

UNIVERSITY OF
NEWCASTLE



University of Newcastle upon Tyne

School of Computing Science

Scalable Internet Auctions

Ph.D. Thesis

By:

Mohammad-Reza Khayyambashi

In Partial Fulfilment of the Requirements

for Degree of

Doctor of Philosophy

March 2006

NEWCASTLE UNIVERSITY LIBRARY

204 26851 7

Thesis L8220

**BLANK PAGE
IN
ORIGINAL**

Dedicated to :

“ My father and my mother ”

Acknowledgements

I would like to express my sincere gratitude to those who have contributed in various ways to the completion of this thesis.

First and foremost, I greatly thank my supervisor, Professor Santosh K. Shrivastava. His guidance and academic contributions throughout this work have been invaluable. The enthusiasm and encouragement Professor Shrivastava consistently showed throughout my PhD studies have been essential for the completion of the thesis.

Many thanks go to Dr. Graham Morgan for teaching me about NewTop and related materials as well as his valuable comments. My thanks are also due to Dr. Paul Ezhilchelvan for the discussions relating to Internet Auction, which are included in this thesis.

I would also like to thank the members of Distributed Systems group, head of school, Dr. John Lloyd, and all the staff members of the school particularly Shierly Craig, the librarian who was a great help specially with the references.

Furthermore, I would like to express my deep heartfelt appreciation to my dear wife, Azam, for her sacrifice, courage and support especially on those lonely and hardship periods; to my lovely daughters, Maedeh and Zahra, for their understanding of the nature of my work, for being supportive and all the joy and sorrow that they shared with me.

I would take this opportunity to thank my parents, sister and brothers for their continued support and encouragement during my life.

Finally, the financial support in the form of a scholarship from the Iranian Ministry of Science, Research and Technology and Isfahan University is gratefully acknowledged.

Abstract

Current Internet based auction services rely, in general, on a centralised auction server; applications with large and geographically dispersed bidder client bases are thus supported in a centralised manner. Such an approach is fundamentally restrictive as too many users can overload the server, making the whole auction process unresponsive. Further, such an architecture can be vulnerable to server's failures, if not equipped with sufficient redundancy. In addition, bidders who are closer to the server are likely to have relatively faster access to the server than remote bidders, thereby gaining an unfair advantage.

To overcome these shortcomings, this thesis investigates ways of enabling widely distributed, arbitrarily large number of auction servers to cooperate in conducting an auction. Allowing a bidder to register with any one of the auction servers and place bids there, coupled with periodic exchange of auction information between servers forms the basis of the solution investigated to achieve scalability, responsiveness and fairness. Scalability and responsiveness are achieved since the total load is shared amongst many bidder servers; fairness is achieved since bidders are able to register with their local servers.

The thesis presents the design and implementation of an hierarchically structured distributed Internet auction system. Protocols for inter-server cooperation are presented. Each server may be replicated locally to mask node failures. Performance evaluations of centralised and distributed configurations are performed to show the advantages of the distributed configuration over the centralised one.

Table of Contents

Acknowledgements	iv
Abstract	v
List of Tables	ix
1 Introduction	
1.1 Overview	1
1.2 Thesis Structure	2
2 Background and Related work	
2.1 Introduction	6
2.2 Type of Auction	9
2.3 Characteristics of Different type of Auction	13
2.4 Complete Auction Process	14
2.5 Significant of Auctions.....	15
2.6 Problem with Auctions	16
2.7 Internet Auctions	17
2.7.1 Existing Internet Auctions	19
2.7.2 Advantage of Internet Auctions	20
2.7.3 Problem with Internet Auctions	21
2.7.4 Other consideration relating to Internet Auctions	23
2.8 Auction Requirements	24
2.9 Related work	25
2.10 Conclusions.....	28
3 Middleware Environments and Group Communication	
3.1 Middleware	30
3.1.1 Properties of Middleware	31
3.1.2 The Proxy/Stub Method.....	32
3.1.3 Distributed Objects	32

3.2	Object – Oriented Middleware Technologies.....	33
3.2.1	Java – RMI	33
3.2.2	DCOM	34
3.2.3	CORBA	34
3.3	OMG and CORBA.....	36
3.3.1	ORB	36
3.3.2	CORBA Interface Architecture.....	37
3.3.3	Object Services.....	39
3.4	Fault Tolerance and Reliability Issues.....	40
3.4.1	Atomic Transaction.....	40
3.5	Group Communication	42
3.5.1	Properties of a Group Communication Service.....	43
3.5.2	The Multicast Mechanism	43
3.5.3	Message Ordering	44
3.5.4	Group Membership.....	47
3.5.5	Highly Available Services	48
3.6	Reliability / Translation – Based Reliability and Comparison.....	52
3.7	Conclusions	54
4	Centralised and Distributed System Architecture for Internet Auction	
4.1	Auction House Architecture	56
4.2	Basic Auction Unit	57
4.2.1	Clients’ Procedure	59
4.2.2	Servers’ Procedure	60
4.2.3	Basic Auction Unit Connection	62
4.3	Pure Hierarchical Architecture	64
4.4	Implementation Framework.....	68
4.4.1	Bidder Servers.....	71
4.4.1.1	Bidder Servers’ Procedure.....	71
4.4.1.2	Forwarding / Disseminating time.....	72

4.4.2	Main Auction Server.....	76
4.4.2.1	Main Auction Server's Procedure.....	77
4.5	Reliability and Fault Tolerance.....	78
4.5.1	Replication – Based Reliability.....	80
4.5.2	Implementation and Reliability Issues.....	82
4.5.2.1	Fault Tolerance Bidder Servers' Procedure.....	87
4.5.2.1.1	Bidder Servers' Procedure.....	88
4.6	Conclusions	90

5 Implementation and Performance of Distributed Auction System

5.1	Implementation of Distributed Auction System.....	91
5.1.1	Auction Object.....	92
5.1.2	Reliability Object.....	92
5.1.3	Distributed Object.....	94
5.1.4	Group Communication Object.....	96
5.2	The NewTop Group Communication Service.....	97
5.2.1	Management Service.....	99
5.2.2	Invocation / Multicast Service.....	100
5.2.3	Group Membership Service.....	102
5.3	Performance Evaluation of Distributed Auction System.....	103
5.3.1	Centralised Server, Non-replicated Auction Object.....	104
5.3.2	Distributed Bidder Servers, Non-replicated Auction object.....	106
5.3.3	Replicated Auction Objects (Centralised and Distributed Approaches).....	108
5.3.4	Conclusions.....	109

6 Conclusions

6.1	Thesis Summary.....	110
6.2	Future Work	112

Bibliography

Bibliography	113
--------------------	-----

List of Tables

Table 2-1	13
Table 5-1	108
Table 5-2	108

Chapter 1

Introduction

1-1 Overview

Advances in information and communication technology have triggered a massive influx of Electronic Commerce transactions, which represent one of the fastest growing business segments that has ever existed, due to its convenience, absence of space and time limitations, limitless choice of goods and services, ability to compare prices and goods between various e-commerce providers, and round the clock online shopping. (It is estimated that 63% of the online population will engage in Electronic Commerce activities by 2006. Furthermore, it is perceived that the total value of Electronic Commerce transactions around the world reached around \$3.8 trillion in 2003, over \$9 trillion in 2005, and around 18% of global sales in 2006) [<http://www.crime-research.org/articles/Wahab1>].

The Internet and World Wide Web have emerged as a valuable networked information source that is increasingly being used for commerce. In recent years a number of auction services have been made available over the Internet (e.g., www.ebay.com, www.antiquorum.com, www.artnet.com, to name a few). A common feature of these services is the considerable amount of time they require to complete the auctioning process. Typically, a user of these services can submit a bid and, only after an amount of time that can range from hours to days, that user knows whether her/his bid has been accepted [Amoroso'03].

The Internet auction incorporates most of the features of the traditional auction, while it also adds features necessary for its new media. Due to the geographically dispersed nature and the sheer number of the auctions relatively anonymous participants, some method of bidder and seller reputation has to be established. The majority of Internet auctions now incorporate some method whereby a bidder or seller's peers can give

him/her a rating and write comments on their behaviour and trustworthiness. Other bidders can then use these rating and comments as a guide to who they are dealing with.

We are concerned with a particular class of Internet-based, server-centered applications whose user domains are typically large, geographically distributed, and perhaps expanding. Examples of such applications are on-line auctions, Internet gaming, etc. On-line auctions are continually expanding into diverse products ranging from second-hand goods to airline tickets and financial products. The well-known Internet auction provider, eBay [<http://www.ebay.com>] has recently entered into the real-estate markets. The size and the nature of the user domain becomes obvious when we observe that eBay runs up to 2 million auctions at any given time, and its systems typically interact simultaneously with millions of Internet based customers from all over the world. Internet games not only are becoming increasingly popular but are such that the more the number of players participates in a game, the more interesting the game becomes for every player. So, in Internet gaming, systems are required to deal with a large number of users whose requests (for example to move or shoot an object) must be processed in an ordered manner and the effect displayed in a timely manner.

Current Internet-based auction services rely, in general, on a centralised auction server; it means the applications with large and geographically dispersed bidder client bases are currently supported in a centralised manner; bidder client requests (over the Internet) to systems located in a central place for processing. Such an approach is fundamentally restrictive as too many users can overload the server, making the whole auction process unresponsive. Internet auction service requires meeting scalability, responsiveness, fairness and data integrity. In the other words, a centralised auction server architectures can not deal adequately with issue of service availability and scalability. Typically, such an architecture can be vulnerable to server's failures, if not equipped with sufficient redundancy. Furthermore, server's overloading may occur, if an arbitrary large number of users concurrently access the service and a customer (a local bidder client) who is close to a central bidder server can have faster bidder server access than a remote bidder client, and thus may have an unfair advantage over the latter.

Therefore the ways of enabling widely distributed arbitrary large number of auction servers to cooperate in conducting an auction have been investigated. Allowing a user to place a bid at any one of the bidder servers is our solution to achieve scalability and responsiveness, since the total load is shared amongst many bidder servers which has been distributed in our auction system.

1-2 Contributions of the thesis

In this thesis we will develop a novel hierarchical architecture for distributed Internet auction system. Hierarchical architecture and Tree-based recursive design approach is shown as a very attractive way to meet Internet auction's scalability and responsiveness requirements. Fairness is preserved using distributed auction servers. Data integrity is attained by atomic interactions between the bidder client and the bidder server and by preserving replicated data mutually consistent.

This thesis studies the types of auctions, existing problems in centralised Internet auctions and architecture of centralised and distributed Internet auctions. It also introduces replication of bidder servers using group communication to achieve the reliability and fault tolerance in centralised and distributed system architecture for Internet auctions. Furthermore, the new algorithms and protocols which have been supported by hierarchical architecture are also presented.

The thesis is structured in six chapters. The details of each chapter will be described in next section.

1-2 Thesis structure

The thesis is organised as following:

Chapter 2: Background and Related Work. This chapter covers the background of Electronic commerce, auction in general and then on Internet auction in particular. The discussion of different types of traditional auctions, their important and the problem they face is followed by a look at existing Internet auction and their

problems and then Internet auction requirement will also be described. The chapter will be ended by related work.

Chapter 3: Middleware Environments and Group Communication. In this chapter description of the Middleware technologies and the mechanism that required to provide replicated servers and group communication service will be covered.

Chapter 4: Centralised and Distributed System Architecture for Internet Auction. This chapter demonstrates auction house architecture, basic auction units and their connection, hierarchical architecture and Tree-based recursive design approach as a well-known method to achieve Internet auction's scalability. This chapter also describes replicated bidder servers to attain reliability and fault tolerance in distributed system architecture for Internet auction followed by discussion of algorithms and protocols for bidder client and bidder servers in both non-fault tolerance and fault tolerance models.

Chapter 5: Implementation and Performance of Distributed Auction System. In this chapter implementation of the distributed system architecture for Internet auction and performance results obtained from experiments are presented. The results compare the performance of replicated and non-replicated auction bidder servers for both centralised and distributed bidder servers approaches.

Chapter 6: Conclusions. The chapter concludes the overall thesis and implementations of its finding. It summarises the performance and briefly discusses the contribution of this thesis and suggests future work.

Chapter 2

Background and Related work

In this chapter information is presented on Electronic Commerce, Auctions in general and then on Internet Auctions in particular. A discussion of different types of traditional auctions, their importance and the problem they face is followed by a look at existing Internet auction and their problems and then internet auction requirement and related work will be discussed.

2-1 Introduction

Electronic Commerce on the Internet has become one of the major issues in computing in the last few years. The development of the World-Wide-Web technology and its related browsers has transferred the idea of commerce, trading, marketing and auction on electronic media into a reality. The vast opportunities presented by Electronic Commerce can be easily gauged by the amount of serious interest shown by the business sector and by researchers in the field of computing.

