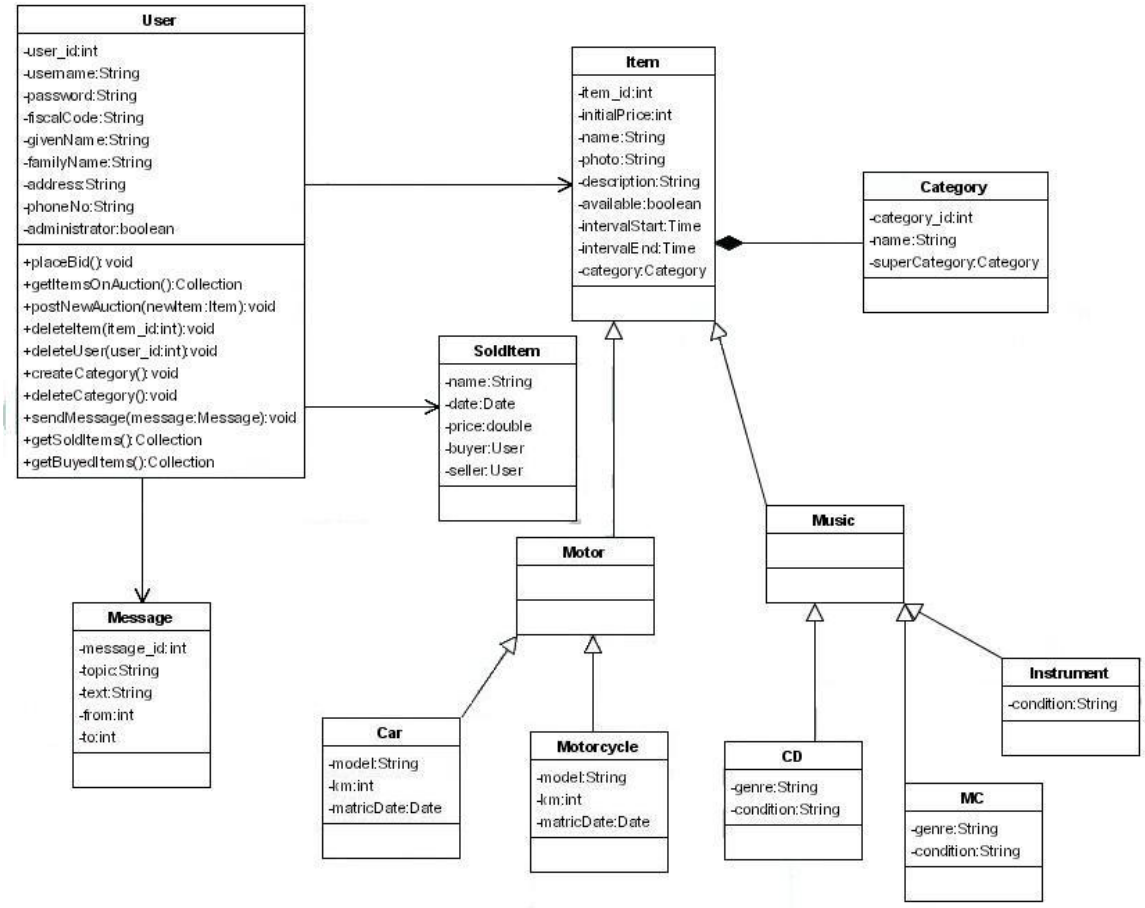
Then, we write a table that assigns to each user story a priority and a risk level:

User Stories	Priority	Risk
User Side Stories		
Home Page	Must	Low
Registration	Must	Medium
Login	Must	Medium
Personal Page	Must	High
Search	Must	Medium
Excluding a word	Should	Medium
In a given category	Must	Medium
Having a price from € to €	Should	Medium
In a given city	Should	Medium
Browse	Must	Medium
Item page	Must	Medium
Bid	Must	High
Post an auction	Must	Medium
Help	Should	Low
Change language	Could	Medium
Chat (spike)	Could	High
Messages	Should	Medium
Administration Side Stories		
Administrator Page	Must	Low
Manage Items	Must	Medium
Manage Users	Must	Medium

Manage Categories	Must	Medium
Accept / refuse auction	Must	Medium

3.3 UML Class Diagram

The next step of the design phase is to draw an UML Class Diagram of the system. Since the programming language of the system is an object oriented one, an UML Class Diagram is particularly adapted to show the classes of the system, their inter-relationships, and the operations and attributes of the classes [1].
Here is the class diagram of the project.

