Division of Informatics, University of Edinburgh

Design and Implementation of an Online Auction

by

Theodoros Theodoropoulos

Informatics Research Report EDI-INF-RR-0029

Division of Informatics **September 2000**
http://www.informatics.ed.ac.uk/

Design and Implementation of an Online Auction

Author:Theodoropoulos Theodoros
Supervisor:Stephen Gilmore

University of Edinburgh
Department of Computer Science
September 15, 2000

Abstract

This dissertation reports on the design and implementation of a distributed application that hosts online auctions. The system is implemented using Java and CORBA. Special care has been taken to make it secure, flexible, easy to use and deploy under different platforms and in addition, more interactive than a typical application.
Several experiments were performed and several configurations were tested. It was found that such an application gives reasonable performance and is an attractive alternative to the more conventional web-based approach that is popular nowadays.

Declaration

I declare that this thesis was composed by myself and the work contained therein is my own, except where explicitly stated otherwise in the text.

Theodoropoulos Theodoros.

Contents

Chapter 1

Purpose of project

The purpose of my project was to write an example application of an online auction system, that would be built to adhere to the Client-Server principle, yet be preferably platform independent and easy to use.

Although some auction sites existed at the time the title of the project was decided, all of them made use of the world-wide-web, with all the advantages and disadvantages that this approach hides. My aim was to develop a separate, stand-alone client (that would not require a web browser), which could still be used under a variety of operating systems.

And that was the innovation: I believed that my approach would contribute in making the whole bidding experience more interactive and hence more enjoyable and exciting for the user.
In contrast to the expensive and 'heavyweight' legacy systems that the traditional online sites used, my proposal would be more flexible, more configurable and ideally based on completely free software components.

Because of the nature of the application, Java was chosen as the implementation language for the program. Several other packages had to be used, but care has been taken to avoid proprietary code (where possible) and use open source programs instead. Additionally, I tried to have most of the third party software I've used also in Java, so that the whole application could be deployed under diverse platforms without any re-compilation.

Chapter 2

A few things about auctions

2.1 Classic Auctions

Before we mention details of the program and the strategies that were used, it would be useful to mention some basic things about auctions.

An auction can be defined as: *the buying and the selling of goods through a system of open public bidding where individuals give successively increasing bids until only one bidder remains.* The individual with the highest bid earns the right to purchase that specific item for that price.

Although we have the impression that the idea of buying through auctions is relatively new Shubik [23] provides an attractively written historical sketch of auctions going back to the Babylonian and Roman empires[1].

In **classic auctions**, items for sale are presented for viewing and evaluation by the public before the auction begins. When (for any reason) bidders [2] have no access to the item in question, the seller of the item usually provides the bidders with a brief description of the item's features and characteristics (and may also include other information that he/she thinks they will tempt the bidders to bid higher).
In any case before an individual can bid, he/she must present credentials to verify their identity as well as that they can follow through with the transaction, should they place the highest bid.
In classic auctions, an Auctioneer [3] is responsible for the fair and smooth flow of

[1] Most famously, the whole Roman empire was sold by ascending auction (see later) in A.D. 193 by the Praetorian Guards; the winner (and therefore the next Emperor), was Didius Julianus who reigned for just over two months before being overthrown and executed by Septimius Severus...

[2] Bidders are member of the public who compete to purchase the item being sold

[3] The Auctioneer is the individual who conducts the auction and most of the time is *not* the

the auction, by taking bids from prospective buyers until the highest bid has been reached. The ending of an auction is determined either by the end of its 'lifetime' (set by the Auctioneer in advance) or by the lack of new bids. If the demand is extremely high for an item, the Auctioneer can extend the auction ending time to satisfy all the prospective buyers. Then the highest bidder purchases the item from the seller (who can also be the auctioneer) for the winning bid price.

Not surprisingly, a huge volume of economic transactions is conducted through auctions. Usually items like work of art, antiques, computers [4] (and peripherals), cars, electronic appliances or even stocks are sold by individuals at auctions. Governments also use auctions to sell treasury bills, foreign exchange, and other assets such as firms to be privatised. Although these latter auctions appeal to organisations and companies rather than individuals, the principle behind them is the same: the bidder is seeking to buy at the lowest possible price. (However this is not always possible)

There are several auction formats, although five of them are widely used and have been analysed by experts.

- The ascending-bid auction (also called open, oral, or English auction)
  This is the most common type of auction, in which the price is successively raised until only one bidder is left (who also wins the item). There are two slightly different 'flavours' of the English auction: in the first, the auctioneer announces the new bidding price (set by the seller) after each bid; in the second flavour, the bidders themselves set the new bidding price by bidding as high as they want (although sometimes a minimum price increment exists).

- The descending-bid auction (also called Dutch [5] auction by economists)
  Although the title confuses the reader in the beginning, the descending auction has a similar effect even though it works exactly in the opposite way: the auctioneer starts at a very high price and gradually lowers it. The first bidder who accepts the current price, wins the auction, and gets the item at that price. This type of auction has less interaction between bidders, nonetheless seems to offer same amount of excitement...

- The simultaneous-bidding auction (or Japanese)
  For this auction all prospective buyers simultaneously use hand signs to

seller of the item, but rather a trustworthy agent set by the owner to sell an item

[4] The variety of items one can find being auctioned is amazing: Recently, a Cray Research Y-MP C90 supercomputer was won by an individual that wanted to put it on display!. The item was auctioned in eBay and was sold for around $45.000. For full details read the online article [29]

[5] named Dutch, because it was initially used in the sale of flowers in the Netherlands

show their bids to the auctioneer as soon as the auction begins, who then picks the highest bid among them and announces the winner.

- The first-price sealed-bid auction (known also as Haphazard Bidding)
  Under this scheme, each bidder secretly submits a single bid to the auctioneer, without knowing about others' bids. The item is again sold to the person with the highest bid.

- The second-price sealed-bid auction
  [6] This type of auction is similar to the first-price sealed-bid auction in the sense that each bidder makes a single bid and none of the bidders knows the current highest bid, but differs in that the winner pays the second highest amount (and *not* the highest, posted by him/her).

For these basic types of auctions, there are some variations:

- Time-interval auction
  This type is a variation to the English type auction, where the bidding stops after a specified time period has elapsed.

- Reserve Price
  This is another variation where the seller specifies the existence of a 'minimum required price' for the item taking part in the auction. Prospective buyers however are unaware of the exact price, and are notified only when they exceed that limit. By the end of the auction, the seller has the right to pass over any bids that are under the reserve price and refuse to sell the item to the highest bidder if the 'reserve price' has not been reached.

- Starting Price
  This variation applies to most auction formats, and simply denotes the existence (or lack) of a starting price for the bidding. Although one would think that a starting price should always be present even for inexpensive items, so that the buyer has some guarantee that he/she will definitely get a respectable amount of money (analogous to the value of the item), practice has shown that low (or null) Starting Price in conjunction with NO Reserve Price draws the interest of more people who are tempted to bid more for a specific item and hence raise the profit for the seller...

- Express Auctions
  Express auctions are short lived auctions that usually last between half to one hour. They have been introduced to offer bidders an exciting and quick

[6] An alternative name for this auction is 'Vickrey auction' taken from the name of the author of the first important paper on auctions

auction experience. Under this scheme, Bidders need not wait several days to find out whether they have won the item or not.

- First bidder discount
- Take-it-Price
- Featured auctions

Some disadvantages of the old-fashioned auction format are:

- Classic type auctions used to take place at specific times and places.
- A significant amount of preparation was needed before each auction, which in conjunction with the costly setup of the auction itself, made this method efficient only for the selling of relatively expensive items.
- For the same reasons, taking part in an auction either as a seller or buyer was a privilege only for the few...

2.2 Online Auctions

Having mentioned how classic auctions work, we can now see what advantages the online version offers to both the seller and the bidders.

First of all, instead of a physical place where all the items for the auction are kept, in the online auction type, a central computer holds descriptions and pictures of the participating items which are (in general) presented to the public via the Internet. Most of the time there is an Internet site which serves this purpose, and all the interested parties visit that site, make their choices and bid online.

Online auctions have several advantages over the old-fashioned format:

- ◇ Online Auctions create more efficient markets by bringing together a wider variety of buyer and sellers.
- ◇ They beat the geographic constraints, by making auctions available to *any* person with access to the Internet.
- ◇ They can be more descriptive than classic auctions and classifieds, by using Images and Hyper-Links that allow the bidder to make comparisons and see reviews regarding the item they are interested in, on the fly.

- ◇ Also, because placing an auction is (most of the time) free, and bidding is simple this type of economic transaction has become very popular. This resulted in hundreds of auctions beginning every minute, so everyone can find something of interest.

- ◇ And of course, online auctions allow for an interactive buying experience through the bidding process. It also contributes to the buyer's and seller's sense of online community. It is so interesting, that it can be addictive in some cases...

- ◇ Although there were security problems when online auctions just became reality, they are now claimed to be more safe, with the introduction of feedback forms (where the buyer can express his/her opinion on the transaction with the seller). Also the identity of the seller and his/her credibility are assured by Credit-Card authentication process from the Internet web-site that hosts the auction. To help enhance the trust between seller and buyer special firms have been formed that guarantee the successful end of the transaction to both the seller and buyer, after the auction has ended.

It is not surprising that several well known online firms have added online auctions to the range of services they offer to the public. Some of the most famous are: Yahoo! Auctions, Lycos Auctions and Amazon. On the other hand, other companies make profit solely by hosting online auctions. Such companies are eBay, QXL, uBid with mirrors sites all over the world.
Naturally, the online auction trend could not but influence traditional auction houses (such as Sotheby's and Cristie's) to make use of the new technology to sell their items online.

Of course this is only a small sample of well known online auction sites; it is difficult to estimate the real number of them as they grow rapidly day by day. On a survey on that matter, Business Week estimated an average of 125 actual online auctions in 1997. Today, a simple search on 'online auctions' on a search engine, gives thousands of hits. Although not all of them will not be actual sites that sell items online by auction, no one can argue that the number has grown hugely in the last few years.

Chapter 3

Distributed Computing, CORBA and alternatives

3.1 The Client-Server model

The application that I developed is based on the Client-Server model, which briefly can be described as *the application development methodology that allows developers to distribute processing among networked computers, and by that, make more efficient use of computer resources.*
In standard Client-Server computing, an application will have at least two parts: the server component and the client component, each running on a different computer.
The role of the Server is to execute requests sent to it by the client and to return the results to it.
The clients on the other hand, are responsible for handling the interactions with the end-user, and by that collect the necessary information to post the correct requests to the Server. A Graphical User Interface is almost always necessary for the client part. It also has to be able to handle any results returned by the Server.

Normally the Server will be one, although the Clients will be many. Because of that, it is necessary for the programmer to divide the computational overhead accordingly to both the Server and Client. Although several years ago the idea of thin-clients (dumb terminals) connecting to a super-Server was popular, this approach is no longer used, mainly because personal computers are getting more and more powerful. Of course the other extent is also not useful for general-purpose applications, so the best solution lies somewhere in between with the server handling *most* of the computations. Developing Client-Server applications is generally more difficult, however this approach also has several advantages:

- As we have seen before, the programmer has the flexibility of being able to assign computation to the most appropriate computer system.

- Although not common practice, server programs can be divided among several server systems so that better load balancing can be achieved.

- Because the application logic is divided to more than one program, the server application can be changed (within certain limits that will be explained later) without having to disturb the users of the client programs, nor having to redistribute an executable.

The standard Client-Server model poses no restrictions on the type of the operating system of the client or the server part, hence the application can, in general, be developed and executed in an heterogeneous environment. However this approach has some unpleasant features:

- The client program has to either use the same network protocol with the server, or to provide for protocol conversion in case it does not.

- Applications running under different operating systems have to also take care of data format conversions (little/big-endian and floating point number conversion). But data marshalling is not a trivial thing...

- Even if client and server applications run under different operating systems, they are tied together, and any attempt at porting the client to another operating system or a computer that uses another network interface, means changing the code (and probably the data marshalling and networking routines) in the server side as well.

3.2 Adding tiers

Because of the above mentioned disadvantages, technology quickly moved from the two-tiered Client-Server approach to the more flexible three-tiered (and multi-tiered) in which extra logic intervenes between the Client and the Server.
Before, most of the business logic resided on the client, and the server was usually a database that returned the requested rows to all the clients.
The first step towards the multi-tiered approach was to split the server to a database and a business logic part. Now only the server talks to the database, gets and processes the results, and then returns the updated values to the client which now has mostly a graphical front-end role. This is known as the application server architecture.

- Apart from better *separation of duties*, this architecture provides *higher security* (and can be fine-tuned at the method level) as sensitive data from the database is not communicated over long distances in a raw format, but is 'encoded' into methods.
- Additionally we get a higher degree of *encapsulation* because the client only has access to methods exposed by the server.
- The *performance* we get from this approach is better, since only filtered-translated data travel through the network (which is a major bottleneck in distributed systems).
- We also get better *application reuse*. Server objects can be used multiple times from the client.
- *Administration* of the distributed application also becomes easier, as most of the business logic resides on the server and any change made to it becomes instantly available to all the clients. Applications can be centrally managed on the server...
- Apart from these features, we should not forget that the main purpose of a distributed application is to be able to provide service to multiple clients at the same time. As the number of clients goes up, we get a problem of scaling; the three-tier approach however, *scales better* because we have the ability to load-balance across multiple systems.

3.3 The middleware, alternatives

Both database and application systems can be enhanced by adding additional tiers to the architecture. So-called four-tier systems have an intermediate component between the client and the business logic part of the server, which in general is called '**middleware**'. The essential role of middleware is to *manage the complexity and heterogeneity of distributed infrastructures and thereby provide a simpler/more efficient programming environment for the development of distributed applications.*

Middleware abstracts the communication interface to the level of method invocations (and by that elevates the problem of working with heterogenous environments that we have seen above); Instead of having to engage in application-level protocols to encode and decode messages, middleware works with methods and gives the illusion to both the client and server sides of the application that all these methods are 'local'. The arguments of the call are packaged by the middleware and shipped off to the call's remote target. All these however are transparent to the user, resulting in perfect location transparency

Three major distributed object platforms have recently become popular and although quite different from each other, all base their functionality and features on earlier technologies like OpenGroup's DCE, Sun's RPC interface and Novell's NetWare (that were not object based). These new contestants are: OMG's CORBA, Microsoft's DCOM and Sun's JAVA RMI.

Microsoft's DCOM:
This is Microsoft's proprietary suggestion for the distributed-object platform, which extends the traditional Component Object Model (COM) that Microsoft uses for application inter-operation within the Windows environment. Although several other companies are trying to port DCOM to other platforms as well, it was —and will be— designed to work better under the Windows OS suite.

Sun's JAVA RMI:
Following the tag line 'the network is the computer' in Sun's campaign, one should not be surprised to find support for distributed operation included in the standard distribution of the Java language. With the use of 'serialisation' (a very important feature of the language), Java manages to convert most of its objects to a transferable form; using a similar method objects are reassembled at the other end, still maintaining their original structure, contents and relations to other objects. This is probably the easiest to use (and most flexible) implementation of the distributed object model, provided that both the client and the server are written in Java. Even more, with Java having been ported to most of the popular platforms, RMI will probably be a natural selection for the Java developers.

OMG's CORBA:
With similar functionality to DCOM, but with completely different design philosophy, CORBA is an open specification (rather than a proprietary implementation) that is supported by the 700+ companies which comprise the OMG group. This approach gives the power, the maturity and the wide acceptance that is necessary for its success.

Other technologies that incorporate the client-server approach (like Enterprise JavaBeans, Aglets...) emerge every day and the selection of the one that is best suited for a specific distributed application, becomes more and more difficult.
Most of the time, certain restrictions are placed on the developer by the available hardware and software infrastructure of a company, that narrow the possibilities; However, in my case no such restriction was made (apart from the fact that Java had to be used as the implementation language), so I had to put my own criteria for the application and from there choose the best approach.

My primary aim was to develop a distributed application that could run under different Operating Systems, while still performing sufficiently well when more clients were added. Java was the ideal choice for that matter, by providing a multi-threaded portable operating system for running objects, with automatic garbage collection and advanced error management, while still being relatively easy to program, debug and deploy the application.

3.4 CORBA

CORBA was chosen for the middleware, mainly because of its maturity over RMI and DCOM, the flexibility that gives to the programmer and its robustness under heavy load. Indeed CORBA has been around for many years, has been tested with various environments and network configurations and shown to perform well. Although bindings for other languages (such as C++, Cobol, ADA, etc) exist, CORBA complements the Java language very well; Java starts where CORBA leaves off: CORBA deals with network transparency, while Java deals with implementation transparency.

3.4.1 The ORB

CORBA defines the '**ORB**' as the middleware that establishes the client-server relationships. The ORB (known also as the *software bus*, or the *object bus*) lets distributed objects inter-operate across address spaces, languages, operating systems and networks. It also provides the mechanisms that let objects exchange meta-data and discover each other. The ORB intercepts the call made by a client and is responsible for finding the object that can implement the request.
A CORBA client knows only how to ask for a task to be executed (or how to call a remote method) and a CORBA server knows only how to accomplish a task that a client has requested. Both of them are unaware of each-other's location, as they communicate via the ORB.
A block diagram that shows how the ORB connects the client and server tiers via the network is shown in Figure 3.1.

3.4.2 The IDL

The operations that a client can request and to which a server has to respond, are defined by the interface that both client and server have to support to inter-operate smoothly. In CORBA, interfaces are defined in a language-independent format called IDL (Interface Definition Language).