In the business sector the numbers of Internet Service Providers have increased dramatically, responding to the ever greater number of people wishing to “connect to the net”. The term network computer has been coined to capture the duality of the nature of personal computers today, namely as a desktop computer and as a gateway to the world of the Internet.

As electronic commerce becomes a major activity on the Internet (and other interconnected networks), user will demand other related services to be delivered through and by the Internet [Hardjono'96].

Contemporary with the growth of Electronic Commerce, at the same time, the use of Electronic Marketplace and Electronic Auction also has made its appearance.

Electronic Marketplaces have become increasingly popular alternatives to traditional forms of commerce. This increase in popularity has led many to predict that one effect will be to lower the market price of goods. Buyers in market inter mediated transactions have to bear search costs to obtain information about the prices and product offerings of sellers. High search costs of buyers enable sellers to maintain prices substantially above their marginal costs and results in allocational inefficiencies in market transactions. Electronic market systems can reduce the search costs that buyer must incur to acquire information about seller prices and product offerings, thus enabling buyers to locate suppliers that better match their needs. The lowered search costs allow buyers to look at more product offerings and make it difficult for sellers to sustain high prices. The reduced price hypothesis predicts that buyers will enjoy lower prices products as a result of the increased competition among sellers in electronic marketplaces [Geun Lee'98].

Auctions and bidding have established methods of commerce for generations. These methods deal with products and services for which the conventional marketing channels are ineffective or inefficient. They can expedite the disposal of items that need liquidation or quick sale, they offer trading opportunities for both buyers and sellers which are not available in the conventional channels, and they assure prudent execution of contracts.

The Internet provides an infrastructure for executing auction and bids much cheaper, with many more involved sellers and buyers. Individual consumers and corporations alike can participate in this rapidly growing and very convenient form of electronic commerce.

Traditional auctions have several limitations and deficiencies. For example, they generally last only a few minutes for each item sold. This rapid process may give buyers little time to make a decision, so they decide not to bid, therefore sellers may not get the highest possible price, and bidders may not get what they really want or they pay too much. Also, in many cases, the bidders do not have much time to examine the goods. Since bidders must usually come to the auction site, many potential bidders are excluded. Similarly, it may be complicated for sellers to move

goods to the auction site. Commissions are fairly high, since a place needs to be rented, the auction needs to be advertised, and the auctioneer and other employees need to be paid. Electronic auctioning removes these deficiencies.

Electronic auctions have been in existence for several years. Notable are the auctioning of pigs in Taiwan and Singapore and the auctioning of flowers in Holland which was computerized in 1995 [Turban], but these were done on local area networks. Auctions on the Internet started in 1995. They are similar to offline auction, except for the fact that they are done on a computer. Host sites on the Internet act like a broker, they offer service for sellers to post their goods for sale and allow buyers to bid on those items. Most auctions open with a starting bid, which is the lowest price the seller is willing to accept. Detailed information on every item for sale is available online. For high value items, additional information may be obtained via e-mail. Bidders look at the descriptions and then start the bidding by sending an e-mail or filling out an election form. The bidding, which may last for a few days, are shown on a page at the host's Website, updated continually to show the current highest bids. Name of bidders are kept coded to maintain privacy [Turban].

Many sites have certain etiquette rules that must be adhered to in order to conduct fair business. For instance, Haggle Online, which allows private individuals to put up their merchandise for sale, has a page dedicated to inform their users of these rules. As with many other auction sites, Haggle Online offers a place where “honest, well-meaning individuals can offer their stuff for sale to other honest, well-meaning individuals.” The emphasis on honesty is important. If misrepresentation is made, one may sue the auctioneers as well, even though the auctioneers make it clear that they are not responsible for presentations made on the Website. The auctioning companies see the use of their services as an ideal channel for selling people's goods and warrant off any “spammers,” con-artist. Since it is in the best interest of the well-meaning users of these sites to have a clean and efficient system, the companies make it the users' job to maintain an honest and orderly auction site [Turban].

Since their first appearance in 1995, online auctions have increasingly gained momentum and accelerated the pace of Business-to-Consumer (B2C) and Consumer-to-Consumer (C2C) e-commerce. In the autumn of 1998, 142 auction websites, 90

percent of which conducted B2C transactions, generated almost \$100 million of trade each month. However, as e-commerce lacks face-to-face interaction and transactions are concluded online there exists a margin of risk with respect to security of payment, data protection, and transaction fraud. According to a very recent survey conducted by Euro barometer, the prime reason stated by consumers for not trusting the Internet for trading purposes was security of payment, where virtually 73% of European consumers who have not used e-commerce gave this reason. Delivery issues were also a great concern, where 37% indicated that this is the main reason for their mistrust of e-commerce. Similarly, 36% their lack of trust was based on their worries of getting a warranty or refund [<http://www.crime-research.org/articles/Wahab1>].

2-2 Type of Auctions

The major commonly used auction types are the Increasing-price auction (open-cry auction), Sealed bid auction (single and multiple rounds) and Decreasing-price auction (Dutch auctions.). The details of each auction are as follows:

* *Increasing-price auction* [Harkavy'98], called an 'English Auction' or 'Open-Cry Auction'. In this type of auction, a good or commodity is offered at increasing price. It may initially be offered at K tokens; at successive points of time i it is bid at $K+i*\Delta$ tokens (Δ may be function of previous bids and other factors). At each unit of time, one or more parties can bid for the item. At the end of the auction, the highest bidder takes the item; he pays the price he bids. This is the sort of auction found at Sotheby's and Christie's[Harkavy'98]. In the other word, the auctioneer and the participants gather in the same location, physical or virtual[Kumar'98], at the pre-specified time and the auctioneer starts the auction by setting an asking price for an item on sale, and requests bids from the floor. Periodically, the auctioneer resets that asking price to the value of the highest bid received from the floor, and starts a new auction round. Thus, the auctioning of each item may take one or multiple rounds[Panzieri'99]. This type of auction has many disadvantages: the time necessary to conduct the auction is potentially proportional to the price at which the item is sold; the communication costs may grow super-linearly in the ultimate price at which the item is sold (since at lower prices, multiple bidders may simultaneously bid for an item); moreover, this type of auction leaks an enormous amount of information-a careful observer will be able to deduce information about the price that each party is willing to pay for the

auctioned good. However, the auction does have a very desirable feature: in economic terms, it allocates the good to the bidder with the highest valuation, since the bidder with the highest valuation will be willing to outbid all other bidders[Harkavy'98].

* *Sealed-bid auction* [Harkavy'98]. In this type of auction, each party sends a Sealed bid to an auctioneer who opens all bids. The auctioneer determines the higher bid and sells the item to that bidder for the bidding price. In fact, buyers are required to submit their bids by a specified deadline. The auctioneer keeps the bid information secret until the deadline, at which time the bids are evaluated and the winners are declared. This type of auction can be executed in a single round or in multiple rounds of communication between the bidders and the auctioneers [Panzieri'99]. In a single round Sealed-bid auction, the bidders register their bids with the auctioneer by a specified deadline. Bids are kept secret by the auctioneer until that deadline expires. Then, the auctioneer evaluates the received bids and declares the winning bid, terminating the auction. As a result, single round Sealed-bid auctions lack the competitive atmosphere (bidding frenzy) of increasing –price auction which encourages the bidders to outbid their rivals[Kumar'98]. This type of auction has some disadvantages as well , first, the auctioneer will know the exact price that each party is willing to pay, and second, it does not support optimal distribution of goods[Harkavy'98]. Multiple rounds Sealed-bid auction, rectify this situation and there is a deadline for each round of bids, and at that deadline either the auction is closed or the bids from the current round are publicized and a fresh round of bids is solicited by some new deadline. In other words, in this type of Sealed-bid auction goods are auctioned through multiple rounds of bids. A deadline is associated with each round of bids; when that deadline expires, either the auction terminates, and the winning bid is declared, or the bids from the current round are made public, and a new round, and its deadline, are advertised .

In a Sealed-bid auction, participants will have beliefs about what others will bid. If a participant believes that they will have the highest bid, and the second highest bid will be substantially beneath that, then they have an incentive to lower their bid. In this type of auction, the winning bid is not necessarily the highest submitted bid; rather, the criterion with which a winning bid is selected can very depending on the

item being auctioned. For example, if the item is a contract for services, the winning bid is likely to be the lowest submitted bid [Panzieri'99].

* *Decreasing-price auction*, called Dutch auction [Harkavy'98]. This type of auction is similar to the English auction in that bidding price varies over time; however, in this case, the price decrease and time i is $K-i*\Delta$. Dutch auctions are used to sell perishable goods, such as vegetables or airplane seats. Auctioneer starts with a very high asking price [Kumar'98]. Then he gradually decreases his asking price until buyers emerge with bids specifying how many items they will purchase at the current asking price. The time period during which the auctioneer maintains the price of a lot unchanged can be very short; as the goods being sold are perishable goods, the sale of a lot can not last more than a few seconds [Panzieri'99]. This type of auction has the advantage of preserving maximum privacy, no information is revealed except the winning bid and bidder; however like the increasing-price auction, it may be time consuming, and like the Sealed-bid auction, it is not economically efficient [Harkavy'98].

* *Second-price auction*, called Vickery auction [Harkavy'98]. This type of auction works like a Sealed-bid auction, in that all bids are sealed and sent to an auctioneer. Like a Sealed-bid auction, the highest bidder wins. But the price the winner pays is the price that the second highest bidder has bid. There are Discriminative auctions, also known as Yankee auction [Kumar'98]. In this auction, the winner pays that bid and in the non Discriminative auction people with winning bids pay the price paid by the winning bidder with lowest bid.

When auctioning multiple homogenous products in this type of auction, the highest bidders are chosen in the same manner as in its first-price counterpart. However, all the bidders pay the same price, equal to the second highest bid.

It may, at first, seem as if this type of auction is unattractive to the seller of a good. It gives the impression of providing the same bidder with the good but at a lower price. Vickery found that this is not the case. In fact, the bidders, on realising that the price

they would pay, if successful, relies solely on the decisions of the other bidders, adjust their bids upwards. This results in the price outcome of the second-price, Sealed bid auction being the same as that of the near-optimally-efficient English auction, in theory.

* *Double Auction* , although not classified as one of the major four auction types, which have already been explained, the double auction has been the principal trading format in U.S. financial institutions for over a hundred years.

In this auction both sellers and buyers submit bids which are then ranked highest to lowest to generate demand and supply profiles. From the profiles, the maximum quantity exchanged can be determined by matching selling offers (starting with lowest price and moving up) with demand bids (starting with highest price and moving down). This format allows buyers to make offers and sellers to accept those offers at any particular moment. It can be confusing to think about the double auction in light of overlapping buy and sell orders. A good way to avoid this confusion is to understand that at one single instant of time, they do not overlap.

The origins of the double auction are not well known, but it is recognized that this form of auction has roots that go back to ancient Egypt and Mesopotamia. Almost certainly the double auction stems from "haggling" in which buyer and seller each suggest prices[Friedman'91].

Much later (in the last quarter of the nineteenth century) when the telegraph and telephone were invented, traders in the stock market could speak directly to interested outside investors. This was viewed at the time not only as a novelty but also as something of a threat. The computer revolution of the twentieth and twenty-first centuries portend far greater shifts as agents and financial markets become automated[Friedman'91].

A "continuous double auction" is one in which many individual transactions are carried on at a single moment and trading does not stop as each auction is concluded. The pit of the Chicago Commodities market is an example of a continuous double auction and the New York Stock Exchange is another. In those institutions a specialist matches bids and asking prices to find matches.

One interesting variation is the Double Dutch auction. Work on this is being done at the University of Arizona [Rassenti'92].

It works like this: A buyer price clock starts ticking at a very high price and continues downward. At some point the buyer stops the clock and bids on the unit at a price favourable to him. At this point a seller clock starts upward from a very low price and continues to ascend until stopped by a seller who then offers a unit at that price. Then the buyer clock resumes in a downward direction. The trading period is over when the two prices cross, and at that point all purchases are made at the crossover point.

Double auction has many variants and is evolving rapidly. Economists believe that the double auction will have many applications as auctions become computerized.

Each of these auction methods has precision variation such as [Kumar'98]:

- Anonymity, i.e., what information is revealed during the auction and after the auction closes.
- Rules for ending increasing-price auctions and decreasing-price auctions.
- Restrictions on bid amount, in all auctions the seller can specify the minimum starting bid.