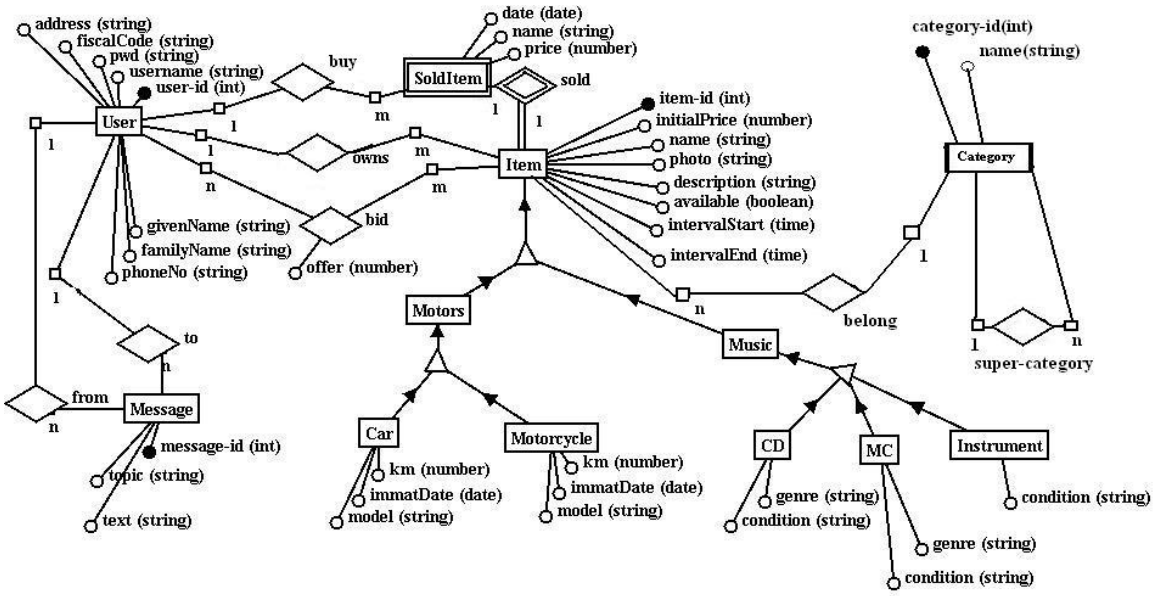


The class User contains several parameters that are information about a normal user that browses the system (username, password, name, surname…); the parameter administrator is a flag that indicates whether the user is an administrator or not. This class performs the operations of a normal user (search for an item, get the items on auction, post a new auction, send messages to other users) and those of a system administrator (create a new category, delete a category, delete an item, delete an user). The class Message has a text, a topic and the user_id of the sender and of the receiver. The class Item contains several information about the item on auction (the name of the item, its photo, a textual description of it..) and about the category it belongs to.
A category has a name and, eventually, a super-category it belongs to: a category can belong to another category (e.g. category sport car belongs to category car).

Class Motor and Music are subclasses of Item. Car and Motorcycle are subclasses of Motor: they inherit the properties of class Item and they have their specific attributes (model, km, matriculation date). CD, MC and Instrument also inherit properties form Item and have their own specific attributes (genre, condition).
Class SoldItem represent an already sold auction item, its attributes are the name of the item, the final price, the date in which it was sold and its buyer and seller.

3.4 EER Diagram for Database Design

After having drawn the UML Class Diagram for the On Line Auction System, it is clear what kind of data should be stored in the database. Since PostgreSQL is a relational database, the EER modelling approach is very useful to design the database schema since it maps well to the relational model and the constructs used in the ER model can easily be transformed into relational tables [13].
Here is the EER Diagram for the database of the system.



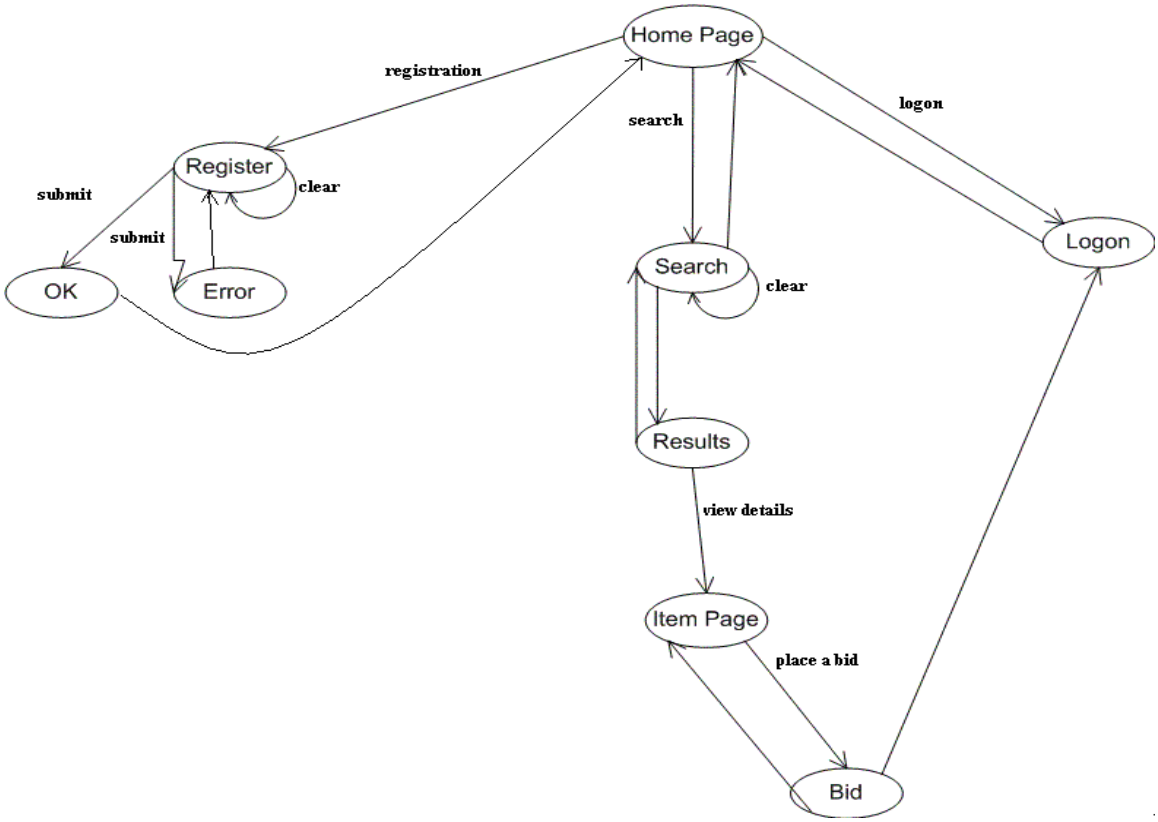
As shown on the schema, there are several entities (user, item, message…) with their own attributes and relations. Motors and music are particular kind of items (they extend the entity item). Each User owns one or more items and can bid, making an offer, for one or more of them. Each Item, on the other hand, is owned by exactly one User and belongs to one category. A Category can have one super-category. A Message is sent from one User to exactly one another User.
Car, motorcycle and CD, MC, instrument are respectively sub-entities of motors and music. In addition to the attributes inherited from item, they have their specific attributes (km, matriculation date, genre..).
Sold-Item is a particular kind of entity called “weak entity”. A weak entity, in contrast to regular entity, does not have key attributes of its own. When an Item is sold, it becomes a Sold-Item.

From the EER diagram, it is easy to get the tables of the database using the "Elmasri" algorithm [3], here are the tables of the on line auction system database. For EMJ conventions, each entity has an integer ID as primary key.