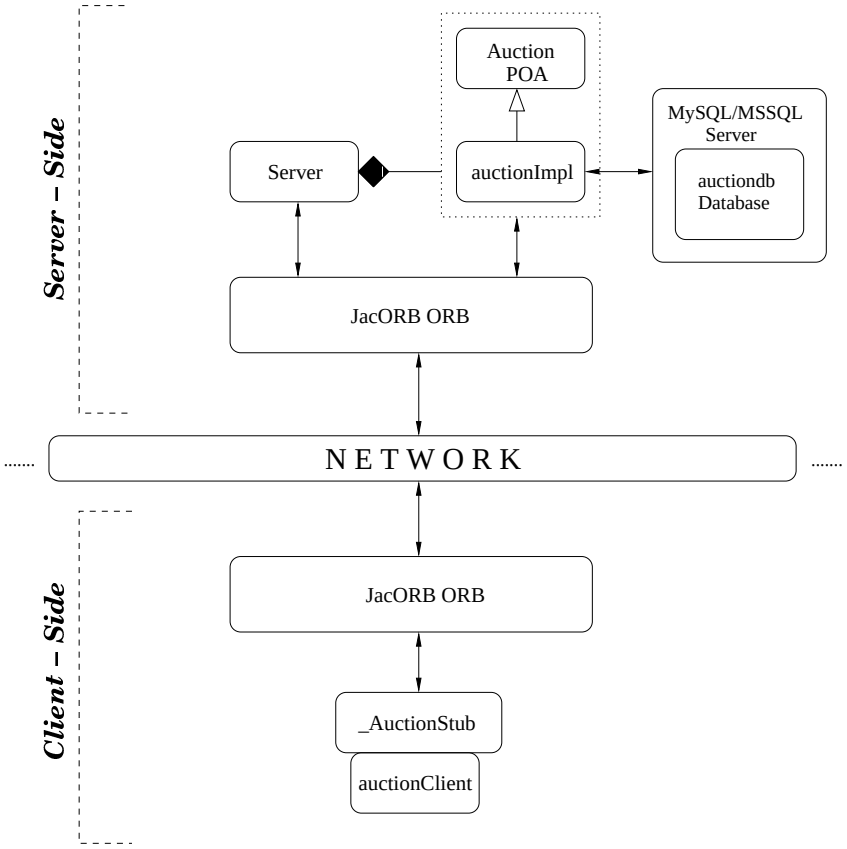

Figure 3.1: A simple block diagram that shows how CORBA communicates with its peers. The blocks represent the actual logical entities that were used in the design phase of the auction program.

IDL is developed and maintained by OMG (the creator of CORBA) and is purely a declarative language that provides no implementation details. IDL for example, defines new data structures, new exceptions and codes all the remote methods that are exposed by the Server (along with their return types and any exception that they might raise), but cannot be used to write ie. new algorithms.
IDL's grammar is basically a subset of C++, with some additional keywords to support distributed concepts.
The use of a definition language, such as IDL is the milestone that enables CORBA to be independent of any particular programming language; separate utilities are included in every ORB implementation that automatically create the necessary language-specific interface files from the IDL. This functionality is available for all the programming languages that CORBA has bindings for. Currently such bindings are available for C ,C++ ,ADA ,Smalltalk ,COBOL ,Lisp and of course Java.

The actual IDL file that was used in the auction program can be found in Appendix A.

3.5 Choosing the ORB

As I've mentioned before, CORBA is just a specification, and there are several implementations available that comply (some more than others) to the standard. Choosing which one to use for the application was also a problem, although this time, some of the competitors were ruled out by the facts....

SUN was one of the founders of the OMG group (the creators of CORBA) so one should not be surprised to find CORBA support included in the standard Java language distribution. This is a major advantage, since Sun's implementation is freely available to all the users of Java, and has been tested thoroughly (judging by the popularity of the language). Unfortunately, the Java ORB is not a complete implementation of the specification, and can only be used for doing relatively simple things (although Sun is currently working on enhancing the CORBA support in Java). It is also found to be not as scalable as other ORBs.

On the other hand, there are other ORBs that are fully compliant to the latest CORBA standards, and show excellent scaling performance even under heavy load. Such ORBs are Borland/Visigenic's VisiBroker for Java (currently version 4) and Iona's OrbixWeb v3.0 (although other companies have equally powerful products). Documentation and support for these software packages is also excellent making them the first choice for professional developers in big companies; However —as you might have imagined by now— these products cost several thousand pounds, which makes them unaffordable by the humble student.

So another solution had to be found... Doing a brief search on open-source/Freeware ORBs, I found that there is a plethora of fairly good products. A full list of such ORBs can be obtained at `http://www.omg.org/technology/corba/corbadownloads.htm`. Following recommendations of professors related to the subject, as well as online reviews, I decided to download and test JacORB v1.1 and Distributed Object Group's JavaORB v2.2.5 (not to be confused with Sun's Java ORB).
JavaORB is a French implementation of the CORBA standards which seemed to have better documentation, and be more up to date, but contained some serious errors in the Unix scripts that were used to execute parts of the program. Additionally some of the errors returned by the ORB were in French which in conjunction with the ones in the —relatively easy— script files, gave me a negative impression about the robustness of the product.

Therefore, I finally decided to use JacORB for the application. JacORB v1.1 is a German implementation of the CORBA v2.0 standards developed at the Software Engineering and System Software Group of the Computer Science Dept in Freie Universitt in Berlin. It offers support for several CORBA services although -as i realised later on- some features were not yet fully implemented. Unfortunately the newest version was announced to be released during the time i would be finishing the writing of this thesis.

Full details of that ORB plus documentation and binaries of the latest release can be downloaded from JacORB's homepage at `http://www.inf.fu-berlin.de/\~brose/jacorb/`

Chapter 4

Main part - program details

4.1 Program Requirements

4.1.1 Hardware

The program itself has no special requirements for hardware, although the computer running the server should have enough CPU power and memory. During the development phase a Celeron 400MHz Linux box was used with 128Mbytes of RAM, interchangeably with a Windows 98 workstation running on an AMD K6-II-400 with 128Mbytes of RAM. They seemed to be adequate for running the database, the application server and one or two clients at the same time.

But in real life, one should expect to have the database tier running on a separate box equipped with (preferably) two or more processors, as much cache memory as possible [1], and plenty of RAM (although 128Mb-256Mb should be enough for most uses, even under Windows).

The server program itself uses approximately only 10Mbytes, so it could reside in the same computer as the database server; By this, we even minimise communication errors/delays (that could occur in case the server tier and the database tier were away from each other) and enhance security, as some of the content that travels from the server to the database is in plaintext format. (The database itself has no 'brains', it is there just to keep data)

Memory usage for the client program is higher (approx. 24Mbytes) due mainly to the use of Swing libraries for the GUI. (Java programs are known to be memory hungry)

[1] Cache memory has been proved to dramatically increase performance in machines dedicated to databases, as most times, executed instructions are the same; hence they can be fetched from cache, provided enough exists.

So, the only real hardware requirement is a network connection for both the client and server application. For the client side, a modem connection should be sufficient, as the amount of information being exchanged is relatively small (although it can grow, depending on user's selections).

4.1.2 Software

For the software requirements, things are more complicated...

The computer that runs the **database server** needs to have at least:

◦ MySQL v3.22.32, OR
◦ Microsoft SQL Server 7.0

The computer that hosts the **server program** has to have at least:
· Sun's JDK1.2.2
· JacORB1.1
· Sun's JavaMail1.1.3
· Sun's JCE1.2.1beta
· ABA security provider (available from www.openjce.org)

Fortunately, all these are available to download for free. Configuring these components to work together should not be a difficult task (although configuration and tuning of the ORB can be quite tricky for the inexperienced).

For **the client**, the only necessary component is the distributable compressed .jar file, which however contains some classes from the JacORB package that are required for it to connect to the server. Included in the jar file are also some selected classes related to security, as both the client and server need to encrypt and exchange information. Of course JDK1.2.2 (or the equivalent JRE) is also required to execute the jar file; it is now included in most of the Linux distributions (and Solaris) as standard, but has to be downloaded for Windows users.

4.2 Program description

When planning for the program, I had to choose between building a Java applet or an application (Java was definitely going to be the implementation language). Although one would consider an applet being a better solution for this type of

program, I found several disadvantages in this approach, and decided to finally develop it as an application.
The main reason for the decision was the size of the resulting code. Although in the beginning I couldn't know the exact size of the program, I estimated that it would be no smaller than 300Kbytes; that means a download time of at least 42 seconds (under optimum conditions) for a 56Kbps Internet connection [2].
I decided that this was unacceptable, as the user would have to undergo the same procedure of downloading every time they would have to use the program.

Additionally, from my experience with Java applets (which follows the public opinion on that matter) the standard Java support that is supplied with web browsers is either incomplete, outdated, or behaves differently depending on the platform and/or product. That was the last thing i wanted for the application: users complaining of not being able to properly execute and use the program.

So, a Java application was chosen[3]. I think I also have to mention that the aim of the project was not to develop a commercial product, that would compete with the packages that are being used by the major auction sites, but rather to present a functional model application that would use Java as the implementation language.

The program is built to support the ascending type auctions (or English type) which is the most common format. It also supports the 'reserve price' and 'take-it-price' options, while the seller (being able to choose the duration of the auction) can simulate the express-auction variation. If necessary, the starting price for bids can also be chosen by the seller, (although it can optionally be left zero). But we will see these in detail in the next section.

4.3 Program Features

4.3.1 Startup Screen

If the Client is connected to the Internet and the Server is running, the user can invoke the following command 'java -jar auctionClient' and hopefully after a

[2] At the end, the jar file was more than 1MB !! (including images, ORB and security related class files)

[3] For those who still think that a Java applet would have been (for some reason) a better solution, there is a 'trick' where a user can click on a button (found in a web page) and get the application running in a new window; I have not yet tried it, but it should hopefully work

while the following screen will appear:

Figure 4.1: Program Startup Screen

The screen displays the user interface, which is divided in three parts:

◇ The first (upper) part is a tabbed panel which contains the 'Login', 'Search', 'My Watchlist' and 'Auction Composer' tabs. Initially the 'Login' tab is selected that contains the UserName/Password fields, the Login and the Register Button.
◇ The second part will display detailed information about the item and the auction it is currently involved in. (More on that in a following section)
◇ The third part contains the Status Bar (which notifies the user of what is currently going on), the UserName (when a user logs in with a UserName/Password), or Anonymous (for anonymous login), the connection status and finally the time as retrieved from the server.

New users that just want to browse the available auctions, can log in anonymously by pressing the login button and leave both UserName and Password fields empty. In this way however, several options become disabled (for instance they will not be able to bid for an auction, nor add an auction to their Watchlist).

4.3.2 Registration

If however users require the above services, they can choose to register for a normal account. This is done by pressing the 'HERE' button; a separate window appears (shown in Figure 4.2), where the user is supposed to enter their details.

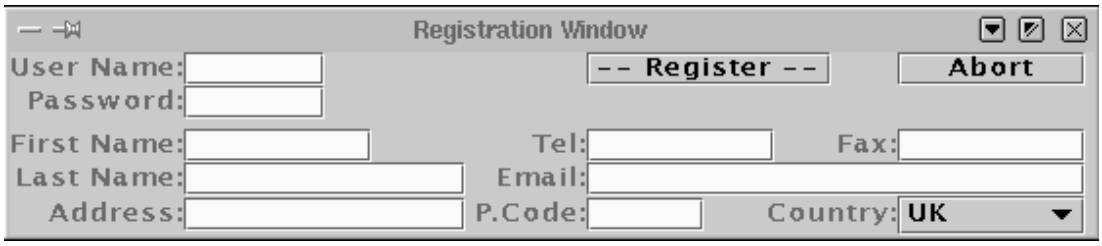

Figure 4.2: Register Screen

Although all fields (UserName/Password, First Name, Last Name, Address, Post Code, Country, Tel/Fax and email address) should be completed [4] by the prospective user, only the UserName/Password and Email field are *required* for a successful registration.
The email is checked for validity by counting the supplied '@' characters; the system accepts the address only if it contains only one '@' character. For the UserName and Password the first 10 characters are taken into account (the user could provide more, but they will be ignored by the system —more specifically by the database server).
In addition to these information, during the registration phase, I should normally be asking the user to provide a valid credit-card number to the system, just for security (and identification) reasons.
This is also done by many online auction systems; this extra piece of information is said to be used only in extreme cases of repeated fraud reported for a seller (where the real identity of the owner of the credit-card is investigated and legal action is taken against them). Unfortunately, I would have no means to verify the validity of the supplied series of numbers (For obvious reasons, the algorithm for credit-card number verification is not publicly available), so I decided not to.

When the user feels happy with the provided information, he/she can press the 'Register' button.

[4] These details will be used when an auction ends, to notify the seller/winner so they should be as accurate as possible. However, even if just the email is supplied (maybe because a user does not feel comfortable with supplying their details to the system) the transaction will complete successfully, and the seller/winner can later exchange their real details

Of course all the information contained in the registration form are considered confidential, and have to be encrypted before they can be communicated to the Server. (this normally takes a few seconds to complete, because a fairly secure method was chosen for encryption). The encryption process will be discussed in detail in the next Chapter.

On the other side, the Server decrypts the supplied information, sends them to the database, and in case another user with the same UserName exists, throws an exception which is transferred to the client (UserExists exception).
The client then brings up an information window, inviting the user to pick another UserName.

When everything is fine, the information gets permanently stored in the database in the UserTBL table.

That concludes the registration phase. The user now can login with their new UserName.

4.3.3 Browsing Auctions

Upon a successful login, the status bar displays the UserName, and the second tab gets automatically selected. This is the Auction Browser and is divided in two parts: the left window which is a tree containing the names of the available auction categories, and the right panel which displays a brief description of the auctions that are available in a certain category. A tree was chosen for the display of the auction categories because the user might want to move up or down the hierarchie and still see the expanded nodes.

A screenshot of the browser tab is displayed below:

Figure 4.3: Auction Browser

The category tree in the left hand side is created dynamically upon the creation

of the client program and contains all the available auction Categories.
It is constructed from the CategoryTBL of the auction database. The approach taken to create and populate the category tree is the following: An entry in the database contains triples of the form category name id, category name, number of the parent category that this category is dependent on. The rows are ordered by the category name id, (and are retrieved in this order) so that we first create the basic categories (those with the lower ids) and then fill the tree with the leaves. There can be more than one leaves in a basic category, although for the example application, I only used a depth of two. The names of the categories where taken from an actual online auction site (uk.auctions.yahoo.com) and there are a total of 64 nodes. These are easily expanded by adding new entries to the database, following the above rule for the allowed triple.
The advantage of this (rather more complicated) approach over the alternative of having the entries statically contained on the client side, is that the system administrator is able to easily add entries to the auction categories; the user will have instant access to the updated entries by re-executing the client program.

On the right hand side of the 'browser tab' a table is created and displayed each time the user clicks a child node on the category tree. That table has 5 fields that give a brief description of the auctions that are available under that category. Each auction is briefly described in a separate row in the table.

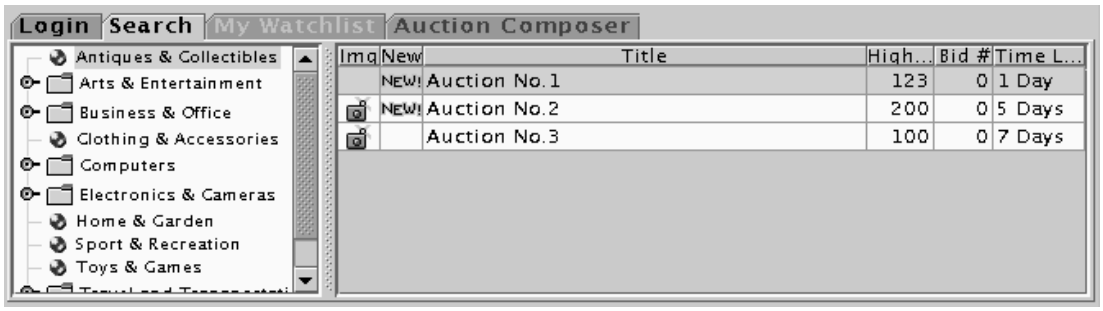

Figure 4.4: Brief Description of available Items

The first column contains a small icon when the detailed description of the item in question has an image, or is left blank in the opposite case.
The second column contains another a 'new' icon, when the auction has been placed during the past 12 hours (and hence is considered as new by the system). This is only meant as information to the regular visitors of the auction system, as they will be probably aware of all the older items.
Next to these icons i decided to put the title of the auction, which is probably the most essential piece of information for the potential buyer. Most of the time this entry contains just the name of the product being sold, as the space is limited.

Then, the highest bid (the current price being offered) is displaced as well as the number of bids. This last information is also helpful, as a high number of bids usually indicates a really good bargain.
The last column in the table contains the remaining time until the end of the auction. If the remaining time is greater than 24 hours, the number of days is displayed; if however less than 24 hours remain, the time format changes so that hours and minutes are displayed.

The combination of the above information is (according to my opinion) enough to help the person looking for an item, decide whether or not that auction deserves more attention. A unique Item ID is held (but not yet shown) for each item being displayed.

4.3.4 Getting Details for an auction

If the user left clicks anywhere on a row, the client sends a request to the server for additional information regarding the item with that particular id. This is where the middle part of the user interface comes to use.