In the survey of the auctions , from 142 sites of auctions , 121 used English increasing- price auctions, 21 used Sealed bid auction , 3 used Dutch decreasing-price auctions, and 4 were continues–trading double auctions. 6 of the sites had more than one auction type, which explains why the sum adds to more than 142. For example, the Auction Nation site gave sellers a choice between running an ascending-bid auction or a “silent” (or Sealed-bid) auction where the high bid is not made public until the closing time . The English auction type is even more domination than it first appears in the raw statistics. Of the 8 sites with a dollar volume of at least \$1,000,000 per month, all use an English type [Lucking-Reiley'99,].

2-3 Characteristics of Different types of Auctions

Table 2-1 shows the details of characteristics of different types of auctions which have been explained [<http://www.agorics.com/Library/Auctions/auction5.html>].

Characteristics of Different Types of Auctions	
Type	Rules
English Auction or Open-Cry Auction (Increasing-price auction)	Seller announces reserve price or some low opening bid. Bidding increases progressively until demand falls. Winning bidder pays highest valuation. Bidder may re-assess evaluation during auction.
Sealed Bid (First-price) Known as discriminatory auction when multiple items are being auctions.	Bids submitted in written form with no knowledge of bids of others. Winner pays the exact amount he bids.
Dutch Auction (Decreasing-price auction)	Seller announces very high opening bid. Bid is lowered progressively until demand rises to match supply.
Vickrey auction or Second-Price Sealed Bid. Known as uniform-price auction when multiple items are being auctioned.	Bids submitted in written form with no knowledge of the bids of others. Winner pays the second-highest amount bid.

Table 2-1

2-4 Complete Auction Process

These following items are necessary for complete auction process [Kumar'98] :

1- *Scheduling and advertising*: To attract potential buyers, items of the same category should be auctioned together at a regular schedule. Popular auctions can be mixed with less popular ones to force people to be present in the less popular auction. Items to be auctioned in upcoming auctions are advertised, and potential buyers are notified in this step.

2- *Initial buyer and seller registration*: This step deals with the issues relating to authentication of trading parties, exchange of cryptography keys and perhaps certain of a profile for each trader that reflects his interest in products of different kinds and possibly his authorized spending limits.

3- *Setting up a particular auction event*: This step deals with describing the item being sold or acquired and setting up the rules of the auction. The auction rules explain the type of auction being conducted, parameters negotiated, starting date and time of the auction, auction closing rules, etc.

4- *Bidding*: The bidding step handles the collection of bids from the buyers and implements the bid control rules of the auction and for increasing-price auctions notifies the participants when new high bids are submitted.

5- *Evaluation of bids and closing the auction*: This step implements the auction closing rules and notifies the winners and losers of the auction.

6- *Trade settlement*: This final step handles the payment of the seller, the transfer of goods to the buyer, and if the seller is not the auctioneer, payment of fees to the auctioneer and other agents.

2-5 Significance of Auctions

The Significance of auctions is their ability to lessen information asymmetries in a market. A seller will almost always have more information on the quality of a good they intend to sell than the potential buyers of that good. However, it is also common that the seller has very little information on the potential buyers' valuations of that good. An auction is suited to this type of situation where the seller can put the good for auction and gain additional information on buyers' valuations.

Another importance of auctions is a highly efficient method of resolving the supply and demand pulls on a market. Auctions in general are "efficient in the sense that an auction usually ensures that resources accrue to those who value them most highly and ensures also that sellers receive the collective assessment of that value.". In

addition to this, certain types of auction, namely the English, Sealed bid and first bid auctions, are considered to be Pareto-optimal.

2-6 Problems with Auctions

Auctions are not, however, without their problems. One of these problems, relating solely to the buyer, is termed ‘winner’s curse’. Since the value of the item is unknown, the winners can bid more than the value and thereby lose money. The winner’s curse occurs if the winners of auction systematically bid above the actual value of the objects and thereby systematically incur losses [Lind'91]. This phenomenon can occur in any auction type since all it takes is an over-estimation of the value of a good.

A supply-side problem that exists in auctions is one of collusion between bidders. Collusion occurs when rings of bidders get together and organise not to outbid each other. One member of the ring will be designated as the ‘winner’ of the good [Robinson'85]. This member will be the member who actually bids to win the good at auction. When the auction is over and the ring has successfully won purchased the good, it is auctioned off between the members of the ring. The benefit of collusion is a lower price for the good [Robinson'85]. Collusive rings of bidders do, however, have a difficult time enforcing congruent behaviour from its members. In many auctions, there is a definite incentive for ring members to renege on the agreement for personal gain [96]. It is considered that this problem for collusive rings can be used to discourage their presence in certain types of auction.

Different auction types are believed to have different susceptibilities to collusion. Looking at the four main types of auction outlined by Vickrey [Vickery'61], the following collusion ordering is thought to exist [96]:

- 1) Increasing-price auction (English Auction)
- 2) Sealed-bid auction , Second-Price, (Vicker)
- 3) Sealed-bid auction , First-Price
- 4) Decreasing-price auction (Dutch Auction)

The English auction is the most predisposed of the above auctions to collusion. There is no real incentive for the member of the ring to cheat on their agreement. If they attempted to outbid the ring, it would be clear for all to see. With the two Sealed bid auctions collusion is less likely. If these auctions are conducted over a single round then a ring member can cheat and not be detected until after the auction has completed, if at all. The Dutch auction is considered the most collusion-proof auction form. Although this auction uses an open format, the cheating member can still benefit from going behind the back of his fellow ring members. When the cheating member makes his bid, the auction will finish and it will be too late for the remainder of the ring to take any action. However, when collusive rings are expected to last for several auctions, the openness of the Dutch auction becomes a sufficient deterrent to the would-be cheater.

There are several methods, in addition to the use of Sealed bid auctions that can be used to deter the formation of collusive rings of bidders. If the auction house keeps the information on the winning bidder secret, this reintroduces the ability for the cheating ring member to renege on the collusive agreement without being detected [Ferbo'96].

If sellers and auction houses use the reserve price on auctions in aggressive manner, they can reduce the expected gains that a collusive ring can expect to make. This will be reducing the likelihood of the formulation of such a collusive ring [Ferbo'96].

Auctions containing small, frequently auctioned volumes of goods are more likely to attract collusion. These types of auctions provide the ability for rings to rotate the designated winner, so that all in the ring get a fair share of the profits. By packaging these 'small volumes of goods into one lot at a single auction, the chance of encountering collusion is reduced. It should be noted that although these small auctions are more favourable to collusive rings, collusion does exist in the bigger auctions [Ferbo'96].

2-7 Internet Auctions

An Internet auction simply takes the traditional auction formats and applies them to a communication medium which gives them a wider ranging audience and makes them more efficient. By placing an auction on the Internet, competition can be seen to increase not only among buyers, but also within the market for auction services. Many companies have moved into the market for auction services, seeing the excessive revenues and success reaped by the unquestioned market leader, eBay, www.eBay.com. This competition means that very few Internet auctions charge a commission on transactions negotiated on their web site. Those Internet auctions that do charge a commission ask for a minimal value when compared with the traditional auction houses, and it is likely to only to apply to one of the participants [Dawe'00].

The Internet auction incorporates most of the features of the traditional auction, while it also adds features necessary for its new media. Due to the geographically dispersed nature and the sheer number of the auctions relatively anonymous participants, some method of bidder and seller reputation had to be established. The majority of Internet auctions now incorporate some method whereby a bidder or seller's peers can give him/her a rating and write comments on their behaviour and trustworthiness. Other bidders can then use these rating and comments as a guide to who they are dealing with.

One of the major advantages of Internet auctions is the audience that they reach. With such a great number of people possibly wanting to view the auction, it was necessary to increase the time that auctions ran from the model of the traditional auction house. Internet auctions tend to run for several days. This gives all potential participants sufficient time to realise that the auction exists and for rounds of bids and counter-bids to be placed. However, due to the increase in the time frame, it is not feasible for a person to observe the entire auction. While it is possible for users to check back at regular intervals to keep abreast of the auction's progress, it can be inconvenient. As a solution to this, many Internet auctions provide a service for automated bidding. With an automated bidding service, the bidder generally provides details of the maximum bid he is willing to place for an item and the automated bidder will constantly outbid any other bidders, up to the pre-specified maximum bid.

2-7-1 Existing Internet Auctions

In recent years a number of auction services have been made available over the Internet (e.g., www.ebay.com, www.antiquorum.com, www.artnet.com, to name a few).

One of the market leader in Internet auctions is eBay, www.eBay.com, who started the US arm of their company in 1995 and since then have expanded their operation to the UK and Europe. It is estimated that “users of eBay place roughly 1,000 bids a minute on more than 4.5 million items for sale at any one time” [Broughton'00]. It is also believed that 1 in every 20 packages currently sent between private individuals in the US is a purchase from eBay [Grossman'99]. It is this sort of popularity that led to eBay winning the Cool Site of the Year Award for Cool Shopping in 1999.

At the beginning of the year 2000, eBay had a registered user base of 10 million people. Taking this into account, and the fact that they charge a fee for their auction services, it is not surprising that eBay is one of the few Internet ventures to actually turn a profit [Kaufman'99].

In both the US and European markets, there is no shortage of competitors for eBay. Since its conception 5 years ago, many other companies have emerged wishing to follow in the footsteps of eBay. While this competition is welcomed by consumers, the major Internet auction companies are reluctant to relinquish their strangle hold on the market. However, since the success of an Internet auction venture is directly related to its user base, FairMarket, a US auction software company, plans to threaten the market by combining the user base of multiple sites. FairMarket works by allowing items listed on one auction site to be viewed and bid for on other sites. FairMarket has already got one hundred sites included in their auction venture, some of which are big names such as Excite and MSN, and already claim to be the third most popular Internet auction in the US.

Internet auctions can be characterized further as follows [Amoroso'03].

- Firstly, in order to enable the largest possible number of participants to take part in an auction, the starting time of that auction is to be announced in ample advance. In an Internet auction, this announcement can be easily disseminated all over the planet. This announcement may well include the deadline by which bids are to be delivered to the auctioneer. This deadline may coincide with the end of the first auction round (e.g., in an open-cry auction), or with the end of the auction itself (e.g., in a single-round sealed bid auction).
- Secondly, depending on the auction type, the auction announcement may include the asking price for each of the various items that will be sold by auction.
- Thirdly, participants can join a real auction already in progress; in an Internet auction, this entails that those participants are to be made aware of the current auction state, in order to be enabled to take part to that auction.
- Fourthly, each type of auction is characterized by both a specific time duration of the rounds in which the auction is structured, and a particular evolution of the selling prices of the items on sale by auction. For example, the time duration of a round in an open-cry Internet-based auction can be set to a few minutes (e.g., 2 to 3 minutes), and the price of the item increases as the auction progresses. In contrast, a round in a Dutch auction can last a few (e.g., 10 to 20) seconds, and the item asking price tends to diminish as rounds progress. Finally, a round in a multiple round sealed-bid auction can last a few days, and the asking price of the item at auction may or may not vary depending on the specific auction policy.

2-7-2 Advantages of Internet Auctions

Internet auctions have several advantages over traditional auctions, as well as other forms of online trading. The increased market size and economical pricing of Internet auction services have already been mentioned. In addition to these, there is no need for the buyer or seller to be physically present at an Internet auction. This can save them both a lot of effort if an auction is being run on the other side of the world. It can also save the seller expense that would be needed to transport their product to the auction house.

Businesses find Internet auctions extremely useful for clearing out excess stock. Whether this stock comes from a failed order, modernisation or un purchased goods from a sale, using an auction allows the company to extract a price for the goods from the market. The relative anonymity of Internet auctions allows companies to hide the fact that they are clearing stock. The large potential market provides those companies with specialised equipment with a greater chance of receiving somewhere close to its market value by finding people who can use it [Deutsch'98].

2-7-3 Problems with Internet Auctions

As might be expected, Internet auctions are not without their problems. One of the major problems they face is that of fraud. Internet auctions are now thought to be the most common Internet con [Schwartz'98]. The US Federal Trade Commission has seen an increase to 10,000, from 107 in 1997, in the number of complaints about Internet auction fraud [Clausing'00].

The fraud can take on a number of forms. One of the more popular forms is referred to as shill bidding, or show-bidding. This is when a seller makes bids for his own product, making the price increase and the good look as though it is highly sought after. This practice is also present in traditional auctions, although in the brick-and-

mortar institutions the ability to see the other bidders makes this practice easier to detect [Schwartz'98].

The peer feedback mechanisms used by many of the modern Internet auction houses have also been used in a fraudulent manner. Users get friends, or may even do it themselves, to post favourable reviews and feedback about them. This gives other users a false sense of the reputability of the fraudster. This can greatly affect a user's decisions, and can lead them into a fraudster's con [Knight'00].