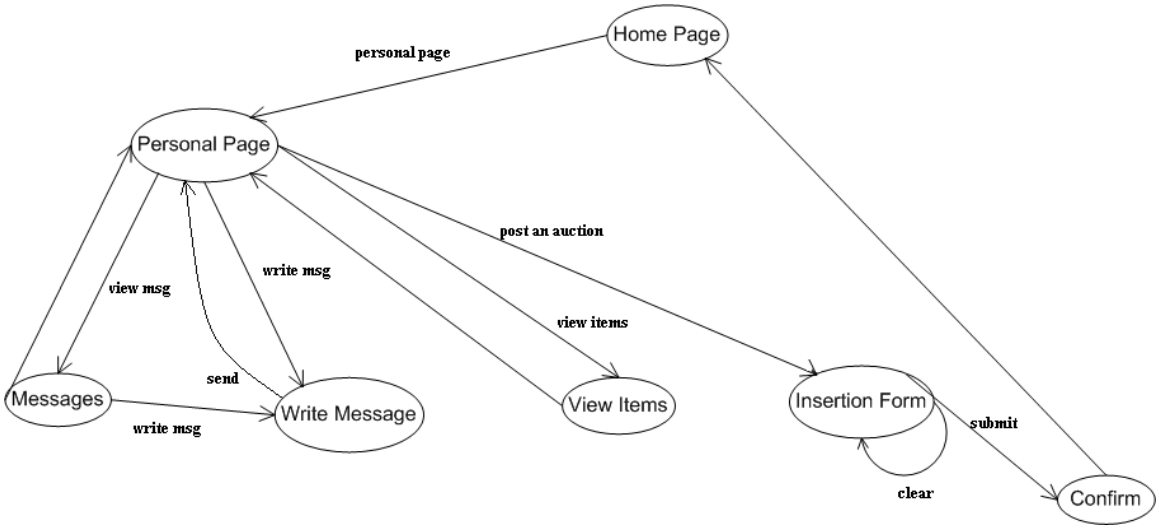
```
USER(user_id(PK), username, pwd, administrator, givenName,
familyName, phoneNo, address, civicNo, city, state,
fiscalCode)
MESSAGE(message_id(PK), from(FK USER), to(FK USER), topic,
text)
ITEM(item_id(PK), initialPrice, name, photo, description,
available, intervalStart, intervalEnd, owner(FK USER))
BID(bid_id(PK), item(FK ITEM), offerer(FK USER), offer)
MOTORS(motors_id(PK, FK ITEM))
MUSIC(music_id(PK, FK ITEM))
CAR(car_id(PK, FK MOTORS), km, immatDate, model)
MOTORCYCLE(motorcycle_id(PK, FK MOTORS), km, immatDate,
model)
CD(cd_id(PK, FK MUSIC), genre, condition)
MC(mc_id(PK, FK MUSIC), genre, condition)
INSTRUMENT(instrument_id(PK, FK MUSIC), condition)
SOLDITEM(item_id(PK), name, date, price, buyer(FK USER))
CATEGORY(category_id(PK, FK ITEM), name, super-category(FK
CATEGORY))
```

3.5 Page Flow Diagram

Since the on line auction portal consists of web pages, it is useful to draw a page flow diagram. A page flow is a diagram that visually organizes the flow and actions of the web pages [9]. Here is the page flow that represents the path, for a normal user, to register to the system, search for an item and place a bid and make a logon.

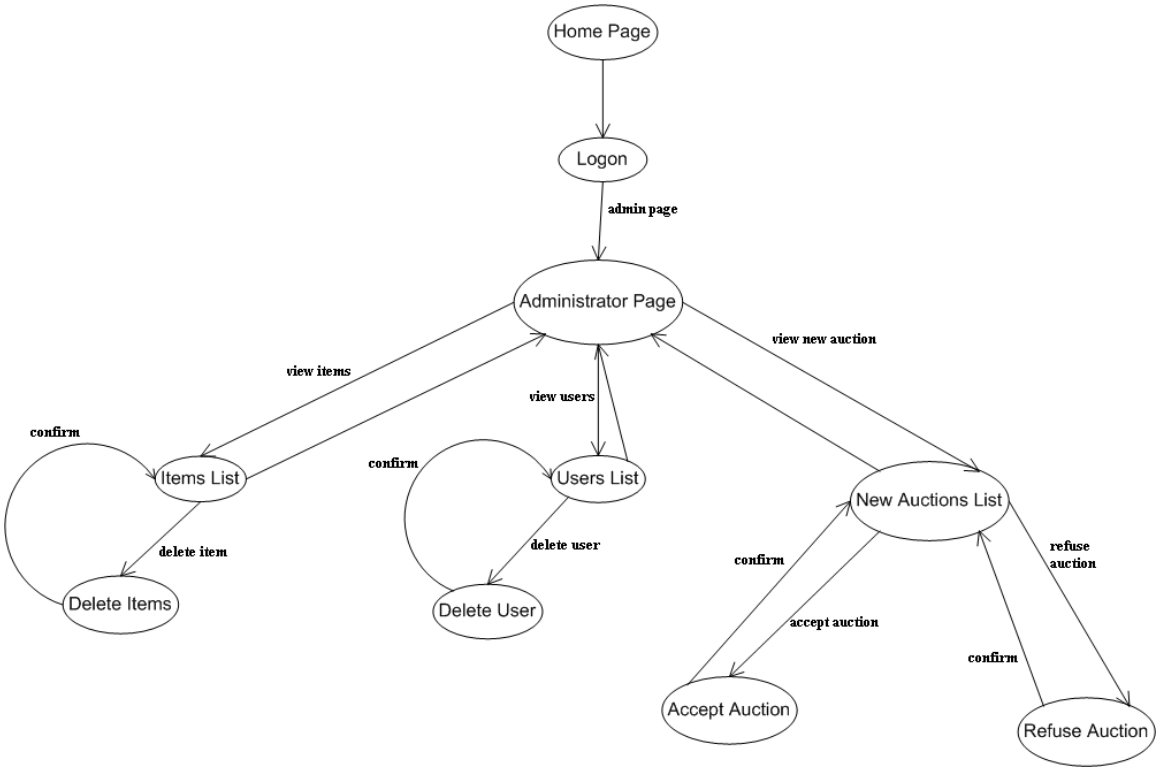


As shown on the diagram, from the home page it is possible to reach the registration page and enter the personal data of the user. If the data are right, the user can reach a confirmation page, otherwise (e.g. the username that he/she choose is already in use) an error page. From the homepage it is also possible logon inserting username and password and reach a search page to search for an item. The result of the search are given in another page where there are the links to see the item's detail (description, photo…) and place a bid for that item.
Here the page flow for write messages to other users, view the items on auction and post a new auction.