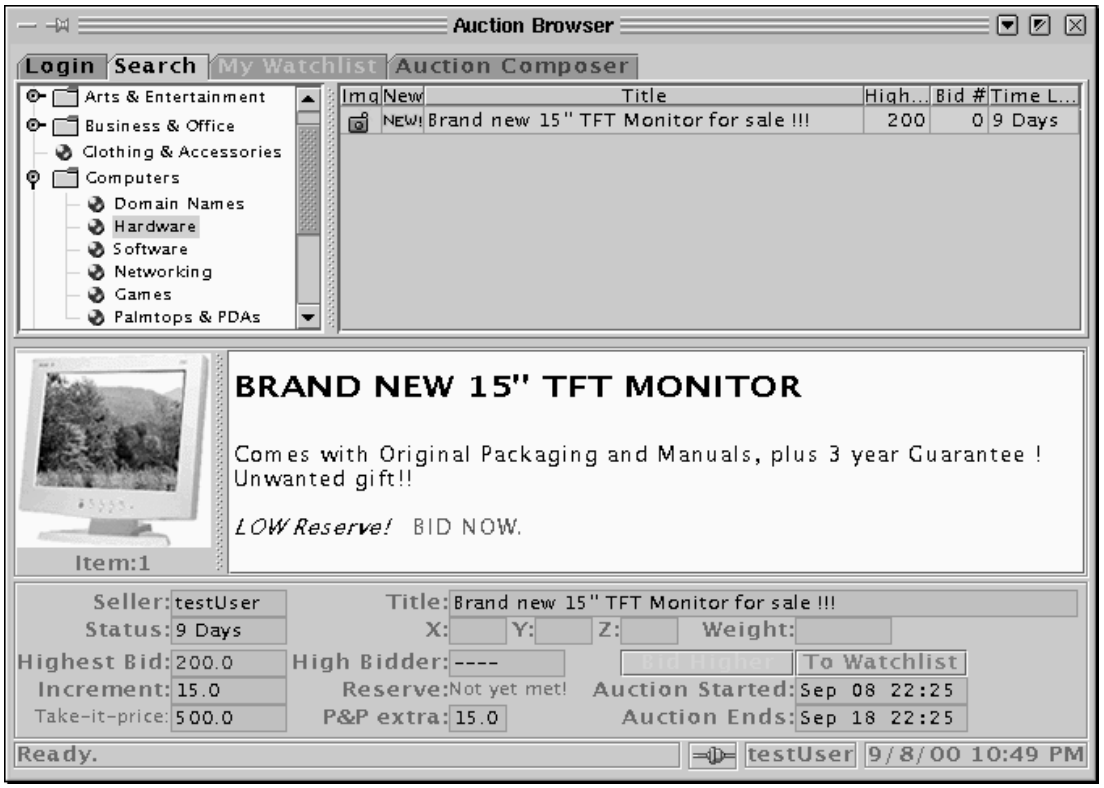

Figure 4.5: Details regarding the auction

Extra information is now displayed, like:

- The full description that the seller gave for that item. This is displayed in a JEditorPane which can optionally show html encoded information (although no links to web pages are allowed)

- Optionally an image of the item (if the seller provided one) along with the item serial number (as held in the database).

- The name of the seller

- The time remaining until the auction ends

- The highest bid received until then

- The desired increment of the bidding amount (if one was defined by the seller)

- The take-it-price (if one was defined by the seller). As we have seen, this option exists so that users who think they can pay the price asked in this field, can instantly get the item (instead of having to wait until the end of the auction, to see if they are the winners).

- The title of the auction is again mentioned

- Dimension (X,Y,Z) and Weight information are displayed if they were given by the seller

- The name of the current highest bidder is also displayed. If no person has bid for the item "----" is displayed in that field.

- A special message (Reserve Met, Reserve not yet met, No Reserve!) is also displayed to support the Reserve-Price auction variation. If NO RESERVE is displayed then the highest bidder knows that he/she will definitely take the item for the price proposed.

- A Postage and Package field is also available for the seller to optionally fill. Usually this amount is being paid by the winner of the auction and is added to the bidding price. This field can be left empty or a different means of transportation of the item can be negotiated with the seller upon the ending of the auction. However, it is always useful for the auction winner to have a vague idea about the P&P costs.

- Finally, there are two fields that inform the user about the starting and ending time of the auction. These are sometimes more useful than the 'time left' field (especially when more than one day is left until the expiration

of the auction) because they display the absolute ending time in detail. Therefore a potential bidder can plan how much to wait before they place the first bid [5].

This information is available to all users, even to those who logged in anonymously (by leaving both UserName and Password fields empty).
However, with a proper login, the user gets two extra buttons in that area: a 'Bid Higher' and a 'Add to Watchlist' button. Both do exactly what they claim:

By clicking the 'Bid Higher' button, the system pops up a window asking the user to place the desired amount of money he/she is willing to bid, and then sends a request to the server with the id number of user, the id of the Item and the amount of money that is placed on it. Then the server verifies that such a user and Item exist, the auction has not expired and the bid placed is higher than the current offer, and only then accepts it. If any other case, it returns an exception to the client and the user gets the appropriate error in a window.
If a take-it-price has been defined and the current bid is greater than, or equal to that price then the bid is placed and the auction ends.

The 'add to Watchlist' calls the appropriate remote method passing the User ID and Item ID as arguments. Then the server sends these two numbers to the database where they are inserted in the WatchlistTBL table. This way a user can keep a 'bookmark' of all the auctions he/she is interested in and see how they progress without having to change categories.

4.3.5 Keeping 'Bookmarks' of important auctions

The watchlist is again a table (it contains the same amount of information as the one found in the 'Auction Browser Window') and can be selected by clicking the 'Watchlist' tag. The difference in this display, is that the table has more space available, because the equivalent of the category tree is not necessary here.
Again the user has the option to left click a row in the table and see the detailed information on that item in the area below. There is however an additional option

[5] It is a shared secret among bidders not to show interest in an auction until the last moment, so not to raise suspicions about the item being a bargain. This is why in most such auctions the interest raises during the last few minutes. This approach however has certain risks -judging also from personal experience- because either the seller can close the auction early, another bidder may give the take-it-price and get the item, the increased traffic may be not allow the bidder to raise the bid, or simply the server may be down during the end of the auction. But... one should not complain: this is part of the excitement that an online auction gives to the potential buyer!

to remove the auction from the watchlist (attention! this does not remove the auction from the system; it is still available from the auction browser tag under the appropriate category). By pressing the button, a remote method is called which removes the UserID, ItemID tuple from the WatchlistTBL table in the database.
This tag is only available when the user logs in with a UserName and password.

4.3.6 Creating new Auctions

The final tag in the tabbed panel is the auction composer. It is only activated if the user has logged on with a correct username and password. As the name suggests it is used to compose new auctions.

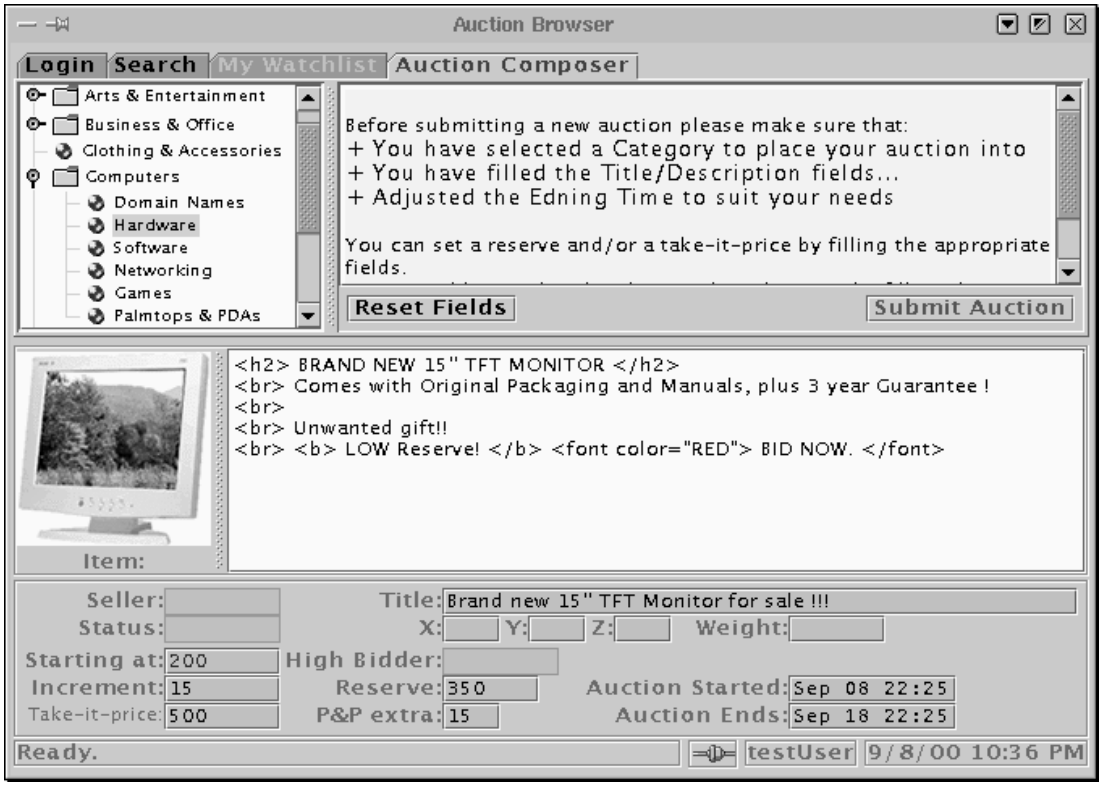

Figure 4.6: Auction Composer

A tree with the available categories is again displayed so the user can choose the place where the new auction will be placed. A small window displays simple instructions on how one can do that.
Now, most of the textfields in the middle of the user interface are editable.

The user can enter the title of the new auction and a description of the item (html tags are allowed, but no links to other web-pages/files will be functional even if entered).
Then by clicking inside the blank place next to the Description area, one can select an image file (.gif and .jpg files are allowed) to accompany the description. This file can be selected from the local hard disk where the application was executed from, and is sent to the server in form of bytes.

The seller's name, Highest bidder and Status fields are filled automatically and remain un-editable for obvious reasons.

The 'highest bid' label now changes to 'starting at' to indicate that this will be the minimum price the bidders will be allowed to bid for.

At any point, the seller can click on the textfield that displays the Starting and Ending Time of the auction; this pops up a window displaying a small calendar[6] (item shown below) which can be used to change these properties of the auction, hence its duration as well.
The default setting for the duration, as given by the system, is 9 Days; it is a reasonable value, and should be satisfactory for most auctions.

Figure 4.7: A simple, yet handy calendar window.

The user has the option to either fill or leave blank all the other fields! Indeed, as we have seen before, it even makes sense to leave Starting price field blank, and hence start the auction from one pound, as this by itself could be a reason to attract several bidders!. This however should be done only when the seller feels

[6]There where some commercially available calendar classes that provided more features to the user than the one shown above, but I chose not to use one of them (because that would oppose to my all-free implementation), and instead wrote my own.

confident enough about the item being sold, as it can be risky[7].

The reserve price label now changes to a textfield where the seller is supposed to optionally enter the minimum price which he/she will accept to receive for the item. Again, specifying no reserve price is possible and welcome by most bidders.

All the above settings can be cleared by pressing the 'reset' button.

When the user is happy with his/her selection, he can press the 'submit' button; a window is then displayed which shows all the supplied values and asks for a confirmation before placing the auction.

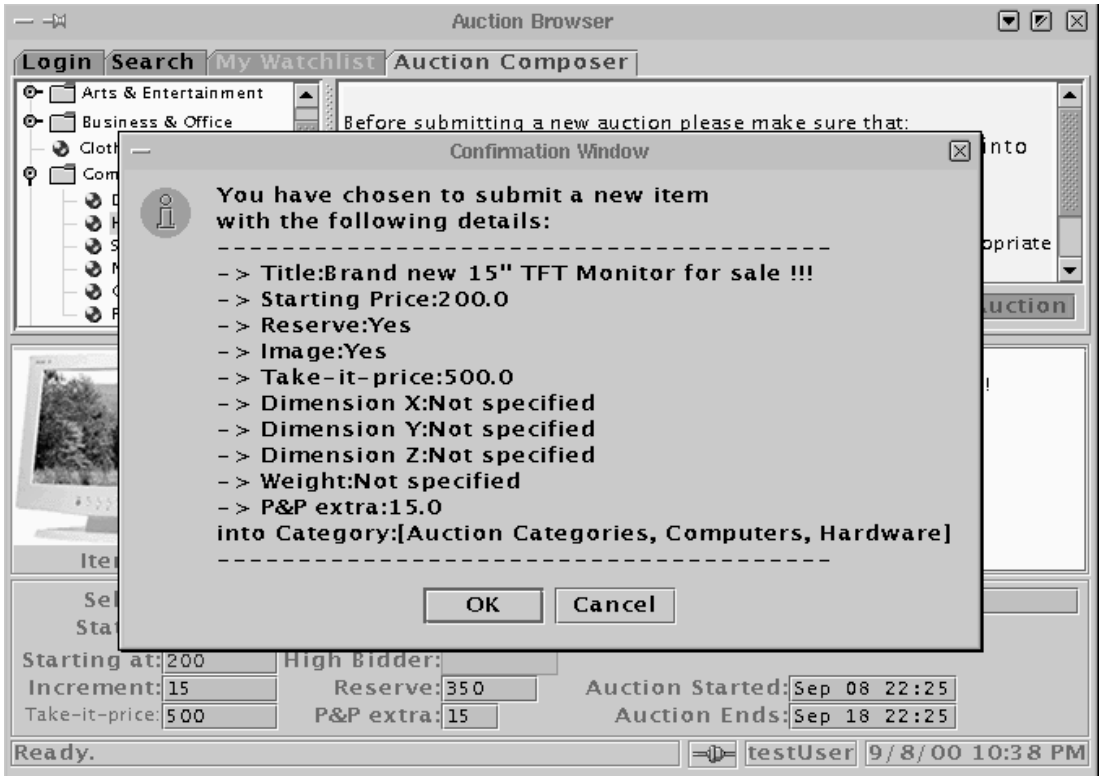

Figure 4.8: Confirmation Window

If the user agrees, then an **encrypted** object, which contains all the above information, is created and sent to the server. On the server side, the object is decrypted and all the above values are passed to the database (and more specifically in the ItemTBL), where a new row is created for that purpose.

[7]There are cases where expensive items were bought for just one! pound, only because that day a football match was on TV and interest in online auctions (and in particular in that specific category) was severely reduced

Additionally, a new entry is also made in the OwnerTBL that links the UserID with the newly created Item's ID.

Guidelines regarding the process of creating a new auction, are constantly displayed in a JEditorPane using html encoding (see Figure 4.6, top right window) in order to help the new user tackle this task safely and quickly.

4.4 Ending of auction

Up to now, we didn't mention anything about how the system determines the end of an auction and what exactly happens then...

Keeping track of the currently running auctions, is a role of the Server, and for that purpose there were again two options:
The first option was to create a separate thread for each auction that would initiate a countdown sequence. The ending time of the countdown would match the end of the auction. Threads can run in parallel, (even in single processor machines) and a counting thread would be fairly easy and not to computationally demanding task.
However, i found out that there are several problems with choosing that solution. The most obvious one is that in case of *Server crash* (or shutdown), we lose track of all the running auctions, because their threads are destroyed. One reasonable workaround would be to reinitiate all the threads (by checking the current time and comparing it with the ending time of all the auction entries in the database) upon Server startup; but that would slow down the time the Server can go into service proportionally to the number of available auctions.
Even worse, -and this brings us to the second, and maybe more important drawback of the 'one thread per auction' approach- the countdown of such long periods (9 Days for example) can be quite *inaccurate* when thousands of threads run at the same time!. That would result in an auction ending several minutes before the advertised time, which could be disastrous [8].
The above inaccuracy was verified for my implementation[9], so I tried to find a more efficient way of achieving the same goal, but without the performance penalties, or problems introduced by the previous approach. Therefore, I decided that it would be better if the Server had one thread that would *periodically* check for recently expired auctions *in the background* while serving the clients as normal. The auction checking thread will come to life approximately every minute. I have

[8] Keep in mind that -in general- most bids are placed during the last minutes of an online auction

[9] This is *not* a shortcoming of the Java language; when using threads, you know that you will not have complete control of when a specific thread will go into life and when it will become idle, as this depends also on the other processes that are currently running at that time

chosen this value, without spending too much time on it, as it can be changed very easily. However, under real life conditions, fine tuning this variable will make a big difference in the performance of the server (and hence of the clients as well). As the number of active auctions at a certain time grows up, the process of checking for recently expired ones, will become a time consuming process; therefore, care must be taken not to choose a small value. On the other hand, we want out clients to be notified of any closed auctions (the winners of an auction will particularly appreciate this) reasonably soon after the advertised ending time, so the value should not be to big as well.

To make things even less computationally demanding, I introduced another column in the AuctionTBL table (see chapter 6) which holds the state of that current auction as a bool variable (it has a value of either Active/Not Active). When an auction is created this variable is set to Active. The expired-auction checking thread which is activated approximately every minute checks the ending time of all active auctions against the current time, and changes their status appropriately. Under this scheme, the next time the thread is scheduled to be executed less auctions will need to be checked (and there is no way an Inactive auction can go back to the Active state).
Additionally, upon the expiration of each auction the conditions of the sale are checked and appropriate messages are sent to the Seller (and the Winner, if there is one). This part however, will be described in detail at a later stage.

Here I should also mention a delicate situation that arises when using the above approach: one should be prepared, as there will probably be a small (and random, although limited) inconsistency between the expiration date of the auction as chosen by the seller, and the time the Server will *realise* that this auction should be no longer active. During that time the system 'believes' that the auction is still running so it will display it in the appropriate table; the user will be able to find details about that auction and he/she might *attempt* to bid for it. This is not acceptable, so a special test condition was placed in the remote method that is responsible to handle the bidding requests, to prevent exactly that; the status field (that shows the remaining time) will also display 'CLOSED' to notify the potential bidder that no bids will be allowed for that auction.