The form of fraud that causes the most concern among bidders is non-delivery of goods. On the completion of an auction, the seller is provided with the winning bidder's contact details, and the winning bidder gets the seller's contact details. It is often left up to the two parties to complete the transaction. The usual manner in which this takes place is for the winning bidder to send the money to the seller, and on receipt of the money, the seller sends the goods. However, once the seller has the money, he/she may not always send the goods. It is for this reason that several Internet auctions also recommend the use of Escrow services. These services provide a safe way for the exchange of goods for money. The winning bidder sends the money to a third party (the Escrow service), who informs the seller that they have the money. The seller then sends the goods to the winning bidder, who informs the third party of the receipt and acceptability of the goods. On receiving this confirmation, the third party sends the money to the seller. The only drawback with this service is that a charge is made for it. If the value of the purchase is not very large, then the charge for the Escrow service may seem worthwhile. Even if the goods do make it to the winning bidder, there is no assurance that they will be as described [Schwartz'98].

The media that Internet auctions work with can also create problems. In June 1999, eBay faced possibly its toughest problem yet. A series of problem with its server resulted in sporadic crashes, leaving its auctions inaccessible. The peak of these crashes came with a 22-hour period of downtime. The reasons for these outages ranged from problems with hardware to database corruptions [Wice'99]. While eBay

claim that excessive traffic played no part in the failures, it can be seen that issues of scalability exist in the context of Internet auctions.

Leaving bidding until the last minutes of the bidding phase of an Internet auction is becoming a more and more popular way to approach the trading mechanism. By leaving bidding to the final few minutes, there is less time for a rival to bid to make a counter-bid. This can clearly be seen to be harmful to the efficiency of the auction, as sales will occur at reduced prices due to a lack of competition. The scope for last-minute bidding arises from the fixed auction deadlines that many Internet auctions use.

The final problem of Internet auctions is excessive bidding. During the rounds of bidding, many bidders can get carried away with the enjoyment of auction trading, so much so that they end up winning the good at a price considerably greater than the value of the good. Stories abound of bidders bidding excessive amounts for goods such as a \$100 gift certificate [McGrane'00]. This problem effects the efficiency of an Internet auction. The good may not be going to the person who values it most highly, as the person may not have an accurate perception of their own valuation due to, what can be considered, a form of 'auction fever'. A solution to this problem was implemented by German company 12Snap. After several weeks of excessive prices winning the auctions, the company established price ceilings to stop bidding getting out of hand ['99].

2-7-4 Other considerations relating to Internet Auctions

The process of an Internet auction is very similar to traditional auction. There are registration, initialisation, bidding, closing and transaction phases involved. However, some of the time frames of these phases are different in an Internet auction. Auctions on the Internet are usually not held for multiple lots, as they are in traditional auctions. Individual auctions are set up for each lot on the Internet. For this reason, the bidding phase is extended to allow potential bidders to see that the item is up for auction and to submit a bid. The time taken to register a seller is greatly reduced. The seller needs

only to fill out a form outlining the features of the auction they wish to create for their good [Dawe'00].

More technical aspects must also be addressed. As the Internet auction will serve many different users with differing needs, it is important that any Internet auction provide a variety of auction market types. This gives the seller the ability to use the efficient type for their good.

2-8 Auction Requirements

Auctions involve competitive bidding among buyers and sellers of goods, and their provisioning places new demands from the underlying distributed computing infrastructure.

Each auction needs the following requirements:

Data integrity: It must be guaranteed that the customer (virtual) shopping basket will not 'lose' purchased items. Data integrity must therefore be met as a requirement of auction service, in order to preserve data consistency in the face of concurrent accesses and occasional system failures. Data integrity is achieved by ensuring atomic interactions between a bidder and a server and by keeping replicated data mutually consistent. Moreover, security mechanisms, based on cryptographic methods and audit trails are required to resolve any disputes and reduce the possibility of misbehaviour (e.g. a customer disclaiming an order, a shop keeper collecting payment for articles in the catalogue that are not available for delivery).

Scalability: An auction service needs to be scalable, i.e., capable of providing its end users with "satisfactory" Quality of Service (QoS), regardless of the number of those users and their geographical distance. We therefore investigate ways of enabling widely distributed, arbitrarily large number of auction servers to cooperate in conducting an auction. Scalable auction can allow interactions among a very large number of customers and suppliers of goods.

Responsiveness: Responsiveness requirement, i.e. a service must be timely and available (availability and timeliness) under specified load and failure hypothesis is motivated by the observation that a service that exhibit poor responsiveness is virtually equivalent to an unavailable service. Thus, within the electronic commerce business loss for its provider.

The goals of responsiveness and scalability are achieved by replicating the auction/bidding service across a number of these auction servers. Allowing users to place a bid at any of the servers in our principal way of achieving scalability and responsiveness, as the total load is shared amongst many users, and users can interest with servers 'closest' to them (advance forms of load balancing strategy are also possible).

Fairness: Fundamental fairness property of an auction must be preserved: all participant bidders in the auction must have an equally fair chance for submitting a successful bid, and that all participant sellers must have an equally fair chance for selling their items (all buyers must be granted an equal opportunity to buy the goods offered by a seller). Achieving fairness of auctions conducted over the Internet using a single auction server is a challenging problem as it is [Peng'98; Wurman'01], since differing message transmission delays experienced by bidders can clearly compromise an auction's fairness. Achieving fairness of an auction conducted over a group of auction servers makes the problem even harder, but this problem must be solved in order to obtain scalability without sacrificing responsiveness.

2-9 Related work

As it has already been mentioned, the first popular auction service was eBay, established in September 1995 [Baldwin'99]. The fast growing popularity of eBay pointed out that the main advantage of on-line auctions was the broad base of possible clients they may reach [Wrigley'97]. However, on-line auctions showed some shortcoming, such as the possibility of frauds, and the problem of maintaining the anonymity of the parties [Amoroso'03].

In the literature, few papers propose distributed architectures for auction systems. An early distributed e-commerce system was Ench`ere [Ban`atre'86] that, in 1986, implemented a prototype of an agricultural marketing system consisting of a loose network of autonomous workstations communicating via message exchange. The auction model was the Dutch auction, due to the perishability of the goods on sale. Ench`ere supports disjoint groups of sellers and buyers connected via a network; auctions in each group can proceed in parallel to the ones in the other groups.

An interesting auction process model is described in [Rachlevsky-Reich'99]. The global market is subdivided in several markets, and the seller starts an auction of its item in the local market. If the seller does not receive bids better than the asking price, it starts new parallel auctions in selected remote markets. The seller continues the starting of new auctions until she/he receives a satisfactory bid; in the case of multiple winning bids, coming from different markets, the seller applies a conflict resolving technique to assign the item. In this model several auctions can be performed in parallel in order to sell the same item, allowing the system to be naturally scalable with respect to both the number of clients and servers. Moreover, a crash of a server during the evolution of an auction does not affect the evolution of the whole auction; rather, it simply results in the casting away of the local market, relative to the crashed server. The principal shortcomings of this model are that, firstly, it lacks the idea of a global single auction; hence, an auction starts with a local portion of the total possible bidders, and gradually increases them by reaching new markets. Secondly, this model does not guarantee that an item be sold at the best possible price, as the final selling price for an item is the one that meets the seller expectancy. Thirdly, this model is suitable for English auctions, only. In Dutch and Sealed bid auctions the timing is an important constraint, which contrasts with the idea of gradual additions of remote auctions' [Amoroso'03].

Several works, such as [Franklin'66; Harkavy'98], deal with Sealed-bid auctions. Those are long standing auctions; the main focus of the systems that implement them is on the security, validity and secrecy of the bids, rather than responsiveness.

It is argued in [Huhns'99] that, in the near future, the on-line auctions market will be dominated by agents that will bid on behalf of human users. This scenario might lead

to a new form of price definition completely different from human experience. Several papers propose agents as an emerging technology to implement auctions over the Internet; e.g., [Collins'01] [Mullen'98] [Sandholm'00]. The principal idea of agent based auctions is that an agent, acting on behalf of a user, may search the Internet for the required goods, and buy them to the “best” price, as defined by that user. The agents auction follows the usual rules; the agents may implement sophisticated bidding policies in order to get a good at the best possible price. In [Greenwald'01; Team'01] the authors describe a competition among agents in order to test both a feasible framework for agent based trade, and trading strategies. In these papers, the competition is principally focussed on strategy issues, and the competition framework is centralized.

A completely different approach to e-commerce is discussed in [Yuan'98] where the parties of a bargain are allowed to negotiate by means of an Internet application, called CBSS, that provides them with several interaction services, such as videoconferencing, whiteboard, document sharing. This system is aimed to replacing a face-to-face negotiation between parties.

Regarding [Feldman'00], electronic marketplaces introduce notable technical challenges, and stimulate new forms of trading. Interesting analyses of the dynamics of prices of goods in electronic marketplaces, and a comparison with the traditional marketplaces, can be found in [Geun Lee'98; Wurman'01] , [Samret'00].

The “real-time” aspect of a distributed architecture for auctions over the Internet represents a challenging problem due to the best effort nature of the communication network [Wellman'98; Fay-Wolfe'00] . Some authors propose a centralized real-time protocol for a client-server auction architecture based on Java applets [Peng'98]. This solution requires strict clock synchronization between the server and the applets running on the clients computers, a fair multicast from the server to the applets, and timely processing and delivery. The system performs periodic updates of the state of the auction, during which the client cannot put bids. The architecture proposed in [Peng'98] is poorly scalable, owing to the strict centralization of the server. In the already cited paper [Maxemchuk'01] , the authors discuss a system to support the stock exchange market. The main focus of that system is on the total ordering of the

bids, trust and responsiveness. The authors propose a twofold hierarchical architecture in order to obtain both acceptable performances and scalability. This architecture has different goals than ours, and the solutions it deploys are not well suited in our context. Finally, [Lin'03] investigates issues of design of small-scale auction applications, based on wireless, ad-hoc networks.

Current Internet-based auction services rely, in general, on a centralised auction server. Such an approach is fundamentally restrictive as too many users can overload the server, making the whole auction process unresponsive. As it has already been mentioned, we require the properties of scalability, responsiveness, fairness and data integrity to be met in an auction. In addition, the centralised auction server architectures exhibit a number of limitations, including the followings:

Firstly, a centralized architecture cannot deal adequately with issues of service availability and scalability which have already been explained as requirements of the auction. Typically, such an architecture can be vulnerable to server's failures, if not equipped with sufficient redundancy; in addition, server's overloading may occur, if an arbitrary large number of users concurrently access the service. The increasing number of customers of Internet based auction services suggests that both these issues are crucial in the design of those services.

In particular, as pointed out in [Panzieri'99], service availability is required as a frequently unavailable service may discourage users from using it, and results in a business loss for its provider. The service scalability is necessary as an auction service is expected to provide all its users with an equally satisfactory (and fair) service, regardless of the number of those users and their geographical location.

Secondly, an Internet-based auction service must be accessible to users that are distributed, at least in principle, on an international, possibly planetary, scale; thus, that service may have to deal with different national selling rules that pertain to individual countries. As pointed out in [Ezhilchelvan'01], within this scenario a centralized architecture may turn out to be inadequate, as a great deal of complexity may have to be incorporated in the centralized auction server, in order to deal with those different selling rules.

It should be mentioned that the additional crucial requirements including, security, privacy and anonymity are also to be met in internet-base auction services.

2-10 Conclusions

E-commerce is increasingly expanding its share of the world trade, and is becoming a global phenomenon that is not affected by physical or territorial barriers. However, the lack of face-to-face interaction in conducting e-commerce acts as a barrier to trust and confidence, especially for consumers who may be dealing with total strangers thousands of miles away. The relative risks associated with e-commerce are further aggravated in online Internet auctions.

Internet and World Wide Web have emerged as a valuable networked information source that is increasingly being used for commerce. A particular class of Internet based server-centred application whose user domains are typically large, geographically distributed, and perhaps expanding. Examples of these applications are on-line Internet auctions, Internet gaming, etc. On-line Internet auctions are continually expanding into diverse products ranging from second-hand goods to airline tickets and financial products [Ezhilchelvan'01].

An auction service is required to be scalable, i.e., capable of providing its end user with “satisfactory” Quality of Service (QoS), regarding of the number of those users and their geographical distance. It should therefore develop novel ways of enabling widely distributed, arbitrary large number of auction servers to cooperate in conducting an auction. However, the fundamental fairness property of an auction must be preserved: all participant bidders in the auction must have an equally fair chance for submitting a successful bid, and that all participant sellers must have an equally fair chance for selling their items. Achieving fairness of auctions conducted over the Internet using a single auction server is a challenging problem as it is [Kumar'98; Wellman'98], since differing message transmission delays experienced by bidders can clearly compromise an auction's fairness. Achieving fairness of an auction conducted over a group of auction servers makes the problem even harder, but

this problem must be solved in order to obtain scalability without sacrificing responsiveness.