From the home page it is possible, for logged users, to reach their personal page. From the personal page it is possible to write message to other users, read the received messages, view their own items on auction and post a new auction with an insertion form.

Finally, the page flow diagram for the administration activities.



When a user is a system administrator, after having inserted username and password, can reach the administration page. From the administration page it is possible to view all data about users and items present in the database and manage them. For an administrator it is also possible to view a list of the new auctions proposed by the user and accept or refuse them.

CHAPTER 4

SYSTEM IMPLEMENTATION

In this chapter will describe a very useful tool used during the coding phase (XDoclet) and its benefits. Then we will provide a manual for using the on-line auction system.

4.1 XDoclet

XDoclet is an open source code generation engine that enables attribute-oriented programming for java. This means that more meaning to the code can be added by inserting meta data (attributes) into the java sources. This is done in special JavaDoc tags. XDoclet parses the source files and generates many artefacts such as XML descriptors and source code from it. These files are generated from templates that use the information provided in the source code and its JavaDoc tags [15].
Here is an example of how did we use the XDoclet tool for creating the Category entity bean.

```
package it.nrkauction.ejb;

import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.RemoveException;

/**
 * @ejb.bean                name = "Category"
 *                          type = "CMP"
 *                          cmp-version = "2.x"
 *                          display-name = "Category"
 *                          description = "Category EJB"
 *                          view-type = "local"
 *                          jndi-name = "ejb/CategoryHome"
 *                          local-jndi-name = "ejb/CategoryLocalHome"
 *
 *
 * @jboss:table-name        TBL_CATEGORY
 *
 * @ejb:finder              signature = "Collection findAll()"
 *                          unchecked = "true"
 *                          query = "SELECT DISTINCT OBJECT(category) FROM
Category category"
 *                                  result-type-mapping = "Local"
 *                          generate = "true"
 *
 * @ejb:util                generate="physical"
```

```
 */
public abstract class Category implements EntityBean {
```

Before the class declaration, there is a special JavaDoc piece with some XDoclet attributes. The first attribute (@ejb.bean) describes the name of the entity, its type (can be CMP or CMR) and the name of the related home interface (CategoryLocalHome).
The second attribute (@jboss:table-name) indicates the name of the database table related to that entity (in fact, each entity bean is related to a database table).
The attribute @ejb:finder represents a sort of query, it is written in EJBQL, a language very similar to SQL. In this case it tells the XDoclet engine to create a method called "findAll()" that returns an object of type Collection with the complete list of all the categories present in the table.
After having started the XDoclet code generation engine, it creates automatically three artefacts:

1. The home interface with the method findAll():

```
public interface CategoryLocalHome extends javax.ejb.EJBLocalHome {
  public static final String COMP_NAME="java:comp/env/ejb/CategoryLocal";
  public static final String JNDI_NAME="ejb/CategoryLocalHome";

        public it.nrkauction.interfaces.CategoryLocal create(int category_ID ,
                                                             java.lang.String    name ,
                                                    it.nrkauction.interfaces.CategoryLoal
                                                    supercategoy) throws
                                                    javax.ejb.CreateException;

  public java.util.Collection findAll() throws javax.ejb.FinderException;

}
```

2. The component interface with the business methods:

```
public interface CategoryLocal extends javax.ejb.EJBLocalObject{

  public int getCategory_ID( ) ;

  public java.lang.String getName( ) ;

  public void setName( java.lang.String value ) ;

  public it.nrkauction.interfaces.CategoryLocal getSupercategory( ) ;

  public void setSupercategory( it.nrkauction.interfaces.CategoryLocal owner ) ;

}
```

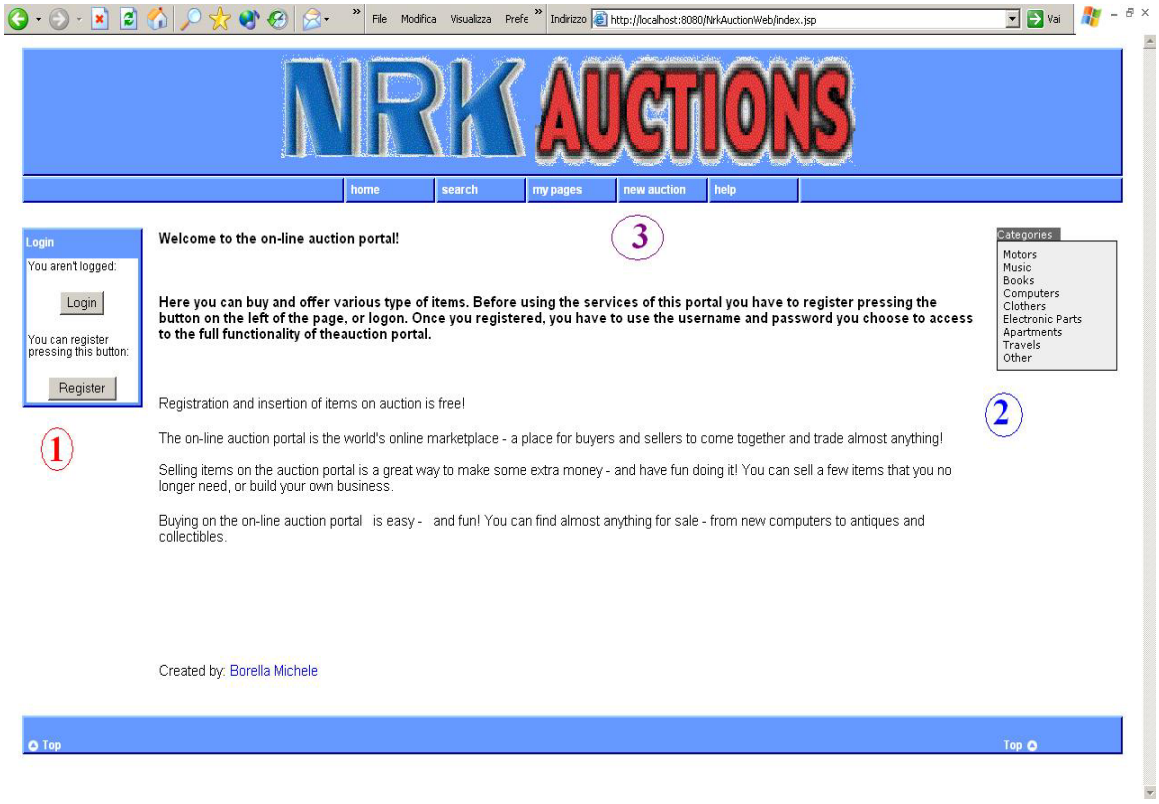
3. The XML deployment descriptors (see Appendix-Deployment descriptors).