4.5 Email notification of seller/winner

As we have seen before, the time when the auction is considered 'closed' is determined by the auction-checking thread.
In an online auction, the server program is playing a role of a referee, but unlike a classic auction (where the winner is present and also has physical access to the

item) has to notify both the seller and winner (if one). Furthermore, the item is *not* held in a store that is run by the operator of the auction program, but instead resides with the seller, who needs also to be notified for the outcome of the auction. So, for that purpose, i have chosen to use the email facilities that are provided for the Java language by the JavaMail extension package (hence the necessity of a valid email address during the registration phase).
Both seller and highest bidder are sent an email, shortly after an auction expires, with details such as the Title of the auction (for verification purposes), Highest Bid, Postage and Package information. The winner also gets details on how to contact the Seller (Address, Tel no, email, ...), and the seller is sent equivalent information regarding the winner of the item, as an additional assurance that contact will be made between the two parties involved.

For that purpose a small part of the Sun's JavaMail extension for Java is used. This is so, because as you probably have realised, the application server is only used to send outgoing emails and does not have to support other functions (such as users being able to log in and check or manually create new messages).
Strangely enough, everything worked fine when dealing with that package, and integration to my program was pretty straight forward. This is mainly due to the excellent documentation that accompanied the software.

4.5.1 JavaMail Details

A central point to sending an email from a Java program is the 'Session' object which manages the configuration options and user authentication information.

Such options include the mail transport protocol and the hostname/username the mail is sent from, and have to be specified, although default values can automatically be generated for most of the cases.
The authentication information controls the security aspects of the Session object. This is not necessary in my implementation, as the server is only used to send emails, without waiting for any replies.Indeed, after the dispatch of the emails, the server is no longer involved in the transaction; the remaining details (if any) will have to be negotiated between the seller and the winner of the auction.
The field that describes the IP of the sender is intentionally changed to 127.0.0.1 which is a dummy address and is normally used by machines that are not connected to a network. With that 'trick' we get sure that the machine that hosts the auction Server, will not get any replies by careless users that pressed the reply button on their mail client programs.

Next, we need to say some things about the message object which contains the body of the mail, along with some vital properties (the header) that must be set before it can be passed to the above session object. These parameters are neces-

sary to send, route, receive, decode and store the message. We can also specify the message content type (text/plain or text/html), encoding and structure.
For the purposes of the program, we will create a simple MIME encoded message and set the Recipient tag to point to the Seller's or Buyer's email accordingly. Just as to make things easier for both parties, i set the (optional) Reply-To tag in the message header as well, so exchange of messages between seller and buyer is straight forward.

Chapter 5

Security/Encryption

I left a separate chapter for security, mainly because it is an essential part of the Implementation, and secondly because there are a lot of things that need to be said.

First of all, I think I should introduce some definitions of the terms that are used in the security related domain.

Encryption/Decryption. Encryption is the process of taking data (also known as cleartext) and a short string (a key) and by using them together, produce data (called ciphertext) which will be meaningless to a third-party who does not know the key. Decryption is the inverse process, in which by applying a short string (the key) to the ciphertext object, produces the initial data. Although there are some Encryption algorithms that are irreversible (like the one used by Unix systems to encrypt passwords), most of the time, when one encrypts some data is because he/she wants to transfer it securely to another location where it will be decrypted, hence a reversible algorithm has to be used.

Password Based Encryption. Password Based Encryption (PBE) derives an encryption key *from* a password (or any string). In order to make the task of breaking the key, (and getting the password from it) very time consuming, most PBE implementations will mix in a random number known as salt, during the key generation process.

Cipher. Encryption and Decryption are done using a cipher. A Cipher is an object capable of carrying out encryption and decryption according to an algorithm.

As I mentioned before, the main reason why encryption was necessary in the implementation, was the fact that Personal Details (such as Telephone num-

bers/emails/Addresses and optionally Credit Card Numbers) would have to be communicated (over the Internet) between the client and the server during the registration phase; Likewise, username and password is passed to the server for confirmation during the login phase. Since the JacORB did not inherently provide SSL encryption[1] for the data that is communicated among its peers, I had to manually encrypt some of the objects. I decided that (for the sake of speed of execution), only the absolutely essential information should be encrypted. That includes everything that is contained in the Personal Details object and the User-Name/Password Strings.

One approach to that problem would be to have a secret password, hardcoded in each client (that would serve as the key to the encryption algorithm) and another copy of it on the server side. Then use PBE encryption, encode and transfer safely all sensitive content. This is the simplistic approach that works, but is not very secure. There are several ways one could get access to the plain-text representation of the data. The simplest would be for one to manage to 'decompile' the distributed .class file using a tool and from there get access to the source code! Then, it is just a matter of reading speed before one finds the password. Knowing the password and having access to the encoded data that travels through the wires (this is easy to achieve and is known as the man-in-the-middle attack) it is fairly easy to get the original content by following the instructions for decryption!.
Another would be for one to manage to 'crash' the application and make it 'dump core[2]'. That file normally contains what is included in the memory area occupied by the program at the time of the crash. Although now the task of finding the password string is more complex (most of the contents of the core file will be gibberish) it is possible for one to get the password. Access to the original (unencrypted) data is then trivial...

So, two things must be done to enhance security in the program.

- Make sure the key (or the string that generates it) is not statically stored in the client (or even in the server) so that a malicious user with access to the source code, will not be able to reproduce and use it to decrypt sensitive information.
- Make sure that the key changes every time an encryption sequence is required, so that even access to the contents of the memory that is currently used by the program becomes useless to the man-in-the-middle.

[1] This is valid until the time of writing of this thesis; support for SSL encryption is announced to be included in the forthcoming version.

[2] This is a common technique, followed by hackers to get access to restricted areas in memory. Although this is known to work with most other programs, I personally couldn't reproduce it for the Java Virtual Machine.... This doesn't mean that it is not feasible. Most probably my hacking skills are not good enough.

The latter observation, generates the problem of 'key distribution' between server and the different clients. It is a vicious circle: we want to distribute a key that will be used to encrypt sensitive information, but we need to encrypt *that key* as well... after some point we end up using more resources encrypting the encryption keys, than to handle the rest of the program.

There is however, a more 'elegant' approach to the key distribution problem:
Some cryptography algorithms can produce a unique **key pair** that consists of a Public and a Private part, with certain characteristics. There is an absolute and unbreakable relation between the two parts of the key, but one is almost impossible to derive one part of it by just knowing the other. So one can freely distribute his Public Key, and let people encrypt data for him using that key. These encrypted data, can however, be decrypted only by applying the private part of the same key pair (which the user has to keep safe).

This is the power of the public key encryption algorithms.
Some very popular public key encryption algorithms that are used today are the: DSA (not to be confused with DES), which however is NOT reversible and therefore can be used only for digital signatures RSA (Rivest-Shamir-Adleman) which is reversible and can be used to encrypt, as well as to digitally encrypt data.

The first one is included in the Java Cryptography Architecture (JCA) that comes along with the standard JDK distribution and is available for general use[3] although support for the Cipher object that does the encryption is provided by the Java Cryptography Extension (JCE) that became available to download only lately by people outside the USA. During the time of the implementation only a beta version was available that had some serious -and well known- bugs. [4] Most importantly however, that beta version did not support the RSA algorithm (and no support was planned for future releases either) which I had to use for my program, so I downloaded and used a third-party alternative instead, which is called OpenJCE.

When the server is started, a unique public-private key pair is generated locally. Each time a client wants to submit some sensitive information, it requests the public key from the server, gets it, and uses it to encrypt the data. Then the client sends a byte stream of encoded data, which is assembled on the server tier, and then decrypted using the private key.

[3] That is probably because an encrypted stream of data with the DSA algorithm is completely useless, as it can never be decrypted.

[4] I personally came across some of them when experimenting with JCE, and spent some time to find a workaround, but in vain... By reading some articles posted by other users who had similar problems to me, I realised that I had to either upgrade to JDK1.3 (which was still not available for the Linux platform, which I used for -most- of the development of the application), or to use a freeware alternative to Sun's JCE.

The alternative to sending encoded bytes, is to use the SealedObject class from the JCE (or OpenJCE in my case) which uses the acquired public key to encrypt an object as a whole. That could have been used (and was actually my first approach to the problem) for the encryption of the PersonalDetails object. However, the resulted object is of type javax.crypto.SealedObject and has no detailed specification.

The problem with such 'generic' objects is that in order to be transfered to a remote location (via the ORB) they have to be mapped to a type 'Any' in the CORBA IDL file. This -at least- is the theory; I followed the IDL documentation on how to achieve that, and at some point i got a 'NON_IMPLEMENTED' exception from JacORB, which means that the feature i was trying to use was not yet implemented (JacORB never claimed to be a FULL implementation of the CORBA standard).
Luckily the solution to that riddle was pretty straight forward: I just had to create a ByteArrayInputStream and feed it with the original object. The output is an array of bytes, which can be encrypted by passing it to the (fully initialised for encryption) Cipher object. The result is an array of (now encrypted) bytes, and bytes is a known type to the ORB and easy to transfer.

Finally, I should also mention that additional logic has been included, so that each user can have his/her own pair of public-private key pairs, that will be created upon registration, and then kept in the database, although this feature has not been implemented, as the above mentioned security measure seems more than enough.

Chapter 6

Database Design

Designing and deploying the database tier proved to be the most time consuming and demanding part of the whole program.

The first task was to determine the logical entities in the program and the to select how to split them into tables; then i had to distribute their characteristics among these tables and to filter out all unrelated information. While developing the program, i changed the design of database several times, partly because i realised that my previous approach was not optimal, and partly because a feature that i assumed was supported by the database server, was not implemented yet. However, here i will present the final layout.

6.1 Table Description

First of all, one separate table is needed to hold the personal details of the seller, such as Username, Password, First/Last Name, Address, Tel/Fax no, Address, Post Code, Country and email address. All these fields take strings of variable size as input, so the VARCHAR SQL type was chosen to describe them. A UserID field is also needed, that serves as the table's primary key, and it is used to join related tables together. This field is an integer (i assumed that 5 digits should be enough, providing support for 99999 registered users) that is automatically increased when a new entry is made to the table.

The second obvious table was the CategoryTBL that contains the necessary information needed by the auctionClient to build the Auction Category tree. As I mentioned in section 4.3.3 , a triple of the CategoryID number, the Name of the Auction Category and the number of the parent Category number, is the minimum required to be able to dynamically construct the tree. A special row with CategoryID=0 and name 'root' is inserted in the beginning of the table, to

hold the 'base' upon which the rest of the tree will be built. The CategoryID is again a positive integer and serves as the primary key of the table. The Name is a string with variable length, and was designed to hold up to 25 characters The last part of the triple is again a positive number that ranges from 0 to the maximum number that exists in the first column. I called it ParentID, for obvious reasons.

The next two tables were not so trivial to design: I decided to differentiate the notion of an 'Item' from that of the 'auction'. By this i mean that consider i them as separate logical entities for the database design. This was done mainly to minimise redundancies and to boost performance. An 'Item' could in theory take part in more than one auctions (in case no buyer is found). So, the following two tables were created:

The ItemTBL (which is also the biggest in the database, with 17 columns) contains STATIC[1] information about the item and the auction it is related to. Although one would expect to see some of these information in the AuctionTBL (described below), there were reasons why i have chosen not to, and i will explain them at a later stage. So in the ItemTBL, data that relates to the Description, physical dimensions, Postage and Package extras and weight of the item is kept along with the Title of the related auction, the Starting/Ending time, the Starting Price, the Reserve Price, the Bid Increment and the Take-it-Price. I should mention that not all fields are needed to be filled; Null values are allowed to most of them. Lastly, this table hosts the 'Is_Active' column [2], which keeps the status of the auction, and is described by a boolean SQL field.

On the other hand, the AuctionTBL, contains fields that are expected to change frequently, such as the Highest Bid, the Number of Bids, and the name of the Highest bidder; a private key for the table (AuctionID) is also included for indexing purposes. Finally a column 'IID' is 'imported' as a foreign key from the ItemTBL, so a connection between the two tables can be established when required.
The initial idea was to insert a NEW row to that table for every bid that is placed. A selection of all the rows that had the same IID number, sorted by the AID would provide a useful history of all the bids being placed on that item. However i was once again let down by the software, which didn't support the necessary SQL features[3]. So I changed my approach, and I am now updating the

[1]These information are expected to remain unchanged during the auction lifetime.

[2]I mentioned the necessity and the function of this column, when discussing the auction-checking thread

[3]For that SQL query , subselects are required, and they were NOT supported by MySQL 2.22.32 (or even by the latest beta:2.23.22). Therefore i posted a related question to the MySQL newsgroup, and got a reply by one of the developers of the database, saying that this feature is planned to be included in the next major release (however no specific date was given for that). I was presented with an alternative, which was known to be completely inefficient, and was supposed to be used only as a last resort. I therefore decided to keep all the necessary

appropriate fields (Highest Bid, Number of Bids, name of Highest Bidder) in the row that has the appropriate IID.

With this latter approach, if I had joined Auction and Item tables together, I would have no redundancies or performance penalties, since I am only *updating* a few fields in a row; however, if the original approach had been taken (which supported keeping bidding history), then I would end up having a *new row* for every bid, with only 15% new information (The joined table would have a total of 20 unique fields, only 3 of which would change in every new row). The rest would be repeated unnecessarily, contributing to a huge redundancy. With the tables split, I still get a descent performance [4] and have the added advantage of more flexible and manageable tables.

To continue with the rest of the tables in the database, the OwnerTBL makes the connection between the User and the Items he/she owns. Therefore this table contains a column with the unique number that characterises a specific user (UserID) which is imported as a foreign key from the UserTBL; the other column has the equivalent number from the ItemTBL. Of course a user of the system may have created more than one auctions, so in this table we may have more than one rows with the same UID (however no tuple has both fields the same)

Exactly the same fields (UserID and ItemID) are contained in the last table, the WatchlistTBL. Although here, the meaning of a tuple is different: it specifies the unique ID of the items that are kept in the Watchlist of a specific user.

6.2 MySQL

Just to wrap things up, figure 6.1 displays all the Tables that were created and in addition shows the relation between them.

MySQL is well known to be blazingly fast for fairly simple operations and this is why it is a very good companion to web-based applications [5]; However, MySQL is not fully SQL92 compatible (let alone SQL95), so some advanced features are still missing:

- It has no support for transactions. In database terminology, "*a transaction is*

background support, but to forget about that extra feature i wanted to add to the program... With support from the database, in the future, retrieving the bidding history would be just a matter of changing a few lines of code...

[4] There is a tiny performance penalty, because i have to join the two tables together (using the common field IID), but this takes less than a fraction of a second in MySQL...

[5] Yahoo! and Google search engines are known to use MySQL as part of their database backends... Their high performance is easily noticed :-).

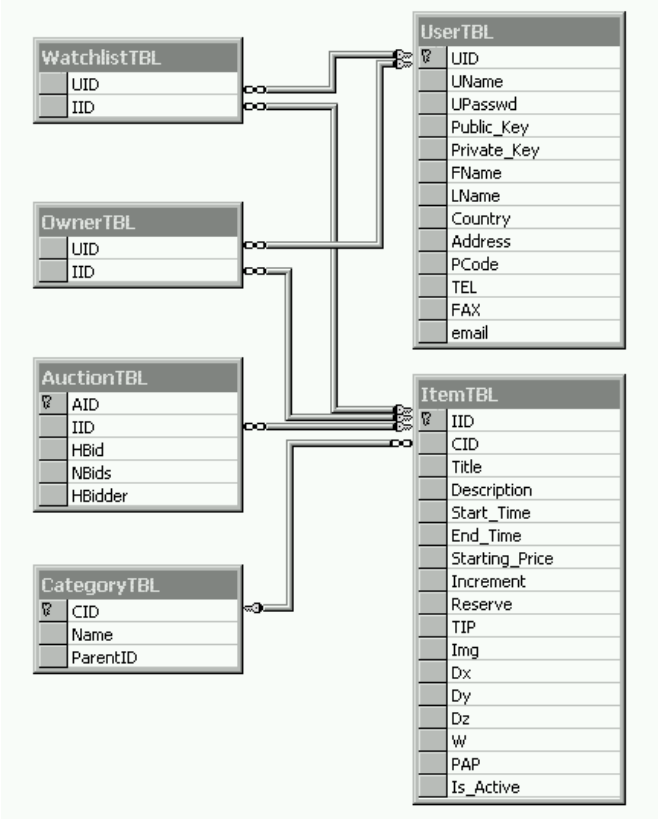

Figure 6.1: Tables in the auctiondb Database.

an abstraction of change, in which individual modifications to the database are aggregated into a single large modification that appears to occur either entirely in a single moment, or not at all". Transactions play a major role in the database industry, and one is initially surprised by the conscious decision to not support them in MySQL[6]. However, as explained in the appropriate chapter of the manual, an alternative solution is advised by the database developers, which achieves almost the same results, but gives better performance!. Luckily, i found the alternative solution convenient for my program, and used it without any second thoughts.

- MySQL also has no support for subselects. Again, a feature that is considered very useful, but is still absent from the current version. Unfortunately the solution to this problem was too inefficient to use. This was the main reason why bidding

[6]I should mention, that MYSQL can optionally be compiled to support transactions (this is true **only** for the last series of beta versions, currently being tested) but this feature is currently under heavy development. The funny thing, is that people *still* find the version without transactional support, easier, faster and more efficient to use!

history was not implemented (see section 6.1).

- Finally I am tempted (although I is not a complain) to mention that MySQL didn't have the nice/easy-to-use GUI-based utilities that can be found in other commercial databases. One has to spend enough time to learn all the command line parameters of the supplied programs in order to setup and configure the server, and to manage the related data.

6.3 Microsoft's SQL Server

For the reasons mentioned above, I considered that it would be really useful to use another (commercial this time) database to test my implementation with, and I've chosen Microsoft's SQL Server 7.0 for that[7].
This database product has been used for many years, and hence has become more mature and robust than other alternatives. In addition, the plethora of utilities and administrative programs that come with the database, allow even the novice user to perform fairly complex tasks (such as replication, backup or repair a damaged database) with great ease. The provided GUIs are easy to understand and use.