Currently available internet based auction services, which are rapidly diversifying into various products, such as eBay [<http://www.eBay.com>] and so on, essentially rely on a central auction server. As the market and internet trading grow, existing central internet auction server and such an approach is fundamentally restrictive as too many users can overload the server, making the whole auction process unresponsive.

The goals of responsive and scalability are achieved by replicating the auction/bidding service across a number of auction servers. Therefore allowing users to place a bid at any of the servers is principal method of achieving scalability and responsiveness, (total load is shared amongst many servers and users can interact with servers 'closest' to them). Data integrity is also achieved by ensuring atomic interactions between a bidder and a server and by keeping replicated data mutually consistent.

In this regard, to achieve distributed auction systems from existing central auction systems, the Middleware technologies and group communications as a requirement tools, for our novel architecture and its implementation, will be described in next chapters.

Chapter 3

Regarding what has been discussed in chapter 2 and also the necessary tools to implement distributed auction system which will be described in next chapters, in the following chapter the relevant issues related to the provision of an object group communication service for use in Middleware environments are described. A description of the Middleware technologies and the mechanisms that may be required to provide a group communication service which will be used in our structure in next chapters are also explained.

3-1 Middleware

A middleware service is a general-purpose service that sits between platforms and applications (see Figure 3.1). These services shield the application developer from platform specific type services. The term platform , indicate low level services and processing elements defined by processor architecture, an operating system's application programming interface (API) and communication primitives (such as sockets for inter-process communication). This section defines Middleware technologies via the properties they seek to exhibit and describes the mechanisms that are common in the enabling of such technologies. A middleware service is defined by the APIs and protocols it supports. It may have multiple implementations that conform to its interface and protocol specifications.

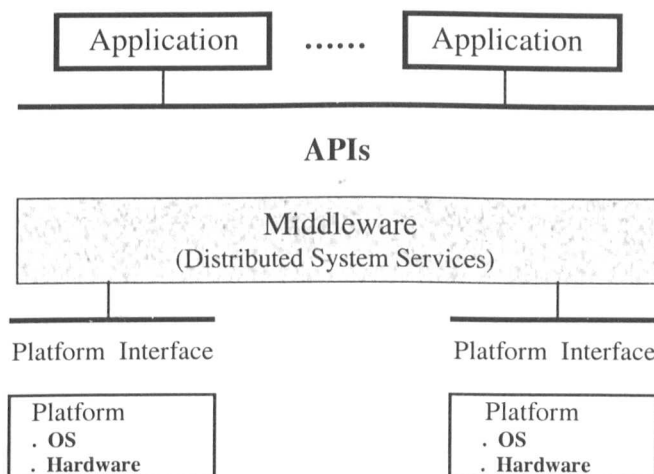


Figure 3.1

Like many high-level system concepts, middleware is hard to define in a technically precise way. However, middleware components have several properties that, taken together, usually make clear that the component is not an application or platform-specific service: They are generic across applications and industries, they run on multiple platforms, they are distributed, and they support standard interfaces and protocols.

3.1.1 Properties of Middleware

As explained in section 3.1, Middleware is commonly understood to mean the layer that sits between applications and operating systems; its function is to ease the development of distributed applications. The Middleware is characterized by four properties which are shown in below:

- Platform independence - Implementations should be available over a number of different operating systems and hardware configurations. The types of available platforms should not hinder an application developers use of a Middleware.

- Distribution - Capable of providing support for services that are not restricted, geographically, to a single location.

- Provision of standard interfaces and protocols - Irrelevant of implementation environment, protocols used by Middleware and interfaces to the Middleware remain the same. As networked systems tend to be heterogeneous in nature, propriety service APIs may be present on a per-platform basis. A developer that becomes proficient on one platform may not be able to easily transfer his/her skills to another type of platform.' To avoid this, a Middleware technology should seek to provide an API that does not deviate, irrespective of which platforms it resides.

- Generic - Meets a wide variety of application requirements across many industries.

The client/server model is commonly used in Middleware in an attempt to satisfy properties of distribution and generic. This is because clients and servers are suitable units for distribution (second property) and have been used to build a wide variety of

application types (forth property). In the client/server model a server satisfies the service requirements of one or more clients. A service is defined by an interface. This interface indicates to a client the manner of interaction a client must assume to gain service from a server. Interaction is realized by a client issuing requests via a suitable message passing protocol and the server replying to client request via the same protocol. Achieving platform independence and distribution properties have proven more difficult. This is due to the variety of low level services and the large number of different organizations involved in the development of such services. Organizations have to cooperate to formulate and agree on industrial standards and adhere to them to ensure that programming APIs and protocols remain consistent over various platforms, irrelevant of the vendor supporting the Middleware implementation. This section continues with a description of a method commonly used to implement the client/server model in Middleware.

3.1.2 The Proxy/Stub Method

This method is used to enable a remote procedure call (RPC). In essence, a proxy resides in the same address space as a client and presents the client with an interface to a service. This interface is presented in a manner that would suggest to the client that this service is no different than any other service located within its own address space. However, requests directed at the proxy interface are then forwarded, by the proxy, across process boundaries, and more usually a network, to the actual service implementation. At the service side a stub, located in the same address space as the server, is responsible for receiving these requests and then forwarding them to the service implementation, receiving any replies, and then returning these replies to the proxy. Replies received by the proxy are then returned to the client.

3.1.3 Distributed Objects

There are three dominant types of Middleware that provide an object-oriented approach to distributed application development:

- Sun Microsystems' Java with Remote Method Invocation (Java-RMI) [Sun Microsystems'97]
- Microsoft's Distributed Component Object Model (DCOM) [Brown'96]

- The Object Management Group's Common Object Request Broker Architecture (CORBA) [The Object Management Group'95]

To ease the production of proxies and stubs for use within distributed applications, Java RMI, DCOM and CORBA provide:

- A language for defining an interface of a service.
- Some mechanism for automating stub/proxy generation from an interface definition.
- A method, appropriate to the target languages of the client and server, for integrating proxies and stubs into client and server code.

More details description of Java-RMI, DCOM and CORBA follow in next section.

3.2 Object-Oriented Middleware Technologies

The purpose of this section is to describe Java-RMI, DCOM and CORBA. Due to the substantial subject areas each of these may cover, a simplified view of the processes required to produce proxy/stub code and the enabling of inter-object communication via this code is described.

3.2.1 Java-RMI

The Java programming language enables application developers to write object-oriented programs that may be executed on a variety of platforms without alteration. This is achieved via an environment that provides a consistent API within which a Java program may execute. This environment is termed the Java virtual machine (JVM).

Once written and compiled, Java code will work wherever a JVM exists. However, it was not until 1997 that support was added to the Java language that enabled objects in different address spaces to communicate using the proxy/stub mechanism. This support took the form of the Java Remote Method Invocation (Java-RMI).

Java-RMI is designed to work when clients and servers are implemented in Java. There is no support for clients and servers if they are implemented in other programming languages.

3.2.2 DCOM

DCOM (Microsoft's Distributed Component Object Model), previously known as Network OLE (Object Linking and Embedding), is an extension of the COM (Component Object Model) designed to network applications. The underlying objective of COM is to permit the independent development of software components that can intercommunicate, regardless of language or function. The unit of distribution in the DCOM environment is commonly termed a component.

A component exports one or more interfaces that defines its functionality. Interfaces may be constructed in an object-oriented fashion, allowing application developers to make use of polymorphism and inheritance. A component consists of an array of function pointers, each pointer indicates the physical address of a method supported by the interface. By placing components into a Dynamic Link Library (DLL) applications may link to (bring into their own address space) components at run time that are implemented in arbitrary languages. This increases code reuse and allows components to be distributed amongst applications.

Extending COM to DCOM required the introduction of an Interface Definition Language (IDL) that aided the production of proxies and stubs for use by clients and servers respectively. The IDL used by DCOM is based on the IDL standard specified by the Open Software Foundation (OSF) for use with its Distributed Computing Environment (DCE) [Rosenberry'92].

3.2.3 CORBA

The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is a widely accepted standard for Middleware. Over 700 companies endorse the standard with implementations of the standard existing on most operation systems.

Objects may interact irrespective of the languages used for their implementation; service providers may be implemented in language *A* whereas clients of such a service may be implemented in language *B*. This interoperability is achieved by

ensuring all service providers specify the services they provide via a standard language (IDL). Unlike DCOM, the IDL language used by CORBA resembles a more traditional object-oriented language. CORBA and DCOM approach the issue of separating implementation from interface in the same manner; via an IDL.

To produce the appropriate proxy/stub code required to enable clients and servers to interact, an IDL interface is passed through a parser (supplied by a vendor). The language of the code produced depends on the parser. Most parsers accommodate C++ or Java. The proxy/stub code produced implement a layer of abstraction known as the Object Request Broker (ORB) within the CORBA environment. To ensure cross compatibility over different platforms between IDL and target languages the CORBA standard specifies mappings from IDL data types to data types found in various languages. A simplified view of the production of a (possibly) remote service is thus:

1. Specify a service using a CORBA IDL
2. Create Proxy and stub code - Pass the interface through a parser supplied by a vendor.
3. Implement the service - The object that implements the service is written in the same language as the proxy/stub code. Most parsers present developers with "ready to use" skeleton code suitable for implementing the service.
4. Write a server program to support the service - The server program creates an instance of the object that implements the service and activates the required mechanisms within the CORBA environment to ensure communications between the service and clients may occur.
5. Publicize server to clients - A mechanism is required to enable clients to retrieve a reference to the service. This may be done via the "Naming Service" (a service where clients can request services by a well known name and retrieve appropriate service references). Alternatively, service references may be cast into the form of a string and passed to a client by other methods .

6. Create client - A client is created with the appropriate proxy code included in the client source code.

7. Enable client/server interaction - Clients retrieve the object reference of a desired service (either via the naming service or by other means). Once a reference is retrieved communications between client and server may commence.

3.3 OMG and CORBA

The OMG, which has been explained, developed a conceptual model, known as the core object model, and a reference architecture, termed the Object Management Architecture (OMA). The OMA consists of four components: Object Request Broker (ORB), Object Services (OS), Common Facilities (CF), and Application Objects (AO). CF relate to object services that aim to satisfy quite specific application requirements (e.g., e-commerce, database management systems) and AO relate directly to applications.

3.3.1 ORB

The core of the OMA is the ORB. The ORB is a communication bus for objects. The ORB architecture specifies an IDL for defining objects and a protocol, Internet Inter-ORB Protocol (IIOP), for enabling inter-object communications. IIOP is a protocol that specifies how detailed information representing a CORBA request is laid out on a network transport service. IIOP ensures multi-vendor interoperability between ORB implementations. Any functional enhancements to the ORB are achieved via object services. This ensures that applications will work on any ORB, irrelevant of the vendor supplying the ORB.

The IDL is simply a declarative language that supports no scope for programming implementation details. This is left to a programming language of the developer's choice. IDL is network neutral and operating system neutral, thus preventing developers from introducing platform dependent mechanisms into a service's IDL definition.

An Interoperable Object reference (IOR) is used to uniquely identify objects in CORBA. An IOR is a sequence of object-specific protocol profiles, plus a type identifier. The IOR is not intended to be visible to application programmers. Programmers are presented with a suitable structure available in the programming language of their choice to represent an IOR.

The IDL allows an object reference to be passed as a parameter in a function call. This is the mechanism that enables the distribution of object references between objects. In addition to this mechanism, a developer may derive a string representation of an IOR and derive an IOR from a string representation. This is useful for passing object references by other methods.

3.3.2 CORBA Interface Architecture

The CORBA specification defines a number of interfaces to allow clients and servers to participate in inter-object communication. These interfaces are described in following section:

- **IDL Proxy** - The IDL Proxy (sometimes termed IDL stub) presents an interface derived from an IDL definition of a service and are linked into the client program.
- **IDL Stub** - The IDL Stub (sometimes termed IDL skeleton) is simply the server side counterpart of the IDL proxy.
- **Dynamic Interfaces** - Statically including proxy/stub code derived from an IDL into client and server programs to enable inter-object communication satisfies the communication requirements for many applications. However, there are instances when this is not adequate. The static mechanism assumes clients are aware of servers and that servers are aware of the way they must satisfy client requests at compile time. This may restrict the way an application may evolve; allowing existing clients to use new services which are introduced during the lifetime of an application becomes difficult. To overcome this problem dynamic interfaces are supported by the CORBA standard.