The use of XDoclet has several benefits:

- The developer has not to worry about the deployment meta-data whenever he touches the code, it is continuously integrated
- An EJB typically consists of more files, so it is easy to lose track. With XDoclet the developer has to maintain only one of them, and the rest is automatically generated.
- The developer can dramatically reduce the development time and can concentrate on business logic while XDoclet generates the rest of the code.

4.2 Manual of the System

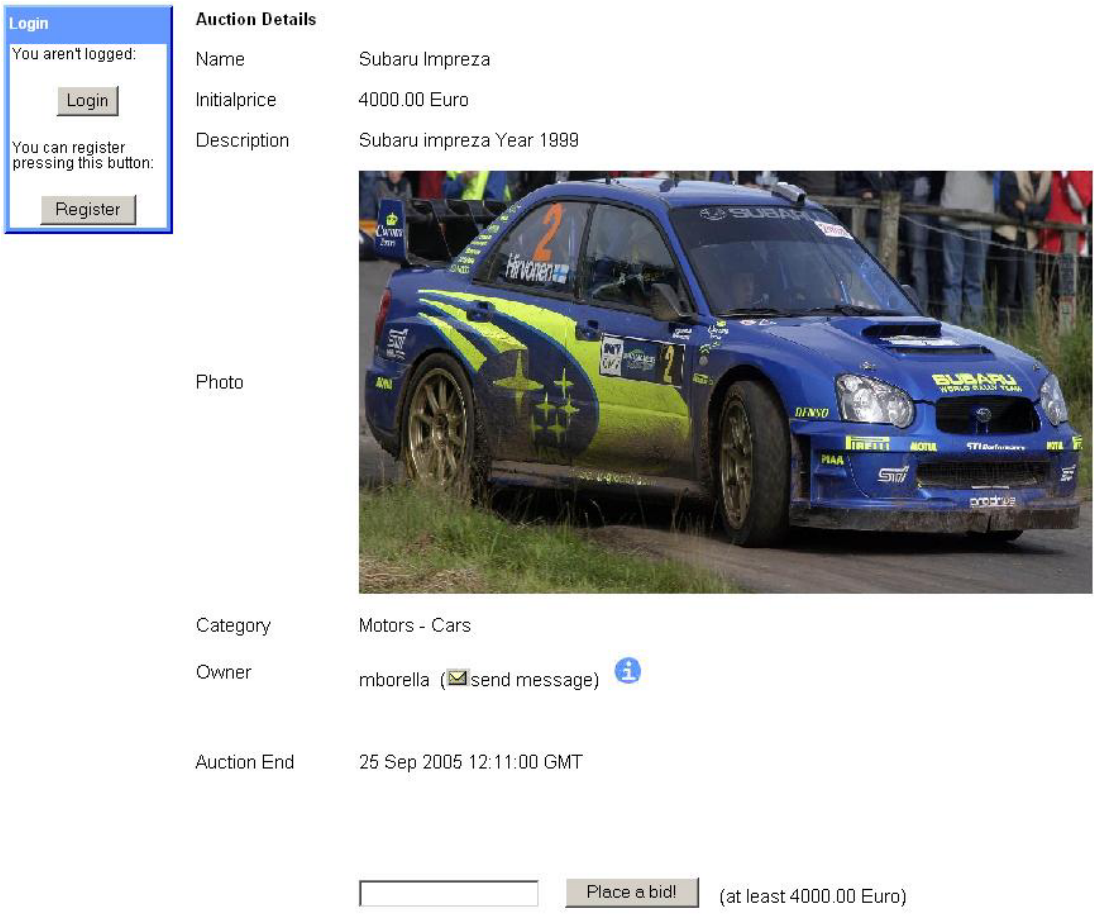
Here we will give a short guide of the system. The first page of the on line auction portal is the home page as shown here below:



The homepage contains several information about how the portal works and several links to other pages. On the left of the page there is a menu (1) from where it is possible to logon inserting username and password or, for non registered users, to register to the auction portal. On the right of the page (2) there is a list of the available categories of auctions. The user can browse the various items belonging to that category by clicking on one of it. On the upper part of the homepage (3) there is a navigation menu where the user can access to the search page (to search for an item), to his/her personal page, to the new auction proposal page (to post a new auction proposal) and to the help pages of the website. Only registered user can access to the personal page and to the new auction proposal page.
In order to access to the full functionality of the web portal, the user has to register inserting his personal data on the form of the registration page as shown here below:

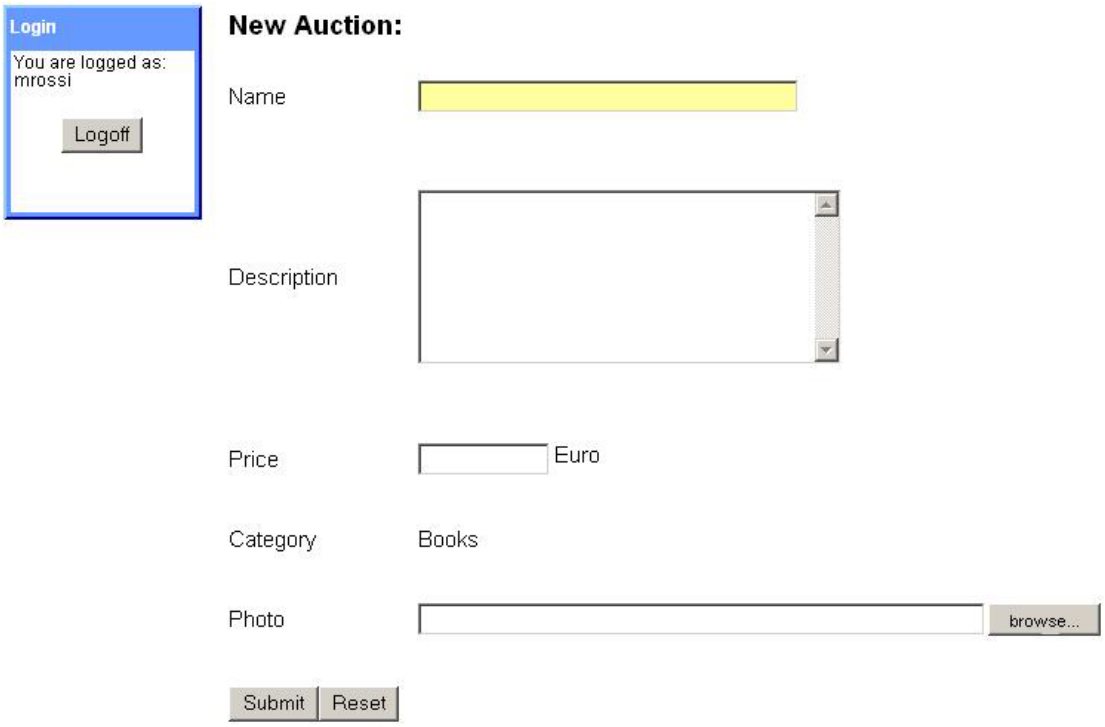
In this form the user has to enter his/her personal data (first name, last name, address etc.) and has to choose a username and a password. After entering the data, the user receives an email with a secret key string that he/she has to enter to confirm the registration. This is done to verify that the email address that the user inserted is really his/her email address and not a fake one.
To search for an item on auction, on the search page the user can specify a keyword and a category of item as shown here below:

After having inserted a keyword and selected a category, the user gets a list of results (if there are items that match the search criteria). Moreover, selecting an item from the list, it is also possible to read the auction details as in the page below:

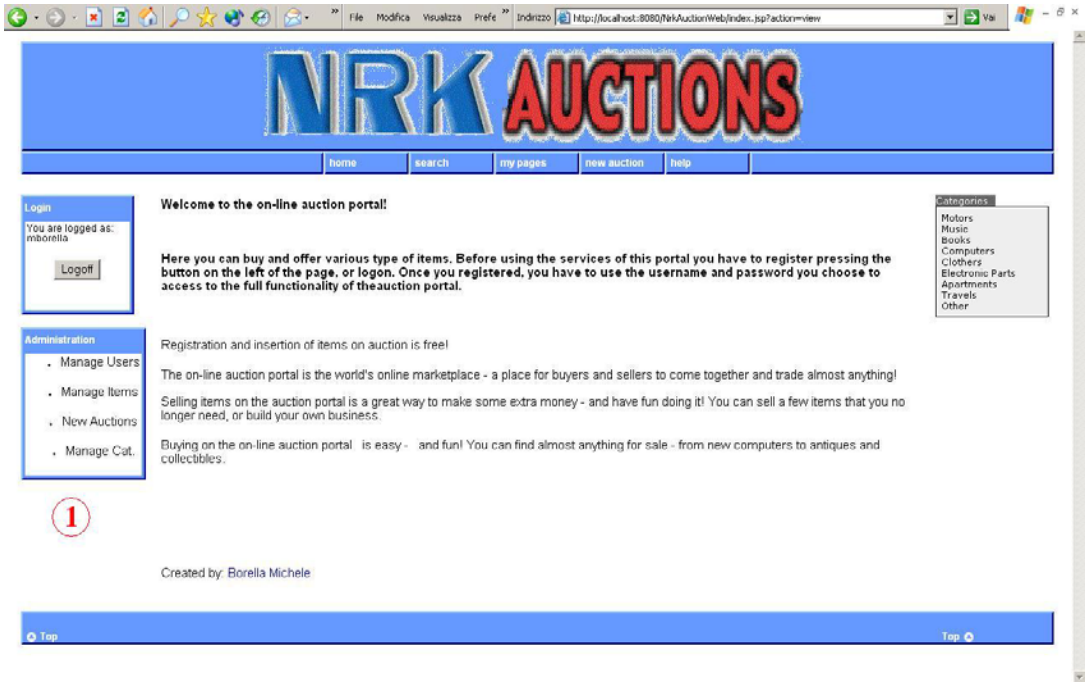


In this page there are information about the item (description, price, photo, auction end date) and from here it is possible for registered users to send messages to the owner of the item, view information about the owner and, eventually, place a bid. The owner is notified via email about every bid placed for his/her auction.
A registered user can view his/her received messages and the status of his/her auctions in the personal page:

A registered user can also propose a new auction by selecting “new auction” from the horizontal menu. After having selected the category of auction, the user can fill the form with the data regarding his/her auction:



Here the user can insert the information regarding the auction (name, description, initial price, possibly a photo…) and submit his/her proposal.
When an administrator is logged, in addition to the normal menu on the home page, it appears also an administration menu as shown here below:



On the left of the page (1) there is a menu from where it is possible to access to the data stored in the database regarding users, items and categories (manage users, manage items, manage categories) and delete or modify them. It is also possible to see the list of the new auction proposals posted by users (new auctions) and accept or refuse them (in fact, an auction has to be accepted from the administrator in order to be available on the portal). Here is, for example, the administration page to manage users:

From this page, it is possible to see a table with the complete list of users of the system. The administrator can see all data regarding a user pressing one of the selection buttons (1) or can delete a user from the portal pressing one of the delete buttons (2). It is also possible to search for an username with the search form in the upper part of the page (3). Moreover, an administrator can insert a new category of auction, or modify or delete one of them as shown in the page here:

From this page it is possible to see all categories of auction present in the system. The administrator can see and modify the data regarding a category with the edit buttons (1) or can delete a category pressing one of the delete buttons. The administrator can also insert a new category with the button on the upper part of the page (4) or search for an existing category with the search form (3).