Setting up the database in SQL Server was not a very difficult task, although I had to get accustomed to the new jargon introduced by Microsoft into the syntax of SQL[8]. The same tables and design philosophy were used, as the ones described above for MySQL. Adding keys and relationships between the tables was easily done, using drag-and-drop. (Also, the snapshot of the database tables, was taken from MS SQL).

Several changes had to be made to the part of the program that queried the database server, partly because of the different syntax of MS SQL, partly because I wanted to take advantage of the fact that transactions were fully supported by the database.

Up to that time, the whole program was being developed mainly under the Linux platform, but MS SQL required a windows environment to run (preferably Windows NT, although it works under Win98/ME as well), so I had to install the

[7]Although SQL Server 7.0 is a very expensive piece of software for an individual to buy, i was fortunate enough to have won a 'Universal MSDN Subscription' from Microsoft earlier this year; therefore i had full access to almost all the software made by Microsoft (hence to SQL Server as well)

[8]This depicts the scenery in the database industry, and how it has changed over the years: Although most software products try to conform to certain standards nowadays, commercial database packages do exactly the opposite by introducing several proprietary commands and functions, and thus making life of software developers harder...

necessary software required by the server tier to the windows workstation; having done that, the rest were easy: the majority of Java classes could be used without changes under windows as well (only the database related part had to redesigned, as explained above).
MS SQL uses a technology called Open DataBase Connectivity (ODBC) to allow applications to connect and retrieve/update from a database source... ODBC is the usual way Databases and applications talk to each other under the Windows Platform.
However, Java uses another technology for the same reason called Java DataBase Connectivity (JDBC). These two technologies are not compatible with each other and unfortunately, there is no JDBC driver for MS SQL in the Public Domain [9]. The common solution to that problem, is to use something known as JDBC-ODBC bridge; it is an intermediate driver that translates all calls from the java program to a format that can be understood from ODBC; from that point on, ODBC performs another translation, specific to the database so that the request reaches its target. There is however, a small performance penalty, when compared to a JDBC driver (which talks straight to the database), but unfortunately there was little i could do about that part.

In the next part we will see performance characteristics of the program under different environments and configurations.

[9]There are about two or three commercial JDBC drivers available for MS SQL, but are very expensive: some cost more than the database itself!

Chapter 7

Tests and Benchmarks

The performance analysis of a client server program is not a trivial task. Several parameters have to be taken under consideration; Many computational components that have to be evaluated and they are split across the various tiers. The main idea however,under all performance tests is to be able to measure either the CPU load under a reproducible sequence of events, or alternatively, measure the elapsed time that this sequence has taken. I used the latter approach, using Java's internal method for getting the system's clock with millisecond accuracy.

Several properties of the program can be measured, by running the same tests under different conditions. For that reason, i wrote a separate application (command line driven, with no GUI) that runs a controlled sequence of remote methods. These methods are also used in the original client program, but here we are more interested in their speed rather than their results.

More specifically i will try to measure:

- How network latency in conjunction with the ORB affects the communication, by running the benchmarking program from two different locations

- How well the application server can handle several simultaneous clients, (by running a different flavour of the test) and measure the system response each time.

- Finally i will try to simulate the behaviour of a 'typical' user, to give a qualitative evaluation of the implementation as a whole.

Results of all these tests appear on the screen in a compact form, and at the same time are written (or appended depending on the specific test) to a file in greater detail, so that further analysis can be performed.

7.1 Network Latency/ORB test

Because physically running the actual auction browsing program from different locations, that are far from each other, is difficult, i devised a simple test that would execute a series of remote methods several times and register the delay to a file. We want to make sure that we have no slowdown from other processes that run at the same time in the system and make use of the network, so threads will *not* be used in this part; the program will stop and wait for the results, until it makes a new request.
Each invocation of the test appends to a file, so averages can be computed later; this is essential because random factors can affect network latency.

The remote computer used for the tests was located in Greece (Aristotle University of Thessaloniki, Department of Astronomy & Astrophysics) and had almost the same hardware and software configuration as the local machine: both were running RedHat Linux v6.2 with kernel 2.2.16, and had the same version of JDK installed. The local machine was a Celeron 400MHz/128MB system, the remote had PII 400MHz/128MB. This way, any inconsistencies related to factors other than the network were minimised.

This part of the benchmarking program mainly uses the secure/insecure registration and login methods of the auction system. In addition to network latency, secure tests are also CPU intensive (because they include encryption/decryption algorithms) and give results proportional to the speed of the computer which runs the application server[1].
In each of the four individual subtests (insecure/secure login and register), System time before and after the invocation of the remote method is measured at the client; separate measurements are taken after the first and second time, and then for a number of times "x" given by the user as a parameter to the benchmarking application. The line that is appended to the result file has the time of the first and second request as well as the *average* of the multiple invocation sequence.
Although measuring times in this great detail may seem trivial[2] , we get different delays for the first and second request:
Because the ORB can cache requests that come from the same location, the second request takes -in general- less time to get processed than the first!.

Thirty four numbers are taken by each invocation (they represent ONE line in the result file); the line ends with the number "x" that was passed to the testing program, and it will be used at later stage.

[1] Assuming that CPUs of the same architecture are used in all tests

[2] One might think that I could have just used 'ping', or 'traceroute' to find network latencies, but the ORB behaves differently from an ordinary program that makes remote calls

The whole test is executed several times for increasing x values; the main goal is to make sure that we get a fairly constant response from the remote methods, as this number goes up. To further minimise any unpredictable network latencies, the test for each individual x is repeated five times. Lines that have the same x value at the end, represent the exact same test, so they need to be averaged before used in the plots. This is done by a small awk script that i have written for this purpose; the output is redirected to a new file, that will be used for the graphs.

The same test is of course executed from the remote and local computer the same number of times, following the exact same procedure. Therefore the resulting graphs will have data for both instances.
However, because the behaviour we get is fairly similar for the most remote methods, only a small part of the results will be presented here. The full range of graphs, along with a description of the unix scripts that were used in the process and a sample result file, can be found in the appendices.

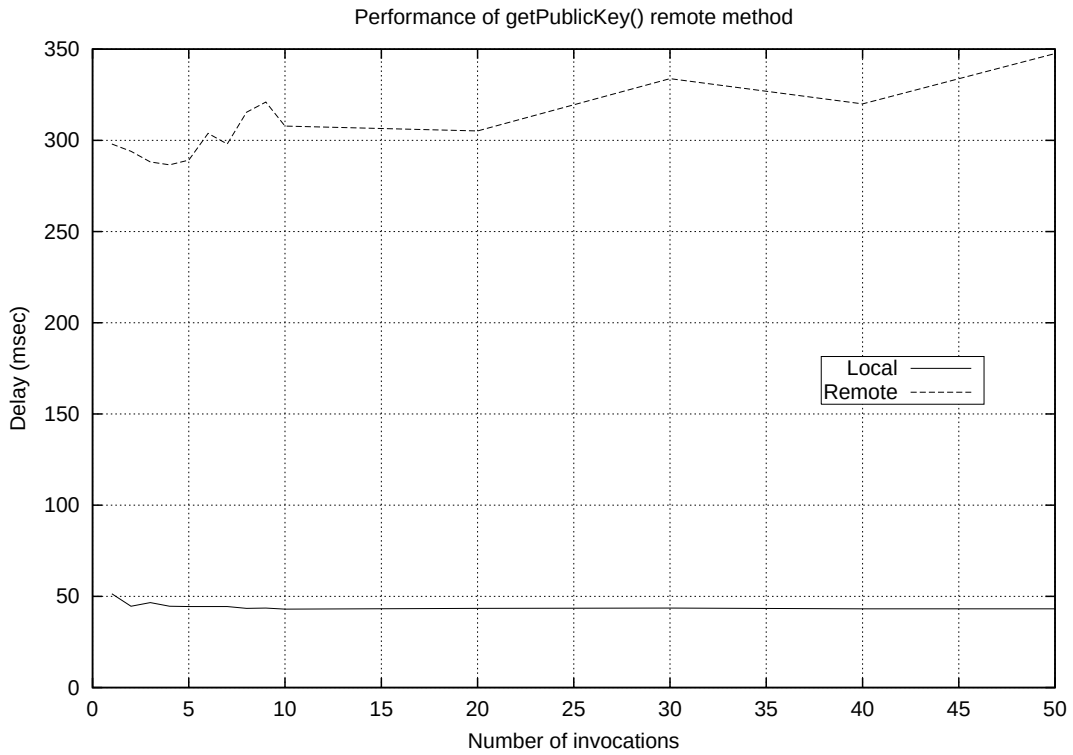

Figure 7.1: getPublicKey() for Remote and Local Computers

Figure 7.1 represents the getPublicKey() remote method. The delay is displayed in the y-axis, the number of invocation of the test (parameter "x") is displayed in the x-axis. The dashed line represents the set of results that were gathered from the remote client, while the continuous line represents the local client.
We can easily see the local client gets the results six times faster (approx. 50ms instead of approx. 300ms); both clients seem to give fairly constant response as x goes up (although the remote one seems more 'nervous').

The results shown in Figure 7.2 represent the local method call that gets the cipher instance. It is a CPU-only test (no remote method is called, hence no network relation) and was included, to measure the difference in execution speed from each machine.
The results are (more or less) expected: they show that the (remote) PII machine performs slightly better than the (local) Celeron one.

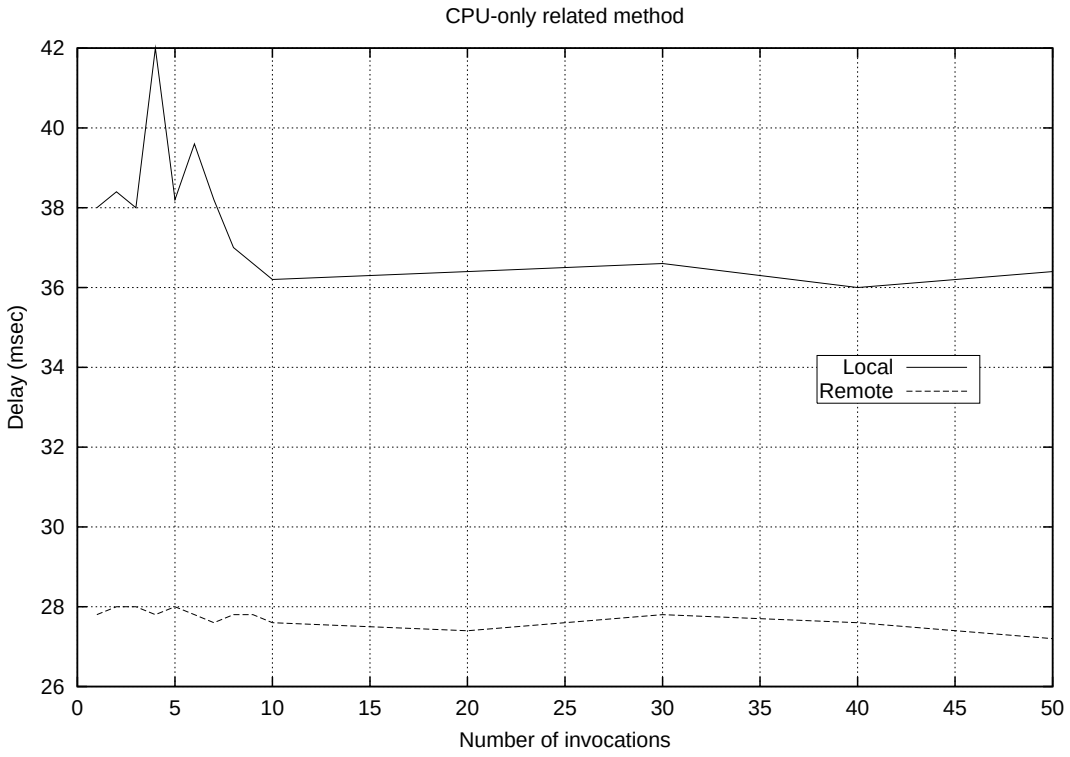

Figure 7.2: Cipher related method for Remote and Local Computers

Another set of results is presented in the pair of graphs shown in Figure 7.3. These graphs represent the remote method that performs insecure registration:setPersonalDetail (it was replaced in the actual program by the secure one, for obvious reasons).
The top one has results for the first invocation of the method, the bottom has

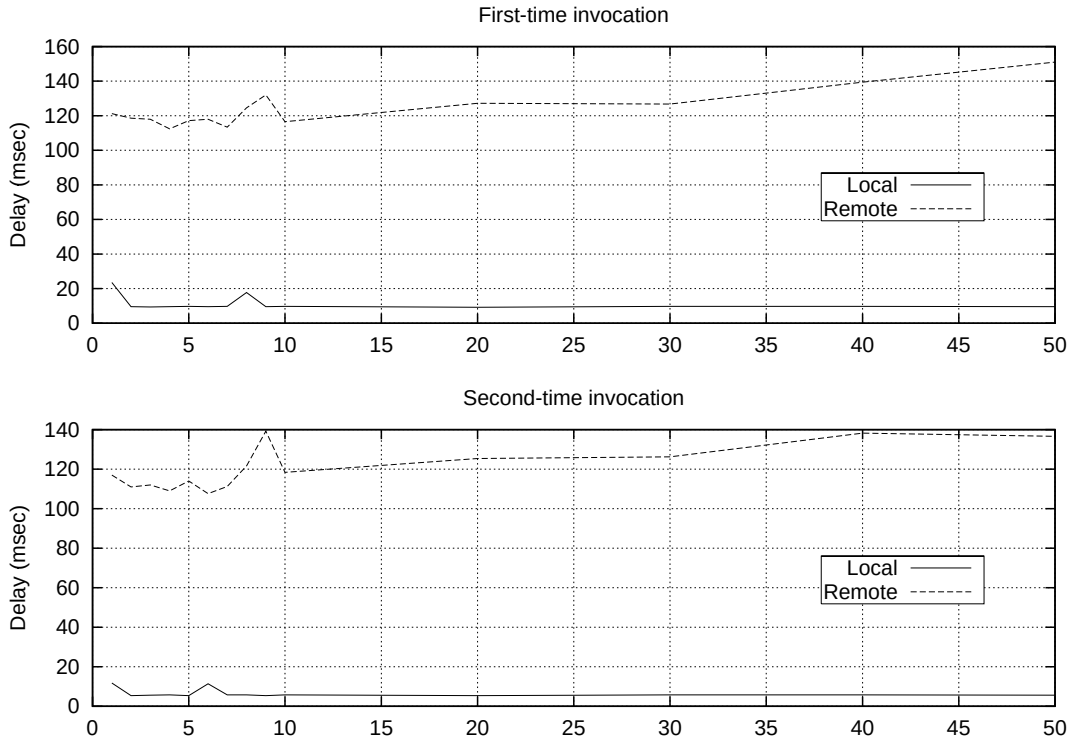

Figure 7.3: setPersonalDetail() for Remote And Local Computers, first and second invocation

results for the second invocation. There are some conclusions that one can draw from these two graphs.
First of all, the local client is approximately thirteen times faster than the remote one.
Then, the second invocation for the local client is faster than the first one.
This however, is better seen in Figure 7.4 that shows the first, second time and the average for many invocations.

In this diagram we verify that, the second invocation has 40% less delay in returning the results. We expected this behaviour, since ORB caches requests to remote methods, made from a specific IP address. For the same reason, invoking the same method many times results in a even faster reply. The big difference however, is between the first and second time.
Similar behaviour is *not* displayed for the equivalent calls in the remote client.

The remaining graphs follow the same idea: where CPU-only operations are performed, the remote computer gives off better results; where remote methods are

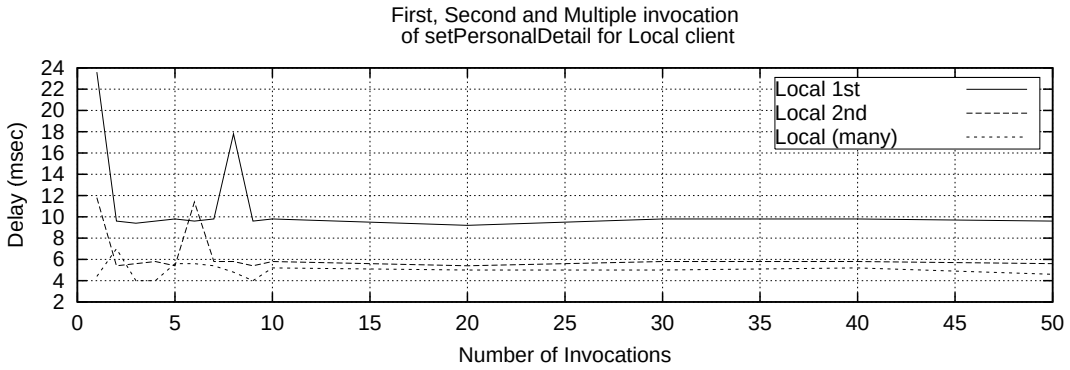

Figure 7.4: Difference in delay of a remote method versus number of invocations

used, an increase of thirteen to twenty eight times is observed in the delay of the remote client compared to the local one.

7.2 Stress Tests

In order to measure how the Server, the Database and the ORB behave under heavy load, i will try to invoke hundreds of remote methods on the server within a very short period, and by that emulate a stressful situation.
For that reason, java threads *have to* be used. Indeed it is not enough to just invoke these methods from a normal, 'sequential' java program, in which case we are bound to wait until a remote method returns the results, in order to make another invocation.
By using threads, we have hundreds of remote method requests, happening at approximately the 'same' time. To further make sure that as many threads as possible are running at a certain point, i *first* create *all* the thread objects, and *then* invoke the 'run' method for each one. This way, the startup time of each one is minimum.