- Dynamic Invocation Interface (DII) - Enables the specifying and building of requests at run time, rather than calling linked-in proxy code. Operations supported by the DII include: create_request, invoke, send, get_response. Invocations made by the static and dynamic methods are indistinguishable by the server object.
- Dynamic Skeleton Interface (DSI) - The server side analogue to the client side DII. The DSI inspects the parameters of an incoming request to determine a target object and method. This interface allows a service to assume the role of another service.
- ORB Interface - Enables direct access of the ORB by clients and servers.
- Basic Object Adapter Interface - The server program that supports the objects that implement services (defined by IDLs) is aided by the Basic Object Adapter (BOA). The server program registers objects ready for use with the BOA. Once this has occurred the BOA manages requests on behalf of the server's objects. Due to the fact that interaction occurs directly between an application and an ORB (and the possibility of an application to be programmed in any one of many languages), defining this interaction was left to vendors. This has resulted in the presentation of these mechanisms in a number of different ways, making it difficult to port code from one vendor's ORB to another. To overcome this, the next release of the CORBA specification identifies a Portable Object Adapter (POA) that seeks to standardize direct application to ORB communications. Figure 3.2 indicates how the interfaces of an ORB are integrated and which interfaces interact with clients and which interact with servers.

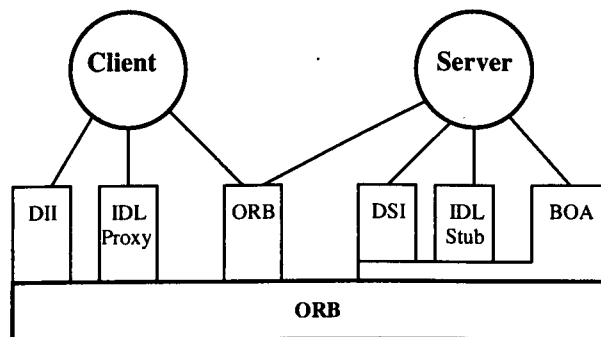


Figure 3.2

3.3.3. Object Services

Following are brief descriptions of object services which are in common use:

- **Naming Service** - Supports name-to-object association. A hierarchical naming structure has been adopted. This allows clients to retrieve the IOR of an object using a reasonable name. The mechanism that enables a client to gain the IOR of the naming service is left to the ORB vendor.
- **Event Service** - De-couples the communication between objects. Objects may assume the roles of supplier or consumer. The service defines two approaches for initiating event communication: the push model and the pull model. A supplier uses the push model to transfer event data to consumers. Consumers use the pull model to request event data from a supplier. The event channel is simply an intervening object that allows multiple suppliers to communicate with multiple consumers simultaneously in an asynchronous manner.
- **Lifecycle service** - Represents a framework for creating, deleting, copying and moving objects. The creation facility is most commonly available in CORBA applications. As there is no existence of a basic creation facility in CORBA IDL factory objects are used to create instances of particular types of objects. There is no standard interface for a factory object, an application developer is to develop their own in a manner they see as appropriate. Usually, for each type of object there is a factory object.
- **Persistence Service** - Provides common interfaces to the mechanisms used for retaining and managing the persistent state of objects in a data store in an independent manner.
- **Transaction service** - Ensures that a computation of one or more operations on one or more objects provides properties of atomicity, consistency, isolation and durability (ACID properties).

3-4 Fault Tolerance and Reliability Issues

The CORBA standard incorporates support for reliability through the following two distinct mechanisms: Replication (using the Fault Tolerant CORBA standard) and Transactions (using the CORBA Object Transaction Service). Transactions represent a roll-back reliability mechanism, and handle a fault by reverting to the last committed state, and by discarding operations that were in progress at the time of the fault. Replication represents a roll-forward reliability mechanism, and handles a fault by re-playing any operations that were in progress at another operational replica of the crashed server [Felber'02].

Object transaction service forms a part of the rich suite of services (such as Naming, Events, Notification, etc.) that CORBA incorporates, and that vendors provide, in order to free CORBA programmers from having to write such commonly-used functionality themselves.

Object transaction service essentially specifies interfaces for synchronizing a transaction across the elements of a distributed client-server application. A transaction satisfies the four so-called ACID properties: Atomicity, i.e., transactions executes completely or not at all; Consistency, i.e., transactions are a correct transformation of state; Isolation, i.e., even though transactions execute concurrently, it appears for each transaction, T , that other transactions execute either before T , or after T , but not both; and Durability, i.e., modifications performed by completed transactions survive failures. The details of ACID properties are described in next section.

3-4-1 Atomic Transaction

An atomic transaction guarantees that, despite failures, either all of the work conducted within its scope will be performed or it will all be undone. Atomic transactions have the well known ACID properties of Atomicity, Consistency, Isolation and Durability.

Atomicity property ensures that a computation will either be terminated normally (committed), producing the intended results (that is, intended state changes to the objects involved) or aborted producing no results (no state changes to the objects). This atomicity property may be obtained by the appropriate use of backward error recovery, which can be invoked whenever a failure occurs that can not be masked. Typical failures causing a computation to be aborted include node crashes and communication failures such as the continued loss of messages (transaction executes completely or not at all)

Consistency property takes the system from one consistent state to another consistent state. It is assumed that, in the absence of failures and concurrency, the invocation of an operation produces consistent (class specific) state changes to the object. Transactions then ensure that only consistent state changes to objects take place despite concurrent access and any failures.

A transaction is isolated from other transactions, in the sense that each transaction behaves as if it were operating alone with all resources to itself. In particular, each transaction will “see” only consistent data in the underlying data sources [Welkum'02].

The consistency property goes hand in hand with the isolation property that ensures freedom from interference: each transaction accesses shared objects without interfering with other transactions. In other words, the effect of concurrently executing transactions can be shown to be equivalent to some serial order of execution. Some form of concurrency control policy, such as that enforced by two-phase locking [Bernstein'87], is required to ensure isolation and consistency properties of transactions.

In durability property, when the application program is notified that a transaction has been successfully completed (when the Commit transaction call is successfully returned) all updates that the transaction has made in the underlying data servers are guaranteed to survive subsequent software or hardware failure [Welkum'02]. In other words, it is reasonable to assume that once a transaction terminates normally, the results produced are not destroyed by subsequent node crashes. This is ensured by

the durability property, which requires that any committed state changes (i.e., new states of objects modified in the transaction) are recorded on stable (crash-proof) storage. Thus, update of committed transactions are durable (until another transaction later modifies the same data items) in that they persist even across failures of the affected data server(s) [Welkum'02]. A (two phase) commit protocol is required during the termination of a transaction to ensure that either all the objects updated within the transaction have their new states recorded on stable storage, or, if the transaction aborts, no updates get recorded. Atomic transactions can also be nested; the effects of a nested transaction are provisional upon the commit/abort of the outermost (top-level) atomic transaction.

3-5 Group Communication

Group communication is a powerful abstraction that can be used whenever groups of distributed processes cooperate for the execution of a given task such as committing a distributed data base transaction, or to achieve fault-tolerance or better performance (by replication). In group communication, processes usually communicate in a group basis where a message is sent to a group of processes, rather than just to one process, which is the case in point-to-point communication. With a group is usually associated a name to which application processes will refer, making transparent the location of the distributed processes forming the group. Due to the uncertainties inherent to distributed systems (emerging from communication or process failures), group communication protocols have to face situations where, for instance, a sender process fails when a multicast is underway or where messages arrive in an inconsistent order at different destination processes. On the other hand, distributed applications usually require that processes forming a group “see” events such as processes failures and message delivery in a mutually consistent way. For example, active replication (will be explain later in this chapter) requires that messages are delivered in the same order at all replicas and that failures are handled in a mutual consistent manner among operational replicas.

Further complications will arise when groups overlap (i.e. a process is allowed to belong to distinct groups). Messages exchanged by distributed processes can be partially ordered according to their causal origin.

3-5-1 Properties of a Group Communication Service

The term “group communications” infers the collaboration of entities/objects to perform tasks via messages directed at multiple recipients (group of entities/objects), rather than at a singleton. This type of message passing, one-to-many, is commonly termed a multicast.

In addition to providing a multicast mechanism systems that depend on group communications may also require quite sophisticated protocols to manage message delivery. For example, messages to be delivered at each member of a group in the same order. Furthermore, groups may be dynamic; members may leave and join a group during the lifetime of a group. To enable a multicast to be directed at actual members of a group (not including departed members), a mechanism is required which ensures that all members of a group have a mutually consistent view of group membership. A mechanism of this type is usually called a group membership service.

To summarize; A service that provides developers with mechanisms that support the integration of group communications into a distributed system should consist of the following:

- A multicast mechanism.
- Protocols for managing message delivery, with certain ordering and reliability properties.
- A group membership service.

3-5-2 The Multicast Mechanism

To allow a multicast communication each individual member of a group must realise the group membership. This is achieved by allowing each member to maintain a group view. By maintaining a group view a member may identify the addresses of each group member to which messages may be sent. Each entry in a group view should be an addressable location suitable for enabling the sending of messages to each group member. When a member wishes to multicast a message, the message is sent to every member that appears in the group view.

A desirable property of a multicast mechanism is that a given multicast be reliable (failure atomic): if a member crashes while multicasting a message, either all or none of the functioning members deliver the message. Consider the interaction of a client and an active replica group.

Multicasts that are not failure atomic may cause problems in maintaining consistency of state between the individual replicas; one replica fails to receive a state-modifying client request but continues to receive and respond to other client requests.

It is sometimes desirable for entities to simultaneously participate in multiple groups. This is certainly true of video conferencing, where users may participate in more than one conference at a time. When an entity belongs to multiple groups a group view for each group must be maintained by the entity. This will enable multicasts to be directed to specific groups.

3-5-3 Messages Ordering

When dealing with a single entity events occur sequentially, each event resulting from some action carried out by the entity. These events are naturally ordered by the sequence in which they happen. A system model may be based on a group of these single entities. Each entity has the ability to send and receive messages to and from other members of the group. The ordering of events in a group is based on two assumptions:

1. The sending of a message m occurs before the receiving of m .
2. If two events occur at the same member then they retain their natural ordering in relation to each other.

A partial ordering of events for distributed systems has been established based on message passing and the above two assumptions. The notion of "happens before" (\rightarrow) is used to indicate partial ordering. The following ordering properties may be derived from previous observations:

- If the event X occurs before the event Y at the same member then $X \rightarrow Y$.
- If the event A is the sending of a message m , and the event B is the receiving of the message m then $A \rightarrow B$.
- If $C \rightarrow D$ and $D \rightarrow E$, then $C \rightarrow E$.

It is possible to state that if $A \rightarrow B$ then A may have caused B . When no causal relationship exists between two events then the ordering between them is arbitrary and two such events may be considered concurrent.

A protocol that introduces ordering highlights a difference between the receiving of a message and the delivery of a message:

- A sends the message m , B receives message m , B delivers m .

Only after a message is delivered may it be accepted by a member for processing. When ordering is relevant a protocol may block the delivery of a message until such a time when ordering requirements are fulfilled. Under certain circumstances a message may never be delivered and so discarded by an order preserving protocol. Following are more detailed descriptions relating to different types of ordering that are common in the support of group communications:

- **Causal Ordering**

The rules regarding the causal ordering of deliverable messages by a protocol may be derived from the assumption made about causal ordering in an event driven system. As a protocol may concern itself only with the sending and delivery of messages to retain causality it is necessary block the delivery of a message m until all messages that may have caused m have been delivered.

To illustrate the need of causal order preserving delivery, consider for example, a computer based conferencing application where users may simultaneously participate in different conversations (or groups). Suppose a given multi-group user generates a message m' in a group B as a consequence of a message m delivered to the same user in group A . Other multi-group users participating simultaneously in groups A and B ,

would then require that m' be delivered only after m has been delivered (otherwise, m' may make no sense). Since message m potentially caused message m' , we say that they are causally related. For correct delivery of messages m and m' , a protocol is required which delivers messages respecting their causal origin or in causal order.

• **Total Ordering**

There are situations in group communications where the delivery of messages to each member should occur in the same order (and preserve causality). Protocols that achieve this are known as total order protocols. Protocols that enforce total ordered message delivery must block the delivery of a message until all members of a group mutually agree on the order in which such a message is to be delivered. Total ordering that lacks causal preserving qualities is commonly termed identical ordering [Macedo'95].

There are two distinct types of protocol for achieving total ordering:

- **Asymmetric** - A single member of the group is responsible for determining the order of delivery.
- **Symmetric** - All members of the group share the responsibility for determining ordering.