CHAPTER 5

CONCLUSIONS

Here it will be explained what we have realized with respect to the requirements, the problems found during the development of the system, what we have learned from the realization of the project and the possible improvements to the on-line auction portal.

5.1 Implemented functionality

Here is the table of the functionality implemented with respect to the system requirements previously explained:

Requirements	**Implemented**
User Side Stories	
Home page	YES
Registration	YES
Login	YES
Personal Page	YES
Browse	YES
Search	YES
Item Page	YES
Bid	YES
Post an Auction	YES
Help	YES
Change Language	NO
Chat	NO
Messages	YES
Administration Side Stories	
Administrator Page	YES
Manage Items	YES
Manage Users	YES
Manage Categories	YES
Accept/Refuse Auction	YES

As shown in the table, all requirements have been implemented except “Change Language” and “Chat”. We did not implement these two requirements for time constraints. However, the two requirements had a much more lower priority with respect to the others.

5.2 Problems

The main problem during the development phase was represented by the lack of documentation about JBoss: the JBoss Group (the creators of JBoss) has a program of documentation-for-sale and provides a free Getting Started guide. Unfortunately, this guide is rather disappointing. Even finding examples was difficult since they were not included in the main product download.
Another problem was represented by the architecture of EJB: a bean consists of more files (the home interface, the component interface, the bean implementation and the deployment descriptor), so it is very easy to lose track. However, the XDoclet tool helped me in solving this problem.

5.3 Knowledge Acquired

During the development of the project we have learned how to use JBoss, a very powerful open-source application server.
We also learned the advantages of using the EJB technology. In facts, EJB makes enterprise application development time relatively less since the developer can concentrate only on business logic. I have already worked with distributed system architectures based solely on JSP/Servlet/JDBC architecture, but EJB offer more advanced scalability features (object activation/passivation for beans, caching for entity beans etc.). It also offers transaction and security features without a need to program to any specific API. Moreover, data access is easier using Entity Beans than using pure JDBC.
With this project we also had the opportunity to see in practice the modelling techniques (use cases, user stories, UML class diagrams, EER diagrams, page flow diagrams) learned during the various courses of the University.

5.4 Future Works

The online auction portal works very well in all of its functionality. However, some future works can be done on the existing system:

- Add an SSL security system. Since a registered user can post new auctions, place bids, send messages etc., username and password are sensible data. So it could be useful to protect these data from being intercepted by a third party.
- Add a chat room to the portal. It would be nice for a user to enter in a chat room to talk with other users about auctions or any other topic. This chat can be realized using the Java Applet technology.
- Add a more attractive graphics to the web pages of the portal. The site is very easy to browse, also for new users, because the pages are simple and clear. However, the graphics of the site is also much simple, so it could be the case to improve it in order to attract more users.

- Add a credit card payment system. It would be nice for users to make payments using their own credit card to exchange money with the help of the website.

5.5 Thanks

For the realization of this text I used several resources (see the Bibliography). In particular, the Sun's website and the XDoclet's website were very useful during the development phase.
In addition, my special thanks go to my thesis' supervisor, Doctor Alessandro Artale, for the precious help given me in all phases of the project.

BIBLIOGRAPHY

[1] Agile Modelling Website www.agilemodeling.com
[2] Eclipse Website www.eclipse.org
[3] Elmasri Ramez, Shamkant Navathe, Fundamentals of Database Systems, 2004 Addison-Wesley
[4] Developer.com Website www.developer.com
[5] InternetWeek Website www.internetweek.com
[6] MyEclipse Website www.myeclipseide.com
[7] MySQL Website www.mysql.org
[8] North Carolina State University Website www.csc.ncsu.edu
[9] Oracle Website www.oracle.com
[10] PostgreSQL Website www.postgresql.org
[11] San Diego software studio Website www.softwarestudio.org
[12] Sun Website www.sun.com
[13] University of Texas Website www.utexas.edu
[14] Webopedia Website www.webopedia.com
[15] XDoclet Website www.xdoclet.org

APPENDIX

DEPLOYMENT DESCRIPTORS

The deployment descriptors include:

- The ejb-jar.xml file that describes the bean's business methods

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar >

   <description><![CDATA[No Description.]]></description>
   <display-name>Generated by XDoclet</display-name>

   <enterprise-beans>

      <!-- Session Beans -->
     <!--
       To add session beans that you have deployment descriptor info
for, add
       a file to your XDoclet merge directory called session-beans.xml
that contains
       the <session></session> markup for those beans.
     -->

      <!-- Entity Beans -->
      <entity >
         <description><![CDATA[Category EJB]]></description>
         <display-name>Category</display-name>

         <ejb-name>Category</ejb-name>

         <local-
home>it.liberedisce.interfaces.CategoryLocalHome</local-home>
         <local>it.liberedisce.interfaces.CategoryLocal</local>

         <ejb-class>it.liberedisce.ejb.Category</ejb-class>
         <persistence-type>Container</persistence-type>
         <prim-key-class>it.liberedisce.interfaces.CategoryPK</prim-
key-class>
         <reentrant>False</reentrant>
         <cmp-version>2.x</cmp-version>
         <abstract-schema-name>Category</abstract-schema-name>
         <cmp-field >
            <description><![CDATA[This is a cmp field.]]></description>
            <field-name>category_ID</field-name>
         </cmp-field>
         <cmp-field >
            <description><![CDATA[This is a cmp field.]]></description>
            <field-name>name</field-name>
         </cmp-field>