There are two different flavours of these stress tests, each one focusing in a different aspect of the system.

7.2.1 Server Stress Test

For this test, i have chosen a remote method that performs a relatively inexpensive database query, which preferably returns a small amount of data. The reason is,

that i am primarily interested to see how well the Application server tier handles hundreds of requests in a short period of time and at the same time minimise the contribution of the database[3].
The ideal method for this purpose was get_servertime(), which gets the local time from the database.
Since I don't have to pass any parameters to that method, i minimise the delays created by the network (or anything that is between the application and the database servers) as well.

So, a wrapper method called stressSRV was written in the Application server that registers the current system time (with millisecond accuracy), invokes get_servertime(), and then immediately after it gets the system time again. By subtracting the two times we get a good approximation of the time penalty caused by get_servertime. Notice that all these measurements take place at the server side, so that network latency between the server and the test client does not alter the results.

Apart from the delay, in this test we need to know when exactly each thread was activated. So, a special struct has been constructed for the return type of the testing method, that includes the starting time *and* the delay of each invocation. With this data, the client knows **when** the method was invoked, and **how long** it took to complete.

On the client side now, we create a series of threads (the number is specified by the user) that simply invoke the remote method, get the tuple (starting_time,delay) and store it in a file, along with the name of the thread [4].
This way, we know which thread was executed and when, and how long it took to return the results; by sorting them by the starting time of the thread, we can also get the full history of the event.

Because the test machine was powerful enough and, the database was running in the same machine as the application server, even for 300 threads, the whole sequence takes no more than 1.8 seconds.
From the data we collect, we can draw the following graphs:

This is the most important diagram, as is has all the information that we need for this test; It shows when each thread started, and how long it took to complete its task.
The vertical y-axis was initially showing the absolute time, but was converted to show the time elapsed from the beginning[5] of the test. We also see that the test took less than two seconds to complete.

[3] If the database query was complex, the Application Server would have to wait for an answer from the Database, and that would slow things down.

[4] Each thread is automatically given a unique name by the system, in a format like: Thread-1,Thread-2,...

[5] as denoted by the invocation of the first thread

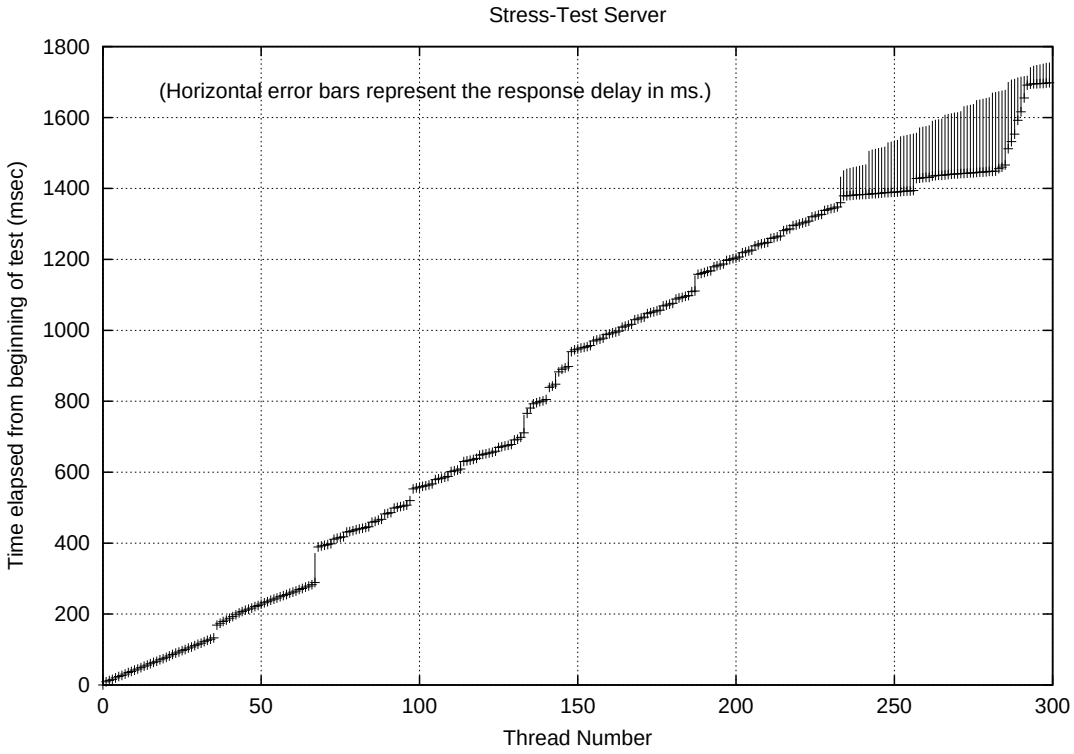

Figure 7.5: Number of threads vs Time

So, each cross in the diagram denotes one unique thread; by projecting to each axis, we can find its number and the relative time that it was executed. Additionally, the vertical 'errorbars' above some points, show the time that thread took to finish.
For the first 238 threads, execution is approximately linear; 50 threads take around 200ms to execute, which gives us an average of 4ms per thread.
An interesting thing happens in the last few threads, which for some reason are executed in batch; indeed, the angle of the last few crosses is less steep, which means that more threads are executed in less time.
Actually, 22 threads were started in 15ms which means 0.68ms per thread (an 83% increase). However, as we see from the errorbars, these threads take more time to finish, resulting in approximately the same rate of execution.
This is expected, as we are using a uni-processor system. Threads may be interlaced, but the actual rate of execution is limited by the speed of the CPU. If a multi-processor system was used instead, and provided that the operating system made use of that fact (Linux with SMP support for example), we would have gotten a drastic increase in execution rate in that last part of the diagram.
The ORB has a property file that specifies the minimum and maximum number

of threads that can concurrently access it. For this test the default values were used (min=5, max=50) [6].

The remaining two diagrams, are based on the same data, but focus on different aspects of them. Figure 7.6 shows how long each thread took to complete. It is equivalent to the vertical errorbars found in 7.5, but has greater detail. Again we see that the last few threads (which were started in a batch) take longer to execute.

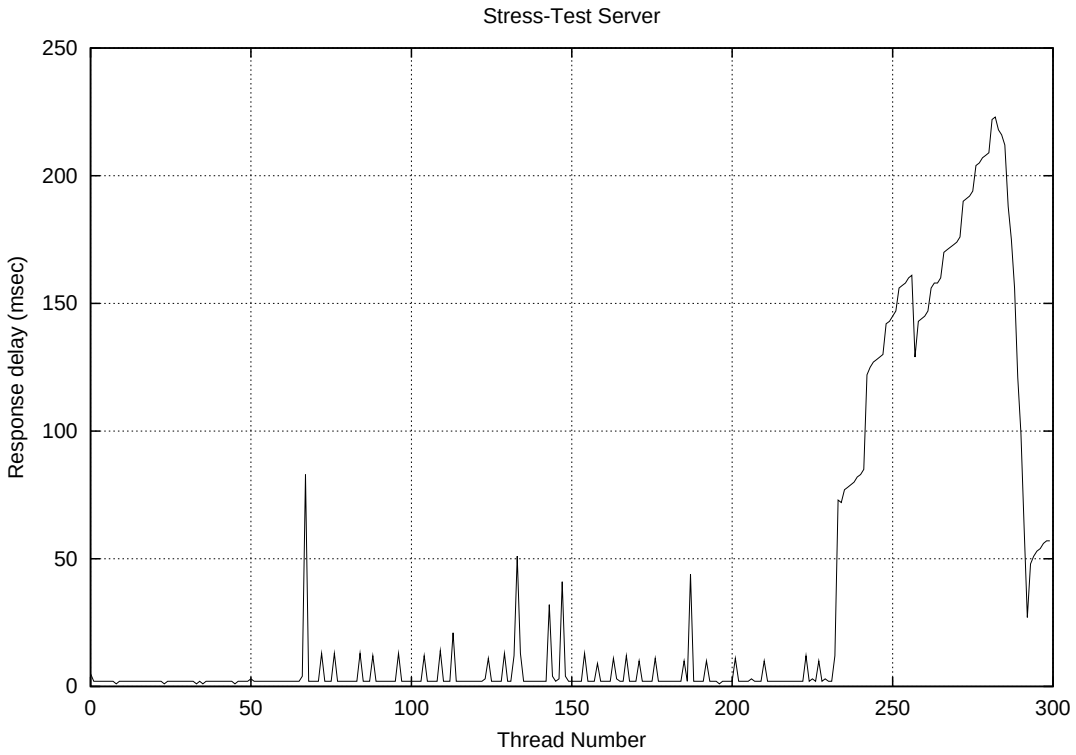

Figure 7.6: Number of threads vs result delay

Diagram 7.7 shows a the delays versus time (is like a 'history' of the delays).
It is expected to be fairly similar to 7.6, because threads are also *started* sequentially.

[6] I also experimented with using max=150, and -as expected- more threads were processed y the ORB (the same behaviour that was found in the last part of the 7.5 diagram). This will be appreciated my owners of multi-processor systems, and has to be fine-tuned according to the usage pattern of the system.

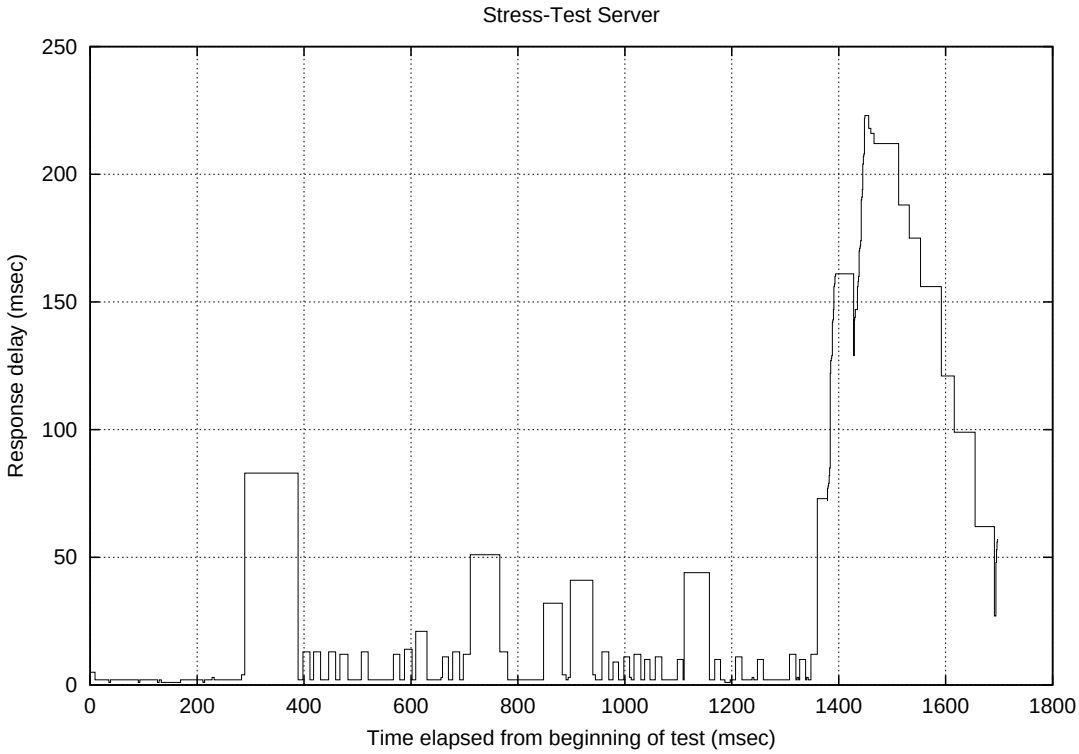

Figure 7.7: Elapsed time vs result delay

7.2.2 Database Stress Test

In this test, similar approach is taken as the one described above. The only difference is, that the remote method that is selected for the test puts some pressure on the database as well. For that reason buildTree() is ideal: again there are no arguments that are needed to be passed to the method, and (for testing purposes) we don't care about the results either.
In this test, 56 rows are returned from the database to the application server with each invocation. These are put into an object and sent to the auction client; however for testing purposes we will not use the results... This is bound to take definitely more time, than the simple get_servertime() used in the Server Stress Test.
We again create and then execute a user defined number of threads that uses the same remote method, and get the tuples (starting_time,delay) at the client side. Results are written to a separate file.
From the collected data, we draw the following graphs:

This graph has the same philosophy as 7.5. It shows when each thread is exe-

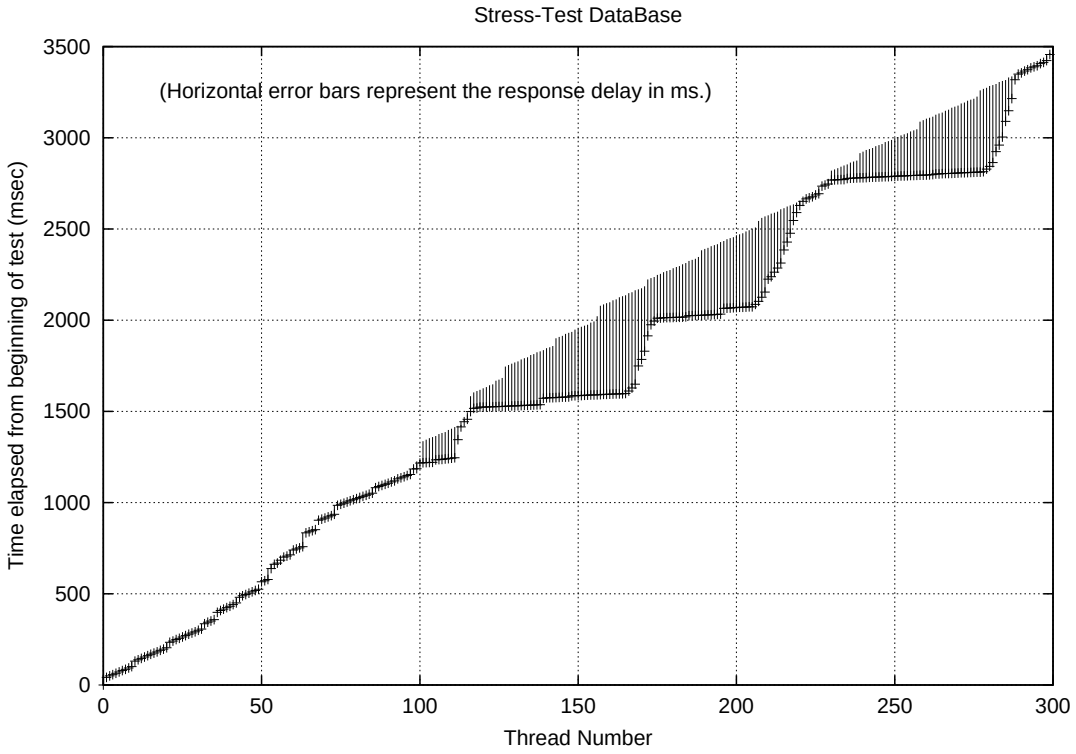

Figure 7.8: Number of threads vs Time

cuted, and how long it takes to complete. Please notice however, that for this stress test, the total execution time is almost doubled (3.5secs). This is expected, as part of the computation is done in the database tier.
Another characteristic that one notices is that more threads are started simultaneously; this is again expected: a thread is started, but has to wait a few milliseconds for the results to come from the database; in the meantime, the system starts the next thread which shows the same behaviour. This results in more 'horizontal regions', and therefore in more vertical errorbars. Note however, that the slope is again fairly constant, showing that the system can easily handle this load. Also note that the total number of threads running *concurrently* (the number of crosses found in the 'horizontal' areas) are approximately 50, which is also the max_thread limit found in the ORB property file. As i mentioned before, this can be changed but a higher number would make more sense in a multi-processor environment.

Figures 7.9 and 7.10 display the delay of each individual thread, and the 'history' of responses for the database test; the same logic applies as the one explained in the previous section, so no additional remarks will be attempted here.

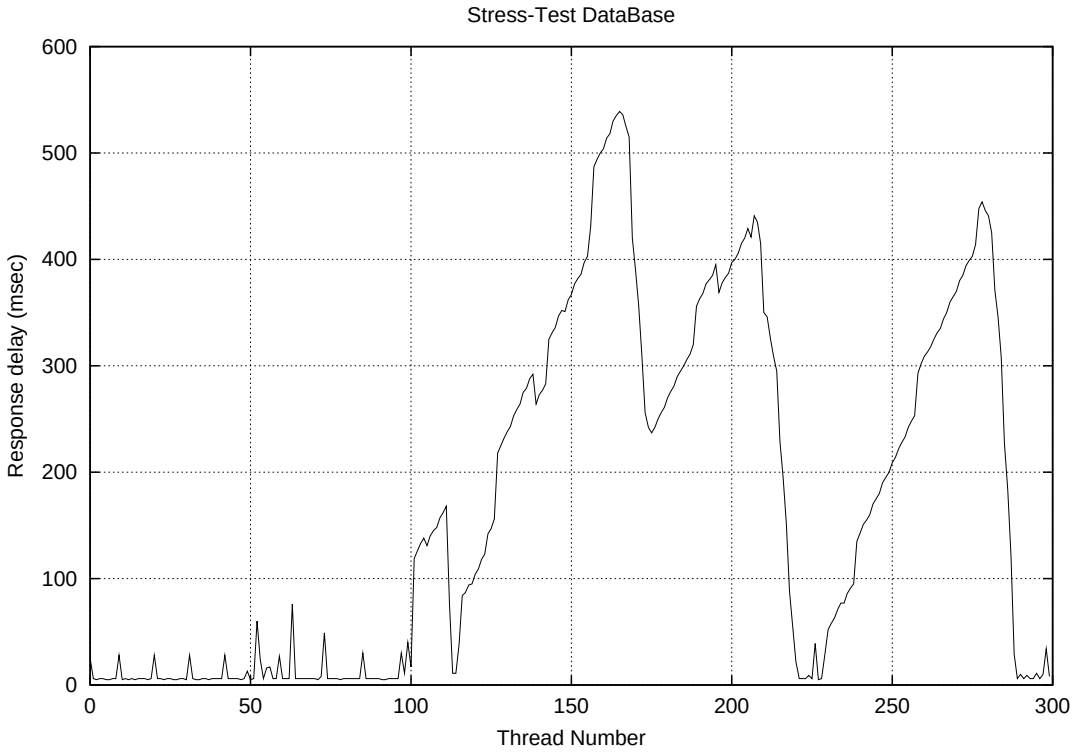

Figure 7.9: Number of threads vs result delay

7.3 Typical User emulation

This test is special, because it was mainly used to test the functionality of the program during development.
However, the test can also be used to give an impression of the behaviour of the auction system as a whole. Initially this task was left to be tackled with the help of my fellow-classmates (hence there would be no need for 'artificial users').
Unfortunately for this purpose, a fully functional / full featured and sufficiently bug-free program was required, and that was only achieved at a later stage, when everybody was too busy and stressed with their own projects.