In an asymmetric protocol the member responsible for ordering is commonly termed the sequencer. Each member of a group may unicast the message they wished distributed throughout the membership of the group to the sequencer. The sequencer is responsible for multicasting such messages to all members of the group. In a symmetric protocol, members simply multicast their messages to the whole membership of the group.

The asymmetric protocol tends to favor groups that only have a subset of the membership regularly multicasting. Such scenarios arise when clients request a service from a group (as in highly available applications). In a symmetric protocol, ensuring client requests may be suitably ordered for delivery requires all members to

participate in a message passing round. This message passing round has to be prompted (on the receiving of a client request) and such message passing may solely exist to order client requests (no computational value to the application). However, if an asymmetric protocol is used members receive client requests already totally ordered and may be delivered without the need for further message passing. When all members frequently multicast in a group the symmetric protocol is favored. Such scenarios arise in Groupware applications. In Groupware applications members wish to share information (such as a video image in teleconferencing). Members tend to multicast in an asynchronous fashion (do not wait for reply). As every member frequently multicasts, message passing rounds will be completed. There is no need to prompt message passing solely for the purpose of message ordering. Furthermore, the redirection of messages through a sequencer (as in the asymmetric approach) adds unnecessary message latency in Groupware applications.

3-5-4 Group Membership

It is necessary for all members of a group to have a mutually consistent view of the membership of the group (group views of individual members of a group remain mutually consistent). When members do not agree on group membership actions within a group may lead to inconsistencies between the functionality of the group and the group's expected behavior as identified in a specification. For example, consider a simple system that consists of a group of three members (*A*, *B*, and *C*) that service client requests. The group's specification is identified by the following five points:

1. *B* and *C* are backup members for *A* (the primary member).
2. Only the primary member may service client requests.
3. When *A* fails *B* should become the primary.
4. When *A* and *B* fail *C* should become the primary.
5. There should always be one, and only one, primary in operation at any one period in time.

An inability to satisfactorily determine mutually consistent group views for each member may result in either one of the following faulty scenarios:

- No primary exists - Assume *A* fails. However, *B* does not register this and still includes *A* in its group view. *B* fails to take up the responsibility of becoming the primary, and as *C* does not assume *B* to have failed does not take on the role of primary.
- Multiple primaries exist - Assume *C* incorrectly suspects *A* and *B* to have failed, reducing its group view to only include itself. This may result in two primaries, *A* and *C*.

As groups are dynamic (members may join or leave a group), a mechanism that enables some form of consensus on group membership is necessary. The difficulties encountered when determining group membership is referred to as the Group Membership Problem (GMP), sometimes referred to as the consensus problem.

To aid in solving GMP a failure detection mechanism is required to indicate the event of member failure to non-faulty members. This will then enable non-faulty members to install a new group view, excluding any failed members. To enable failure detection it is first necessary to establish the correctness of a member.

3-5-5 Highly Available Services

The group communication paradigm allows the provision of high availability through replication in a straightforward way by gathering a set of replicas into a single group. A common method used to increase the availability of a service is to replicate the service over nodes in a network. A service that is replicated is commonly termed a replica group. The aim of a replica group is to allow the failure of a number nodes, parts of the network, or a number of objects that provide the service to be tolerated before the service becomes unavailable. There are two main techniques available for providing service replication. Active and passive (primary-backup) replication, in a brief definition, passive replication of process is based on one process acting as a leader and updating other process after every operation has been informed. When the leader dies, one of the processes take over. However active replication is when all processes provide service in steps and they are up-to-date with its operation, so a

failing leader process does not need to send its own state to other processes. The details of these two techniques are as the following:

*** Active / Passive (Primary-backup) Replication**

In active replication client requests are directed at each replica. Each replica then attempts to process the request and may reply to client requests. The active replication of an object requires two conditions to be met:

- i.* Agreement - All the non-faulty replicas of an object receive identical input messages.
- ii.* Order - All the non-faulty replicas process messages in an identical order.

Therefore, if all the non-faulty replicas have identical initial states then identical output messages in an identical order will be produced by them (assuming that an action performed by an object on a selected message is deterministic). To ensure the message ordering requirement is satisfied suitable protocols must be available that can provide guaranteed identical ordered message delivery within the replica group.

When member failures occur clients of an active replica group may not suffer from gaps in service. The only time an actively replicated group cannot service requests is when all replicas have failed, or the replica group is unreachable by a client due to network failures.

Detecting member failures is required to satisfy the agreement condition. Inconsistent views of group membership may lead to the failure of client requests reaching non-faulty replicas and/or the inclusion by some members of faulty members in their group views.

Passive replication requires only one member of the replica group, the primary (sometimes referred to as the coordinator), to receive, process, and reply to client requests. To ensure that members of the replica group stay mutually consistent the

primary must send a checkpoint of its state to the passive group members, usually when the state of the primary has changed.

In the event of the primary failing the remaining members use a protocol to elect a new primary, which then takes over the duties of the failed primary.

As opposed to active replication, it is not necessary for computations performed by the replicated objects to be deterministic: state is imposed upon the passive members of the group by the primary guaranteeing that all members of the group will remain mutually consistent.

According to the concept of replications which have been described, the following contest will be raised:

With active replication, all of the replicas of the object play the same role: every active replica receives each request, processes it, updates its state, and sends a response back to the client. Because the client's invocations are always sent to, and processed by, every server replica, the failure of any of the server replicas can be made transparent to the client. With passive (primary-backup) replication, one of the server replicas is designated as the primary, while all the other entire replicas serve as backups. A client typically sends its request only to the primary, which executes the request, updates its own state, updates the states of the backups, and sends the response to the client. The periodic state updates from the primary to the backups serve to synchronize the states of all of the server replicas at specific points in their execution. Replication implements *roll-forward* recovery mechanisms that promote liveness by continuing processing where it had been left at the time of the failure. In active replication, in the event of a fault (one of the active replicas crashes), the other replicas continue processing the current request, regardless, thereby implicitly implementing a roll-forward mechanism. In passive (primary-backup) replication, in the event of a fault (the primary replica crashes), one of the backup replicas takes over as the new primary and re-processes any requests that the previous primary was performing before it failed. If a backup replica crashes, then, there is no loss in processing. Thus, the roll-forward mechanism is explicitly implemented in the re-election of a new primary replica, and the re-processing of requests by the new

primary. Consistency is maintained for both active and primary-backup replication by guaranteeing that partial request execution will not harm since the request will be eventually completed (by “rolling forward”).

An example of a bank account service made highly available via replication (active / passive) may be used to demonstrate the benefit a group communications service may bring to a fault tolerant application:

Copies of a bank account B reside at three of a bank's branches. This degree of replication allows routine audit checks of an account to be carried out at a branch (making the inspected account unavailable for a short time) while still allowing access to the other two copies of the account. The account is accessible via an Automatic Teller Machine (ATM). Each branch has an ATM ($A1$, $A2$ and $A3$). Each copy of the bank account should present the same balance whenever queried via an ATM. When a transaction is requested at an ATM, information relating to the transaction request are formulated into a single message and sent to all copies of the bank account. Each account then acts on the request and replies with a suitable answer. The requesting ATM takes the first answer only and discards the rest. Whenever an account balance falls below zero, bank charges are incurred.

Let us concentrate on the functioning of a single account held jointly by a husband and wife. The husband deposits \$50 (generating a message $M1$) at $A1$ and then withdraws \$20 (generating a message $M2$), again via $A1$. We may identify $M1$ and $M2$ as being causally related and expect $M1$ to be received by all copies of the account before $M2$ is received. If this is not so, then some accounts may actually become overdrawn (there will be a time when some accounts would show a balance of \$20) causing bank charges to be incurred.

We now extend our example and assume the wife is requesting a withdrawal of \$20 ($M3$) at $A2$ at the same time the husband is at $A1$. As there is no causal relationship between $M3$ and the other two messages ($M1$ and $M2$) $M3$ may be received at any time by the replica accounts. If $M3$ arrives before $M1$ then bank charges will be incurred, if $M3$ arrives after $M1$ bank charges will not be incurred. Therefore, we have a scenario, where some accounts may incur charges while others do not.

To overcome this inconsistency, we must ensure that messages arrive at the same order at each account. This type of ordering is commonly termed total ordering (identical ordering while preserving causality).

In the same manner as our previous groupware example identified the three requirements that a group communication service aims to satisfy, so the observation is repeated for the highly available account example (for the sake of completeness passive replication is also mentioned):

- An ATM is required to send a single copy of a message to multiple bank accounts: A multicast mechanism is required to allow an ATM to send a single message to all bank accounts simultaneously.
- Prevent the balance of the replica accounts from deviating, the consequences of which could result in users been presented with bank charges: Protocols that preserve the total (while still preserving causal) ordering of messages are required.
- Allowing the audit of a replica account without inhibiting the operation of the other accounts: A group membership service may identify when an audit is taking place (remove replica) or when an audit is completed (add replica) during the lifetime of a group. In passive replication there is still a need to determine if a member has failed/departed to ensure suitable passive members exist or a failed primary may be replaced.

3-6 Replication / Transaction - Based Reliability and Comparison

A widely used computational model for constructing fault-tolerant distributed applications employs atomic transactions for controlling operations on persistent objects. There has been considerable work on data replication techniques for increasing the availability of persistent data is manipulated under the control of transactions. Process group with ordered group communications has also emerged as a model for building available distributed applications. High service availability can be

achieved by replicating the service state on multiple processes managed by a group communication infrastructure. These two models are often seen as rivals [Little'99].

Replication is intended at protecting computational resources through the use of redundancy: if a processor fails, then another processor can take over the processing of the failed processor.

Regarding the two best-known replication styles (active and passive) a replicated object is often represented by an object group, with the replicas of the object forming the members of the group. The object group membership may be static or dynamic. Static membership implies that the number, and the identity, of the replicas do not change over the lifetime of the replicated object; on the other hand, dynamic replication allows replicas to be added or removed at run-time.

Unlike replication, transaction processing systems essentially aim at protecting data. then a failure occurs in the context of a transaction, the objects involved in the transaction are reverted to their state just prior to the beginning of the transaction. All of the state updates and all of the processing that occurred during the transaction are discarded, often with no trace left in the system. Some systems support nested transactions, where a new (child) transaction can be initiated within the scope of an existing (parent) transaction. If the nested (child) transaction fails, the enclosing (parent) transaction needs not automatically roll back; the application can attempt to correct the problem, and subsequently retry the nested transaction. However, if the enclosing transaction encounters a fault, then all the nested transactions roll back, along with the enclosing transaction.

Transactions use roll-back recovery mechanisms that guarantee consistency by undoing partial request processing. Data is protected from the undesirable side-effects of failures, but computational resources may become unavailable for arbitrary durations. Transactions are thus an effective mechanism for preserving consistency, but not for achieving high availability, as they sometimes trade liveness for safety.

With roll-forward reliability strategies, invocations are traditionally sent using reliable multicast (also known as reliable group communication), so that all of the replicas of

an object receive every request. This is evident in an active replication configuration, where a client does not need to re-issue the request if one of the active server replicas fails (in fact, the client is typically not even aware of this failure). In a primary-backup setting, when the primary has finished processing a request, it multicasts both the response and a state update to the backups before returning the response to the client. The state update allows the backups to synchronize their state with that of the primary. The response is also cached by the backups for retrieval, should the primary fail. If the primary fails, then a backup assumes the role of the new primary transparently. If the primary fails before returning a response to the client, the client will re-issue the request to one of the backups (now the new primary); if the new primary has a cached response and the last state update of the old primary, it can readily return a response; if it doesn't have the cached response, it will re-process the request.

3-7 Conclusions

This chapter concentrated on a number of existing Middleware technologies for enabling distributed application development in an object-oriented style. The three most popular Middleware technologies (Java-RMI, DCOM, CORBA) were briefly described, followed by a more in-depth description of CORBA. The lack of support for object groups in the CORBA (and also Java-RMI and DCOM) standard was highlighted, followed by descriptions of two types of applications (highly available, Groupware) that may benefit from such support. Services (multicast, message ordering, atomic message delivery, membership) that enable group communications were then described in more detail followed by the three methods (integration, interception, service) that may be used to incorporate such services into CORBA and at the end of the chapter, the comparison of replication and transaction has been described.

As can be seen from this chapter, group communications for Middleware environments has resulted in a number of services that enable an application developer to make use of group communication protocols in their applications.

The centralised and distributed system architecture for Internet auction will be described in chapter 4. Regarding the tools which have been explained in this chapter, the implementing of the centralised and distributed auction system will be discussed and used in chapter 5.