         <query>
```

```
            <query-method>
               <method-name>findAll</method-name>
               <method-params>
               </method-params>
            </query-method>
            <result-type-mapping>Local</result-type-mapping>
            <ejb-ql><![CDATA[SELECT DISTINCT OBJECT(category) FROM
Category category]]></ejb-ql>
         </query>
         <query>
            <query-method>
               <method-name>findByName</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
               </method-params>
            </query-method>
            <result-type-mapping>Local</result-type-mapping>
            <ejb-ql><![CDATA[SELECT DISTINCT OBJECT(category) FROM
Category category WHERE category.name = ?1]]></ejb-ql>
         </query>
        <!-- Write a file named ejb-finders-Category.xml if you want to
define extra finders. -->
      </entity>

     <!--
       To add entity beans that you have deployment descriptor info
for, add
       a file to your XDoclet merge directory called entity-beans.xml
that contains
       the <entity></entity> markup for those beans.
     -->

      <!-- Message Driven Beans -->
     <!--
       To add message driven beans that you have deployment descriptor
info for, add
       a file to your XDoclet merge directory called message-driven-
beans.xml that contains
       the <message-driven></message-driven> markup for those beans.
     -->

   </enterprise-beans>

   <!-- Relationships -->
   <relationships >
      <ejb-relation >
         <ejb-relation-name>CategoryHasCategory</ejb-relation-name>

         <ejb-relationship-role >
            <ejb-relationship-role-name>CategoryHasCategory</ejb-
relationship-role-name>
            <multiplicity>Many</multiplicity>
            <relationship-role-source >
               <ejb-name>Category</ejb-name>
            </relationship-role-source>
            <cmr-field >
               <cmr-field-name>supercategory</cmr-field-name>
            </cmr-field>
         </ejb-relationship-role>

         <ejb-relationship-role >
```

```
            <ejb-relationship-role-name>CategoryOfCategory</ejb-
relationship-role-name>
            <multiplicity>One</multiplicity>
            <relationship-role-source >
               <ejb-name>Category</ejb-name>
            </relationship-role-source>
         </ejb-relationship-role>

      </ejb-relation>
   </relationships>

   <!-- Assembly Descriptor -->
   <assembly-descriptor >
     <!--
       To add additional assembly descriptor info here, add a file to
your
       XDoclet merge directory called assembly-descriptor.xml that
contains
       the <assembly-descriptor></assembly-descriptor> markup.
     -->

   <!-- finder permissions -->

   <method-permission >
      <description><![CDATA[description not supported yet by
ejbdoclet]]></description>
      <unchecked/>
      <method >
         <ejb-name>Category</ejb-name>
         <method-name>findAll</method-name>
         <method-params>
         </method-params>
      </method>
   </method-permission>

   <method-permission >
      <description><![CDATA[description not supported yet by
ejbdoclet]]></description>
      <unchecked/>
      <method >
         <ejb-name>Category</ejb-name>
         <method-name>findByName</method-name>
         <method-params>
            <method-param>java.lang.String</method-param>
         </method-params>
      </method>
   </method-permission>

   <!-- transactions -->
   <container-transaction >
      <method >
         <ejb-name>Category</ejb-name>
         <method-intf>Local</method-intf>
         <method-name>getSupercategory</method-name>
         <method-params>
         </method-params>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>

   <!-- finder transactions -->
```

```
   </assembly-descriptor>

</ejb-jar>
```

- The jboss.xml file, that describes the interfaces:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss_3_2.dtd">

<jboss>

   <enterprise-beans>

     <!--
       To add beans that you have deployment descriptor info for, add
       a file to your XDoclet merge directory called jboss-beans.xml
that contains
       the <session></session>, <entity></entity> and <message-
driven></message-driven>
       markup for those beans.
     -->

      <entity>
         <ejb-name>Category</ejb-name>
         <local-jndi-name>ejb/CategoryLocalHome</local-jndi-name>

        <method-attributes>
            <method>
              <method-name>getCategory_ID</method-name>
              <read-only>true</read-only>
            </method>
            <method>
              <method-name>getName</method-name>
              <read-only>true</read-only>
            </method>
        </method-attributes>

      </entity>

   </enterprise-beans>

   <resource-managers>
   </resource-managers>

</jboss>
```

- The jbosscmp-jdbc.xml file, that describes the bridge between JBoss and the database

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jbosscmp-jdbc PUBLIC "-//JBoss//DTD JBOSSCMP-JDBC 3.2//EN"
"http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_3_2.dtd">

<jbosscmp-jdbc>
   <defaults>
     <datasource>PostgreSQLDS</datasource>
     <datasource-mapping>PostgreSQL</datasource-mapping>
   </defaults>
```

```
   <enterprise-beans>

     <!--
       To add beans that you have deployment descriptor info for, add
       a file to your XDoclet merge directory called jbosscmp-jdbc-
beans.xml
       that contains the <entity></entity> markup for those beans.
     -->

      <entity>
         <ejb-name>Category</ejb-name>
         <table-name>TBL_CATEGORY</table-name>

         <cmp-field>
            <field-name>category_ID</field-name>
            <column-name>CATEGORY_ID</column-name>

            <jdbc-type>INTEGER</jdbc-type>
            <sql-type>INTEGER</sql-type>

        </cmp-field>
         <cmp-field>
            <field-name>name</field-name>
            <column-name>NAME</column-name>

            <jdbc-type>VARCHAR</jdbc-type>
            <sql-type>VARCHAR(250) BINARY</sql-type>

        </cmp-field>

<!-- jboss 3.2 features -->
<!-- optimistic locking does not express the exclusions needed -->
      </entity>

   </enterprise-beans>

  <relationships>
    <ejb-relation>
      <ejb-relation-name>CategoryHasCategory</ejb-relation-name>

      <ejb-relationship-role>
          <ejb-relationship-role-name>CategoryHasCategory</ejb-
relationship-role-name>
              <key-fields/>

      </ejb-relationship-role>
      <ejb-relationship-role>
          <ejb-relationship-role-name>CategoryOfCategory</ejb-
relationship-role-name>
          <key-fields>
             <key-field>
               <field-name>category_ID</field-name>
               <column-name>SUPERCATEGORY_ID</column-name>
             </key-field>
          </key-fields>

      </ejb-relationship-role>
    </ejb-relation>
  </relationships>

</jbosscmp-jdbc>
```