The test emulates a typical user that it general logs in to the system, browses the available auctions, bids for them, but also create new ones.
Every 'user' is emulated by a separated thread. But let's see in detail what's going on...
First, the benchmarking program retrieves the public key from the server, encrypts the password and logs in to the system with using a dummy account.

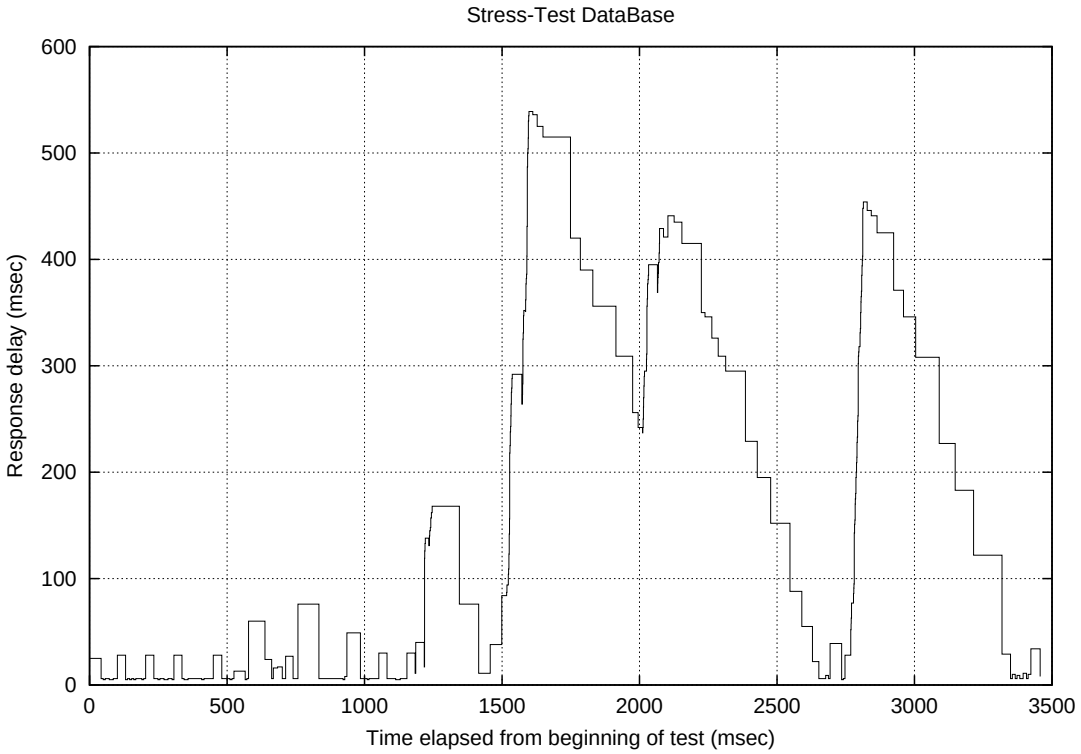

Figure 7.10: Elapsed time vs result delay

Then it gets all the available auction categories (like in the normal client, although here no GUI is available, so no tree will be drawn).
Then we enter a finite loop, that is repeated up to 10 times (the exact number is randomly chosen each time); this loop stands for the number of 'sessions' that this user will have with the system, and includes the following actions:

- the thread waits for six seconds, imitating the time a normal user will take to browse the auction categories and select the one he/she is interested in. Then a selection is made; of course the test system chooses one in random.
- after waiting for some time (that corresponds to the time taken by the user to browse the available auction titles), the thread makes a random selection of an auction that is listed under the category that was chosen in the previous step.

From this point the behaviour of each user (hence thread as well) is differentiated. For that reason i used some percentages that were based on completely subjective criteria. So, i decided that:

- Only 15% of the threads will attempt to create a new auction. In that case the appropriate remote methods are called that create a dummy auction for testing purposes only.
- The rest 85% of the users will not attempt to create new auctions, but will keep browsing them.
- From the browsing users, i assume that 20% will find a good bargain and bid for an item. In that case, the bidding method is called on the currently selected (dummy) item and its price is raised by a fixed amount. The remaining 80% will keep on browsing.

This sequence is repeated several times for each thread, so if we create a sufficient amount of threads, we can emulate a relatively busy system, where several different actions take place at the same time. We can even invoke the benchmarking program from different locations to include a (non-null) network latency.

I used that test with both Windows and Linux platform, and i found that the Linux-MySQL combination is the fastest from all other alternatives; if however, Windows environment has to be used, again MySQL performed better than MS-SQL. Notice though, that we cannot safely draw any general conclusions about the performance of each product, as the results are heavily dependant on the specific hardware & software that was used during the development phase of the program.

Chapter 8

Evaluation of the Project

I have mentioned before that the aim of my project was not to design a program that would compete (in terms of functionality and number of features available) with the commercial packages that empower most of the online auction websites. Still, I believe that the final result is sufficiently featured and extensible and therefore could easily be used in real life environments.

Detailed criticism about the decisions that were made during the development phase of the program and their impact on its flexibility and performance, was given in the appropriate chapters.
By now the reader has probably acquired a good understanding of all the interactions that take place in the background, between the individual parts of the program. Therefore I believe, that this is the right place to briefly mention some ideas that I have in mind (regarding the program) that would either add some new features (hence enhance its usability) or improve some that already exist.

Identity check:
One can easily understand how important it is for the bidding process, that the personal data the system has for the registered users, are valid and true.
For that reason, it would be useful to have some sort of identity check during the registration phase of the program. This would enhance the trust among users of the auction system. The check could have the form of credit card number verification. Since the appropriate algorithm is not available publicly, the number that is given by the user could be send electronically to an appropriate organisation for verification purposes. This should be fairly easy to arrange in a commercial environment.

Two-phase registration:
Following the same idea, a two-phase registration process could be implemented, to make sure that the provided email address is valid and belongs to the user who

attempts to register: The first phase would ask for the user's details (as normal), but instead of registering the user, it would generate a random verification code. For that purpose, a hash code of the concatenation of the provided strings could be a natural and good choice. This code is then sent to the email address provided by the user.
Having received the verification code, the user is then expected to fill in a second form that contains the Username and the appropriate code. Provided the user gave his/her true email, this second phase is just a trivial task of copying and pasting.
This is conceptually fairly straight forward to implement, but was considered too sophisticated for the needs of this program.

Feedback forms:
Another very useful feature to have in the program would be to include some sort of feedback form. Upon the expiration of an auction (provided there was a winner), and after the actual exchange of goods has been completed, both parties involved (seller, auction winner) would have to fill in a feedback form that briefly described the experience and rated the response of the other party..
This rating would follow the user in every transaction, and would be available to everyone. Detailed comments related to each rating could be also displayed upon user request.
By this, we manage to minimise any cases of fraud, where malicious users take advantage of the anonymity the Internet provides, and get richer by manipulating innocent individuals.
This feature is present in some web-based auction sites, and is highly appreciated by the users, but would require a lot of changes to be implemented in my application[1].

Working hyperlinks:
There are also some minor things that could be done to enhance the usability of the program as well.
For instance, it would be nice for the user to be able to include actual working hyperlinks in the window that displays the detailed description of an item[2].
This way, all necessary information could be viewed from within the auction browsing program.
However, I still believe that independent web-browser applications (such as Netscape, or Internet Explorer) are better suited for such task and have better support for the html language than Java does.

[1]For example, the application server will have to keep a separate mailbox for each user; JavaMail provides this functionality, but the resulting program will be far more complicated...

[2]Popular links that one might like to include would be ie. the manufacturer of the item being auctioned, a web-site that has a detailed review of it, or an online store with a sample high-street price

8.1 Conclusion

This project has been a good reason for me to get involved in the e-commerce world, and learn details on how online auctions are being made.
In addition, the task of implementing the actual auction system, gave me a good understanding of:

- the client-server programming techniques (the advantages and the difficulties)
- the Java language in conjunction with CORBA,
- the JavaMail extention,
- encryption techniques, the various algorithms and their uses,
- and of course, I acquired some experience on how to setup, create and interact with databases.

Because of the diversity of the fields that were required to complete this task, the whole project has been an exciting (and quite demanding I might add) experience that kept me busy for more than two months.
According to my beliefs, the novel approach that I considered for the online auction paradigm lived up to my expectations and proved to perform sufficiently well even under heavy load. Java language seems now to be mature enough to be used in real-life, distributed applications with great success.

Chapter 9

Software used:

1. Sun's JDK1.2.2:
   The most fundamental component of the project was the Java language itself, and for that purpose, JDK version 1.2.2 from Sun was used. Both Windows and Linux versions of the JDK were used throughout the writing of the program. Although the final program was tested against JDK1.3 (the latest version available for Windows at the time of writing) and found to be fully compatible, that version was not used for the development phase, because it was still only available as a Beta release for the Linux platform.

2. JacORB1.1:
   JacORB is an object request broker (ORB) written entirely in Java. It is a fully multi-threaded CORBA 2.0 Compliant ORB, with good performance, and several features/utilities. An added advantage is that this piece of software is freely available (published under the GNU Public Library Licence). Version 1.1 (that was used for the project) lacked some features that were found in most of the commercial ORBs and could be useful for the program. Although some of these features were advertised to be present in the next version of the JacORB, this was not released by the time of writing of this document. Nonetheless, other approaches have been taken during the development of the program to overcome this shortcoming.

3. Sun's JCE1.2.1Beta:
   This is the beta version of Java Cryptography Extension package from Sun, which adds cryptographic facilities to the Java language. Fortunately, this version was released in time to be used for the encryption purposes of the program and was the first release ever of the JCE, to be able to be used outside the USA/CANADA, although with reduced strength.

4. eSec's OpenJCE provider:
   The OpenJCE Project from eSec (formerly known as ABA) is a 'clean room'

implementation of the Java Cryptography Extension API, and the underlying cryptographic algorithms. Unlike the equivalent package distributed by SUN, it was developed in Australia (hence, outside USA/Canada) and had no distribution restrictions from the beginning of its release. In addition, the ABA security provider supported the RSA asymmetric public key encryption algorithm that was used in the auction program.

5. Sun's JavaMail 1.1.3:
   Although a new major release of JavaMail is about to be released during the time of the writing of this thesis, version 1.1.3 was used with success in the auction program for the purpose of sending e-mail notifications. This is a standard extension package freely available from SUN, and is written entirely in Java.

6. MySQL 3.22.32 & 3.23.22(beta):
   This is the Database system developed by TCX. It is a high performance, multi-threaded database written in C++, and ported to multiple platforms. Linux and Windows versions were used for the purposes of the program. MySQL v3.22.32 was the latest stable release available during the time of the development phase of the program. Although that version was not Public Domain software, all the releases after v3.23.19(beta) onwards were published under the GPL Licence, hence available for free for non commercial use. Under the suggestion of the TCX developers, I also tried version 3.23.22, which proved to be more stable in most cases. A new major release (3.24) is expected later this year.

7. MSSQL Server 7.0:
   This is the well-known database system from Microsoft. It is definitely more robust and has more features and graphical utilities, although is more complex to use and setup than MySQL (not to mention a *lot* more expensive and demanding in CPU and system resources). This database was chosen as an alternative to MySQL and I was tempted to perform simple benchmarking tests to measure its performance, and compare it to MySQL's.

8. Borland/Imprise JBuilder 3.5 Foundation:
   JBuilder 3.5 is an award-winning visual development tool for building Pure Java applications. It is itself written in Java, so it is available for several platforms (Windows and Linux versions were used in the project). Furthermore, following the philosophy of my all-free implementation, 'Foundation' Edition is available for download from Borland's homepage with no cost. This program was used as an advanced text editor for all the Java related files, with useful features like syntax highlighting and automatic method/parameter lookup and completion.

Appendix A

The auction IDL file:

This is the actual IDL file that was used in the auction application. Please note that some remote methods that can be found at the end, were only included for benchmarking purposes and do not affect the performance/functionality of the rest of the program.

```
module Auction
{
  typedef string<10> TimeDate;

  typedef sequence<octet> blob;

  typedef sequence<octet> encodedPublicKey;

  typedef sequence<octet> encodedStream;

  struct PersonalDetail
  {
    unsigned long  UID;
    string<10> UName;
    string<10> PWord;
    string FName;
    string LName;
    string Address;
    string<7> PCode;
    string Country;
    unsigned long Tel;
    unsigned long Fax;
    string Email;
    string Public_Key;
    string Private_key;
```

```
};

struct Item
{
  string<10> Seller;
  string Title;
  string Description;
  string<10> HBidder;
  float HBid;  // Starting Price as well
  float Inc;
  float Tip;
  blob Img;
  float Reserve;  //0:NoReserve
  float x,y,z,w;
  float pp;
  unsigned long IID;
  unsigned long CID;
  long long start_time;
  long long end_time;
};

struct Brief_Description
{
  unsigned long IID;  //was AID
  boolean has_image;
  boolean is_new;
  //boolean is_active;
  string Title;
  float HBid;
  unsigned long NBids;
  string time_left;
};

struct Category_Leaf
{
  unsigned long CID;
  string Name;
  unsigned long ParentID;
};

struct test_result
{
  unsigned long difference;
  long long started_at;
};
```

```
typedef sequence<Brief_Description> Results;
typedef sequence<Category_Leaf> Category_Leaf_Seq;

interface AuctionCallBack
{
  // Refresh details on client side.
  // We pass the IID, so the _client_ decides whether it needs
  // to actually refresh the details (if the user is looking at that item)
  // or NOT! (if the user is looking at another item...)
  void refreshDetails(in unsigned long IID);

  // ...The same here...
  // Update Items only if CID is the same with the one the User sees :-)
  void refreshItems(in unsigned long CID);

  // Update WatchList only if the current user changed sth in
  // his/her watchlist.
  void refreshWatchList(in unsigned long UID);
  // this introduces a big overhead (only one client will be
  // interested in the changes, and all clients will be notified)
  // Unlike the other calls where more than one clients will be interested
  // (hopefully). So i might not use this method ...

};

interface Coordinator
{
  exception NoSuchUser {
  //string<10> UName;
  };

  exception WrongIID {
    //unsigned long IID;
  };

  exception WrongUID {
    //unsigned long UID;
  };

  exception WrongCID {
    //unsigned long CID;
  };

  exception ErrorInEmailField {};
```

```
exception UserExists {};

// Send:(IID)
// Get: Auction Details
Item getInfoOnItem(in unsigned long IID);

// Send:(Item),(UID)
// Get: result (true:OK, false:sth went wrong)
boolean submitItem(in Item tempItem,in unsigned long UID);

// Send:(UID)
Results getWatchList(in unsigned long UID) raises (WrongUID);

// Send:(UID),(IID)
// Get:true=OK,false=sth wrong
boolean addToWatchList(in unsigned long UID,in unsigned long IID)
raises (WrongUID,WrongIID);

// Send:(UID),(IID)
// Get:true=OK,false=sth wrong
boolean removeFromWatchList(in unsigned long UID,in unsigned long IID)
raises (WrongUID,WrongIID);

//Send:Category ID
// Get:an array of a brief description of items in that category
Results getItems(in unsigned long CID) raises (WrongCID);

// Get: an array of objects of type Category Leaf
Category_Leaf_Seq buildTree();

// Get: long value, that will be converted into a timestamp
long long get_servertime();

// Send:(UName)
// Get:PublicKey
encodedPublicKey getPublicKey(in string<10> UName) raises (NoSuchUser);

// Send:(UName),(encoded PWord)
// Get:UID (!=-1 if correct, -1 if anonymous/wrong)
unsigned long secureLogin(in string<10> UName,in encodedStream encPWord)
raises (NoSuchUser);

// After Encoding password given
// from the user, it is encoded with the
```

```
// Public Key, and ONLY THEN passed to
// the server for authentication!

// Send:(UName),(unencoded PWord)
// Get:UID (!=-1 if correct, -1 if anonymous/wrong)
unsigned long simpleLogin(in string<10> UName,in string<10> PWord)
raises (NoSuchUser);

// Send:(UID),(AID),(bid amount) (we bid in a specific Auction not Item!)
// Get:true=OK,false=sth wrong
boolean bid(in unsigned long UID,in unsigned long IID,in float Bid);
// Exception:User Not Found, Auction Not Found, Ammount not accepted

// Send:(UName)
// Get:-1=NoSuchUser/Anonymous , !-1=UID
unsigned long getUID(in string<10> UName) raises (NoSuchUser);

// Send:(UID)
// Get:UName
string<10> getUName(in unsigned long UID) raises (WrongUID);

// Send:(UID)
// Get:Personal Details ... Used after auction is closed
PersonalDetail getPersonalDetail(in unsigned long UID) raises (WrongUID);

// Send:Personal Details (when registering)
// Get:true=OK,false=sth wrong
boolean setPersonalDetail(in PersonalDetail PDetail) raises (UserExists);

// Send:Personal Details (when registering)
// Get:true=OK,false=sth wrong
//boolean setPersonalDetailSecure(in any SecurePDetail)
boolean setPersonalDetailSecure(in encodedStream SecurePDetail) raises (UserExists);

// In case that the OWNER of an auction wants to close it early...
boolean closeAuction(in unsigned long IID);

// The following sends the (reference to the Callback Obj) to the Server
// So objects of the CallBackClient class can be created on the Server side
// and methods on these objects can also be called...
void registerClientWithServer(in AuctionCallBack AuctionCallBackRef);

void disconnectClientFromServer(in AuctionCallBack AuctionCallBackRef);

//WARNING! Completely insecure and dangerous method
```

```
    //It is used ONLY in the Benchmarking program for convenience...
    //Remove it and recompile the idl if in commercial environment
    void clear_tables();

    //The following methods are only for benchmarking reasons
    //In both cases i will try to keep the network communication
    //as little as possible.... (no arguments sent, an integer is returned)

    //Here, i don't care about how 'difficult' the query to the DB is
    //I am just interested in having LOTS of remote requests....
    test_result stresstest_Server();

    //Here, i have LOTS of requests, but i am stressing the DB as well
    test_result stresstest_DB();
  };
};
```

Appendix B

Benchmark numeric results and screen output

B.1 Network/ORB test

Screen output:

```
Registering Security provider.
Creating file to keep results.
Running Network latency / Database test
Getting public key:                     [426ms]
Converting to RSAPublicKey:             [17ms]
Getting cipher Instance:                [28ms]
Initializing Cipher for Encryption:     [15ms]

--Insecure registration tests--
Insecure registration, first time:      [188ms]
Insecure registration, second time:     [164ms]
Insecure registration x50:              [179ms] each

--Insecure login tests--
Insecure login, first time:             [162ms]
Insecure login, second time:            [133ms]
Insecure login x50:                     [179ms] each

--Secure registration tests--
Getting public key:                     [161ms]
Converting to RSAPublicKey:             [1ms]
Getting cipher Instance:                [1ms]
Initializing Cipher for Encryption:     [4ms]
```

```
Encoding personal details, first time:  [5101ms]
Secure registration, first time:        [642ms]
-
Getting public key:                     [184ms]
Converting to RSAPublicKey:             [0ms]
Getting cipher Instance:                [1ms]
Initializing Cipher for Encryption:     [1ms]
Encoding personal details, second time: [4ms]
Secure registration, second time:       [241ms]
-
Security related stuff x50:             [136ms] each
Encoding personal details x50:          [2ms] each
Secure registration x50:                [196ms] each

--Secure login tests--
Security related stuff:                 [129ms]
Encoding pword, first time:             [0ms]
Secure login, first time:               [132ms]
-
Security related stuff:                 [120ms]
Encoding pword, second time:            [1ms]
Secure login, second time:              [124ms]
-
Security related stuff x50:             [126ms] each
Encoding pword x50:                     [0ms] each
Secure login x50:                       [137ms] each
Closed results file
Now, removing ALL Users/Owners/Auctions/Items from database!

Total Running Time:58344
```

Result file:

I am including the result file that was produced for local computer (Celeron 400MHz) running Linux/MySQL. This is the output after being processed by the 'averaging script' (for the source code look in Appendix B.4).

```
51.40 19.40 38.00 17.60 23.60 11.80 4.40 5.40 6.80 10.80 3.20 0.60 1.00 5.40 5164.20 72.80 2.00 0.00 \
      0.40 0.00 5.00 61.60 2.60 2.60 62.40 2.80 0.40 15.60 2.00 1.00 14.20 2.80 0.60 20.00 1.00
44.60 19.20 38.40 18.00 9.60 5.40 7.00 5.20 6.00 5.00 7.60 0.60 1.60 5.00 5193.40 57.60 1.60 0.00  \
      0.40 0.20 4.80 57.60 2.20 2.80 60.80 2.40 0.80 16.40 2.60 0.60 14.40 6.00 0.20 15.20 2.00
46.60 19.60 38.00 17.60 9.40 5.60 4.00 5.60 6.40 5.40 3.00 0.60 1.80 5.40 5188.00 55.60 2.00 0.00  \
      0.80 0.40 4.60 62.00 4.40 2.60 63.80 4.80 0.80 21.40 12.80 0.80 19.40 10.20 0.20 16.60 3.00
44.60 19.20 42.00 18.00 9.60 5.80 4.00 5.80 6.40 5.00 3.20 1.00 1.00 5.20 5166.80 56.80 1.40 0.00  \
      0.20 0.80 4.40 57.00 2.00 2.40 65.20 5.80 0.60 22.00 8.00 1.00 16.60 11.00 0.20 20.80 4.00
44.40 19.20 38.20 18.00 9.80 5.40 5.60 5.60 6.60 8.00 3.00 0.80 1.20 5.40 5222.80 57.00 2.00 0.00  \
      0.00 0.60 4.80 57.00 7.20 2.20 77.60 34.80 0.80 25.20 2.80 0.60 16.40 6.80 0.00 18.80 5.00
```

```
44.40 19.00 39.60 18.40 9.60 11.40 5.60 6.00 6.80 7.00 3.00 0.60 1.20 5.20 5179.40 61.40 1.40 0.00 \
      0.40 0.60 4.40 62.80 6.00 2.00 71.80 17.60 0.40 23.60 7.20 0.40 16.80 6.40 0.00 17.60 6.00
44.40 19.80 38.20 18.00 9.80 5.80 5.40 5.60 14.00 7.80 3.20 0.60 1.20 5.40 5162.20 56.20 1.60 0.00 \
      0.00 0.80 4.60 58.00 7.20 2.40 75.00 19.20 0.60 19.40 5.60 0.40 15.60 8.60 0.00 17.00 7.00
43.40 19.20 37.00 17.60 17.80 5.80 4.80 6.40 9.60 8.00 3.00 1.00 1.00 5.20 5189.80 57.00 1.80 0.20 \
      0.20 0.20 4.80 55.20 9.00 2.20 72.00 23.40 1.00 16.80 4.80 1.00 16.80 8.40 0.20 16.80 8.00
43.60 18.80 36.60 17.80 9.60 5.40 4.00 5.80 6.80 6.00 2.80 0.80 1.20 5.40 5193.80 85.80 2.00 0.00  \
      0.00 1.00 4.20 54.80 12.20 2.80 72.60 8.80 0.60 18.80 7.60 1.00 24.20 9.60 0.40 18.80 9.00
43.00 18.80 36.20 17.80 9.80 5.80 5.20 6.00 7.60 8.00 3.00 0.80 1.20 5.60 5168.60 56.00 2.00 0.00  \
      0.00 0.80 4.60 58.00 7.60 2.00 79.20 11.80 0.60 19.00 10.40 0.80 24.80 6.00 0.00 15.80 10.00
43.40 19.00 36.40 18.00 9.20 5.40 5.00 6.00 7.00 9.00 2.80 0.40 1.60 5.40 5168.60 54.40 1.80 0.00  \
      0.20 0.80 4.20 55.00 11.20 2.00 82.20 15.80 0.80 16.40 17.80 0.80 15.60 7.00 0.00 16.40 20.00
43.60 18.60 36.60 17.60 9.80 5.80 5.00 6.20 7.40 7.00 3.00 0.60 1.20 5.20 5178.80 55.60 1.80 0.00  \
      0.00 0.80 4.40 56.40 12.40 2.00 85.80 6.00 0.20 19.20 3.40 0.40 19.80 8.20 0.00 16.00 30.00
43.20 18.80 36.00 18.00 9.80 5.80 5.20 6.00 21.00 6.40 2.80 1.00 1.20 5.00 5204.40 55.80 1.80 0.00 \
      0.20 0.60 4.80 54.20 13.60 2.00 83.80 8.00 0.40 15.60 13.40 0.20 16.60 7.80 0.00 16.00 40.00
43.20 18.80 36.40 17.80 9.60 5.60 4.60 6.00 7.20 6.60 3.00 0.80 1.20 5.40 5190.20 54.20 1.60 0.20  \
      0.60 0.40 5.00 54.20 11.00 2.00 87.20 25.00 0.60 20.80 3.60 0.60 15.80 7.60 0.00 16.20 50.00
```

B.2 Stress tests

This is only a small part of the actual stressDB file; the remaining data showed no special interest, and therefore was ommited from this part, as it would take up precious space.

```
0 Thread-4 25
42 Thread-5 6
50 Thread-6 5
57 Thread-7 6
64 Thread-8 6
72 Thread-9 5
79 Thread-10 5
86 Thread-11 6
93 Thread-12 6
101 Thread-13 28
131 Thread-14 5
138 Thread-15 6
146 Thread-16 5
153 Thread-17 6
161 Thread-18 5
168 Thread-19 6
175 Thread-20 6
182 Thread-21 6
190 Thread-22 5
197 Thread-23 6
204 Thread-24 28
[...]
```

StressSRV has the exact same syntax, but different data. It will therefore not be included here.

B.3 Typical User simulation

The output (as we get it from the benchmarking program) is shown below:

```
968623892350 Thread-7 Category no:64/65  Name:Other
968623892356 Thread-6 Category no:14/65  Name:Magazines
968623892357 Thread-8 Category no:2/65  Name:Arts & Entertainment
968623892367 Thread-9 Category no:56/65  Name:Recreational Vehicles
968623892370 Thread-10 Category no:12/65  Name:Art
968623892373 Thread-11 Category no:58/65  Name:Sport Utility Vehicles
968623892376 Thread-12 Category no:8/65  Name:Sport & Recreation
968623892379 Thread-13 Category no:29/65  Name:General Office Supplies
968623892384 Thread-14 Category no:23/65  Name:Business Books
968623897647 Thread-5 Category no:58/65  Name:Sport Utility Vehicles
968623898508 Thread-8 Category no:38/65  Name:Palmtops & PDAs
968623898565 Thread-6 Category no:0/65  Name:Categories
968623898567 Thread-10 Category no:28/65  Name:Furniture
968623898569 Thread-12 Category no:57/65  Name:Snowmobiles
968623898820 Thread-7 Created new auction: [249ms]
968623898821 Thread-7 Category no:26/65  Name:Consulting
968623898822 Thread-9 Created new auction: [251ms]
968623898870 Thread-11 Created new auction: [299ms]
968623898870 Thread-11 Category no:48/65  Name:Other
968623898884 Thread-13 Created new auction: [313ms]
968623898900 Thread-14 Created new auction: [329ms]
968623898901 Thread-14 Category no:31/65  Name:Writing Instruments
968623903659 Thread-5 Category no:16/65  Name:Movies and Film
968623904527 Thread-8 Category no:16/65  Name:Movies and Film
968623904836 Thread-6 Created new auction: [35ms]
968623904837 Thread-10 Created new auction: [37ms]
968623904838 Thread-10 Category no:32/65  Name:Other
968623905040 Thread-7 Contains:5 auctions, chosen no.4 [1ms]
968623905050 Thread-11 Contains:5 auctions, chosen no.1 [11ms]
968623905051 Thread-14 Contains:5 auctions, chosen no.2 [13ms]
968623909731 Thread-5 Created new auction: [44ms]
968623909732 Thread-5 Category no:43/65  Name:GPS
968623910606 Thread-8 Created new auction: [19ms]
968623910607 Thread-8 Category no:52/65  Name:Cars
968623910940 Thread-10 Created new auction: [50ms]
```

```
968623910941 Thread-10 Category no:39/65  Name:Amateur Radio
968623913141 Thread-7 Bidded for:5 [82ms]
968623913142 Thread-7 Category no:53/65  Name:Cycling
968623913216 Thread-11 Bidded for:5 [157ms]
968623913217 Thread-11 Category no:10/65  Name:Travel and Transportation
968623913301 Thread-14 Bidded for:5 [208ms]
968623913302 Thread-14 Category no:22/65  Name:Briefcases and Bags
968623915806 Thread-5 Created new auction: [19ms]
968623915807 Thread-5 Category no:35/65  Name:Software
968623916686 Thread-8 Created new auction: [20ms]
968623917016 Thread-10 Created new auction: [19ms]
968623919226 Thread-7 Contains:10 auctions, chosen no.4 [9ms]
968623919327 Thread-11 Created new auction: [20ms]
968623919379 Thread-14 Contains:10 auctions, chosen no.2 [12ms]
968623921943 Thread-5 Created new auction: [46ms]
968623921944 Thread-5 Category no:18/65  Name:Services
968623927248 Thread-7 Bidded for:10 [11ms]
968623927249 Thread-7 Category no:28/65  Name:Furniture
968623927387 Thread-14 Category no:41/65  Name:Calculators
968623928047 Thread-5 Created new auction: [20ms]
968623928048 Thread-5 Category no:32/65  Name:Other
968623933376 Thread-7 Created new auction: [20ms]
968623933486 Thread-14 Created new auction: [20ms]
968623934176 Thread-5 Created new auction: [19ms]
```

Output was sorted by time (the first column).

B.4 Averaging script

This script is my first attempt with gawk. The algorithm that was used might not have been optimal, but the result is satisfactory.

```
awk '{
if(max_nf<NF) max_nf=NF
max_NR=NR
rr[$NF]=rr[$NF] " " NR
for(x=1;x<=NF;x++) data[x,NR]=$x
#rr[$NF]=rr[$NF] " " NR
}
END {
  for(x=1;x<=max_nf;x++)
  {
```

```
      orc=1
      for(z in rr)
      {
        split(rr[z],i)
        count=1
        for (rows in i)
        {
          output[x,orc]+=data[x,i[rows]]
          count++
        }
        output[x,orc]=output[x,orc]/(count-1)
        orc++
      }
    }
    orc--
#   for(y=1;y<=orc;y++)
#   {
#     for(x=1;x<=max_nf;x++) printf "%2.2f ",output[x,y]
#     printf "\n"
#   }
    for(y=1;y<=orc;y++)
    {
      for(x=1;x<=max_nf;x++) printf "%2.2f ",output[x,y]
      printf "\n"
    }
}' $1 |sort +34 -n
```

Bibliography

[1] Robert Orfali, Dan Harkey.
John Wiley & Sons, 1998
Client/server programming with Java and CORBA

[2] Robert Orfali, Dan Harkey, Jeri Edwards.
John Wiley & Sons, 1996
Instant CORBA

[3] Andreas Vogel, Keith Duddy.
Java Programming with CORBA, second edition.

[4] OMG's Homepage
Various documents (mostly formal specifications related to CORBA)
http://www.omg.org

[5] Steve Vinoski (Iona Technologies, Inc.)
CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments.

[6] Andrew T.Campbell, Geoff Coulson, Michael Kounavis
IT Pro September/October 1999
Managing Complexity: Middleware explained

[7] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna
IEEE transactions on software engineering, Vol 24, 1998
Understanding Code Mobility

[8] Andrew E. Wade (Objectivity Inc.)
Distributed Client Server Objectbases
(for Object Magazine)

[9] Gopalan Suresh Raj
A Detailed Comparison of CORBA, DCOM and Java/RMI
http://www.execpc.com/g̃opalan/misc/compare.html

[10] JacORB Homepage
http://www.inf.fu-berlin.de/~brose/jacorb/
(Performance comparison)
http://www.inf.fu-berlin.de/~brose/jacorb/performance/index.html

[11] Distributed Object Group / JavaORB Homepage
http://www.multimania.com/dogweb/

[12] JavaTM 2 Platform, Standard Edition, v1.2.2 API Specification
http://java.sun.com/products/jdk/1.2/docs/api/index.html

[13] The Java Tutorial, A practical guide for programmers
http://java.sun.com/docs/books/tutorial/index.html

[14] Java Cryptography Edition Api Specification
(local copy, available only via registration)

[15] OpenJCE Homepage
http://www.openjce.org
JCA/JCE Api Overview:
http://www.openjce.org/docs/jce_api_overview.html

[16] JavaMail extention to Java language
API Specification and examples
http://java.sun.com/products/javamail/index.html

[17] Getting Started with the JDBC Api
http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html

[18] JDBC a Java SQL Api
http://java.sun.com/j2se/1.3/docs/guide/jdbc/spec/jdbcspec.frame.html

[19] JDBC Api specification
http://java.sun.com/j2se/1.3/docs/api/index.html

[20] MySQL Homepage
http://www.mysql.com/
(latest manual)
http://www.mysql.com/documentation/mysql/commented/

[21] Randy Jay Yanger, George Rees & Tim King
O'REILLY Series, July 1999
MySQL & mSQL

[22] Optimising MySQL (oscon2000)
http://www.mysql.com/news/article-26.html

[23] Shubik,M (1983) Auctions, Bidding, and Markets: An Historical Sketch. In R.Engelbrecht-Wiggans, M.Shubik and J. Stark. New York: New York University Press

[24] Paul Klemperer (1999), Nuffield College, Oxford University. Auction Theory: A guide to the Literature. Forthcoming Journal of Economic Surveys.

[25] Internet Auctions: A guide for Buyers and Sellers
USA Federal Trade Commission
http://www.ftc.gov/bcp/conline/pubs/online/auctions.htm

[26] Online auction system
Mark Greer, August - December 1998
http://www4.ncsu.edu/eos/users/r/reeves/rtcomm/DistRT/auction.html

[27] Online auctions
`http://www2000.ogsm.vanderbilt.edu/Student.Projects/consumer.online.auctions/Auctions(htm).htm`

[28] Online auctions: Glossary of terms
http://www.auctionposter.com/help/help_glossary.htm

[29] Cray supercomputer finds afterlife on eBay
Troy Wolverton, CNET News.com, 7Sep 2000
http://news.cnet.com/news/0-1007-200-2718503.html