Chapter 4

4-1 Auction House Architecture

Auction house is where buyer and seller should go in order to contact an auction as well as to running the auction, it is also responsible for setting up and guaranteeing various contract that are used to create and manage the auction, ensuring that bidders have sufficient credit limits, enough points if it is necessary regarding sellers' request, certifying that the seller is authorised to sell the item, buyer and seller have not been previously barred from bidding and selling and guaranteeing specific quality of service contracts. The auction house paradigm transfers relatively easily from the physical to the electronic world, and represents a "concrete" entity that users can reason about. From the outside, the auction house essentially represents a "black box"; internally, however, the contracts it enforces, such as security, authentication, and bidder/seller anonymity, help to provide the assurances traders (buyers and sellers) expect from their real-world equivalents. If the auction house allows agents to participate in auctions on behalf of bidders then it will be necessary to ensure that a security sand-box exists for them to reside within.

An auction house may be composed of many physically remote auction rooms that co-operate to provide the abstraction of single centralised auction house, also an auction room itself may be (recursively) composed of several auction rooms and so on; auction rooms will be taken here as indivisible atomic units within an auction house. Regarding the auction rules, the auction room may be owned by different organisations, who have agreed to work together towards the sale of a particular item.

Each auction room has an auctioneer who collects bids submitted in each auction room and determines whether the bidding should continue or be terminated and the auctioneer of one of the rooms is designed as the head or root auctioneer. The root

an auctioneer will determine the auction rules and disseminate them at the start of an auction to the entire house and the seller has the item and the right to sell it. Regarding the auction rules permitting the seller either can actively participate in the auction and modify the ask depending on the demand perceived or asking reserve price which should be met by bidder. We suppose the seller and root auctioneer are in the same auction room.

The bidder in an auction room, which place their bids with auctioneer, is allowed to place a bid that is larger than the current highest bid and the seller is also allowed to have a reserve price to sell. How bid placement happens will depend upon the type of auction (e.g. sealed-bid auction versus open-cry) . A bidder may be required by the auction room to provide proof that he has the required credit limit or enough positive points from previous trading in the auction. Each auction room is free to impose its own constraints on buyer and sellers who use it . Therefore , for example , one auction house may require all bidders to be known , whereas another may allow certain (or all) bidders to remain anonymous. Flexibility in the auction contracts that are imposed by auction rooms may be the deciding factor in how a bidder (and seller) choose an auction room for conducting his trade.

4-2 Basic Auction Unit

For the sake of simplicity, we assume that there is just one seller and single indivisible item is being sold. Assume initially that auction house has only one auction room with whom the seller and the bidders are registered. We select an auction model that treats sellers and buyers symmetrically. This symmetry enable a computational node to play at one level of the tree the role of a seller by dealing with a group of potential buyer as well as to play the role of a potential buyer at the next higher level (this aspect will be discussed later). When a round initiate, the seller quoting an ask price and bidders are invited to place the bid that exceed the quoted ask. (Both seller and buyer should already been registered in the auction house). A bid once placed, can not be withdrawn. A bidder's offer is made known to all other bidders who are encouraged to out-bid that offer before the expiry of a publicly-announced deadline which is determined. The new bid should be greater than the current highest bid and should be placed before the deadline is expired. The seller is aware of every bid placed and

hence of the bidding pattern . There is a restriction that the seller can not decrease his ask during the round , and just seller can have a reserve price when the round has been initiated . Decreasing the ask would mean the current round being abandoned and a fresh round initiated. A bargain round terminates once the seller has quoted his final and after every bidder has been given sufficient time to outbid the highest bid. The highest bid at the end of the round, which is called the final bid, if it is less than the final ask which is requested, the seller can either initiate a new bargain round probably quoting a smaller initiate ask or give up the trade. If the final bid is larger than the final ask, the trade is consummated and if two or more bidders had placed the same final bid, then a single bidder among these finalists is selected through a draw that is statistically fair.

The single room auction model can be realised with the help of two types of nodes. A bidder server (*BS*) and bidder client(s) (*BC*); bidder server is representing an auction room and bidder client is operating as a potential buyer (figure 4-1).

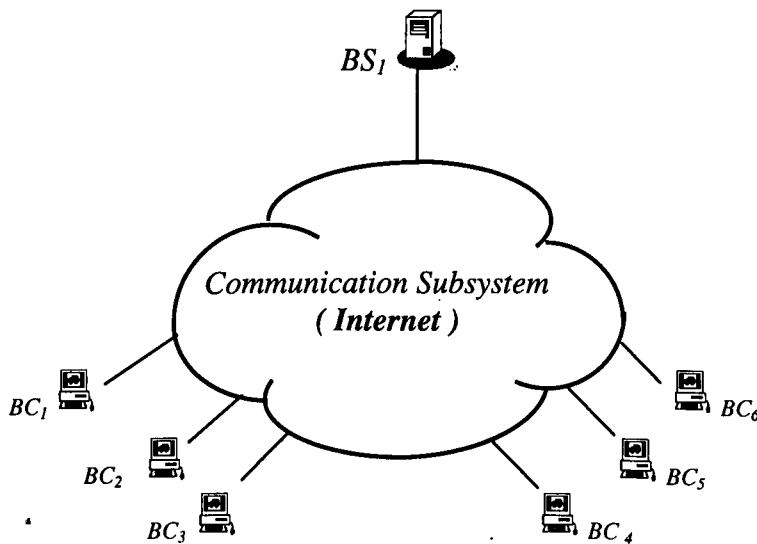


Figure 4-1

The group comprising a bidder server (*BS*) and the bidders client (*BC*) registered with *BS*, will be called a Basic Auction Unit (figure 4-2). Before taking part in the trade, each bidder *BC* must register with the bidder server, and this process deals with

issues relating to bidder/server authentication, exchange of cryptography keys, authorisation on the bidder's spending limits etc.

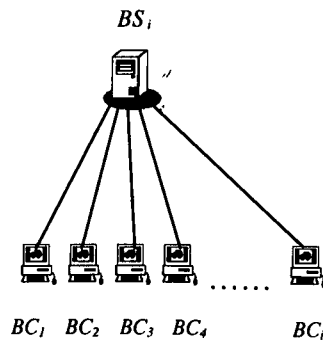


Figure 4-2

A bidder is required to register with any of the bidder servers whose auctioneer is responsible for taking the registered bidders' bids and for providing up-to-date state information about the auction process (such as the highest placed so far, latest ask price, deadline for receiving bids, etc.).

When applying the conventional auction, the clients and the servers act as shown in figures 4-3 and 4-4.

The following operations will be provided by bidder server (BS_i) in the future figures:

(BS_i. get_bid_val(value) : Bidder Client sends a request to Bidder Server for getting value of particular item .

(BS_i. submit_bid(value) : Bidder Client submits a bid to the in Bidder Server

4-2-1 Clients' Procedure:

Figure 4-3 shows the procedure of the conventional clients.

```

while ( auction_held ) // Whenever auction is being held and clients attend in auction
{
    set_timeout;
    BSi. get_bid_val(max_val ); // Client sends request to bidder server ( BSi )
                                // and blocks ( waiting for results )

    < on_time_out : abort the action > ; // Time_out has happened (connection
                                        // failed or bidder server problem) in both cases
                                        // Client's request has not been succeeded,
                                        // display appropriate message on client's screen,
                                        // exit from procedure and should try later

    bid_val = clientdetermine_bid_val ( max_val ); // Client determines new bid_val
    set_timeout;

```

```

BSi. submit_bid(bid_val) ; // Client sends new bid value to bidder sever (BSi) to submit the
bid_val
display ( result ) ; // Display response from bidder server (BSi) : “ Accepted “ or “ Rejected “

< on_time_out : abort the action > ; // Timeout has happened so Client’s new bid value has
not been accepted , exit from procedure and should
try later

} // End of “ while (auction_held) “

```

Figure 4-3

Client repeats the following processes while the corresponding auction is held. First, an optional timeout should be set to trigger at the particular time, and client requests a maximal value information from the bidder server and receive it in appropriate time (otherwise client’s request has been aborted). Maximal value (*max_val* in figure 4-3) is obtained from the bidder server and according to the obtained maximal value, the client determines bidding value (*bid_val* in figure 4-3). After that, the client set timeout and sends the determined bid to the bidder server, and an *result* (“Accepted” or “Rejected” , see figure 4-4) is obtained from bidder sever (within timeout). If *action_result* is true , bidding process has been done successfully (confirmation message will be sent to client through displaying on bidder client’s screen). Otherwise client needs to repeat submitting the bid. (Client biding will be aborted if time out has happened).

4-2-2 Servers’ Procedure:

The bidder server sets an initial value (*init_val*) and result message as a global boolean variable. The bidder server repeats the following process (figure 4-4) while the corresponding auction is held.

```

// Initialization
val = init_val;
action_result:String ;

while (auction_held)
{
// Bidder server replies, whenever receives message from clients
get_operation_request (); // Bidder server receives a message from client
case request_type of :
get_bid_val ( max_val ) ; // Bidder server has been asked by client for existing
max_val

```

```

    {
        return (max_val); // Bidder server replies existing max_value to corresponding client
    }
submit_bid ( bid_val ); // Bidder server receives new bid value from client
    {
        bid_val = clientbid(bid_val); // bid value from client
        if (bid_val > val)
            {
                val = bid_val; // Bidder server updates current value with new
                               bid_val which has been received from client if it is higher
                               than the current bid_val

                action_result = " Accepted ";
                return (action_result); // Client's bidding has been submitted by bidder server
                                       and sends " Accepted " to the corresponding client
            } // End of " if (bid_val > val) "
        else
            {
                action_result = " Rejected ";
                return (action_result); // Client's bidding has not been accepted due to new
                                       bid_val was not higher than current bid_val ,
                                       therefore " Rejected " will be returned to the corresponding client
            }
    other ; // The receiving message is not about existing max_val nor new bid_val
    {
        ignore the message; // Ignoring the message if bidder server receives
                             messages neither about existing max_val nor new bid_val
    }

} // End of " while (auction_held) "

```

Figure 4-4

First, if a request for the maximal value information comes from a client, the bidder server returns the current value to the client , (*max_val*). If bidder server receives a submission bid message , a bidding value (*bid_val*) is obtained and compared with the current value information. If the bidding value is higher than the current one, the out of bid message is sent to the client whose bid is now out of bid (most of the auction servers provide automated "outbid notification " email messages to let bidders know instantly when they are no longer the high bidder in an auction) and the current one is updated and the " *Accepted* " message is returned to the client through the *action_result*; otherwise, the server returns " *Rejected* " message to the corresponding client. If bidder server's receiving message is not about existing *max_val* or *new bid_value*, therefore ignoring the message .

4-2-3 Basic Auction Unit Connection

Basic auction units can be connected to each other, if they are close, through their respected bidder server directly. In this case, the normal local area network would be fine to communicate between bidder servers (BS) and for bidder client (BC) to take part in an auction through bidder clients (BC). Figure 4-5 shows the detail.

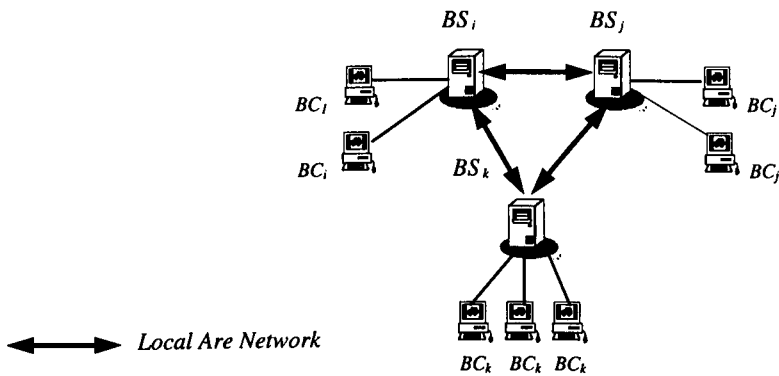


Figure 4-5

We now extend the above model to a distributed auction model to support a large number of bidders, geographically wide apart, to take part in an auction. A lot of methods are existing to connect basic auction units to each other. We regard the distributed auction system to be made up of many servers connected to each other via the privately-owned, high-bandwidth network or through the Internet. Each basic auction unit connect together as depict in figures 4-6 and 4-7.

In figure 4-6, three different groups of basic auction units, are connected together through the privately-owned high band network. As shown in this figure, if we increase the number of basic auction units, it will cause the main problem to connect all bidder server (and bidder client as well) together through the basic auction unit, and as a result, achieving scalability will be difficult and probably impossible. In addition as this choice will need more connect and communication, so as a result will cause more cost.

Basically, the auction should be scalable so to resolve above problem, basic auction units communicate to each other through the Internet (See figure 4-7). In this regard,

achieving scalability and responsiveness will be possible if suitable structure is considered.

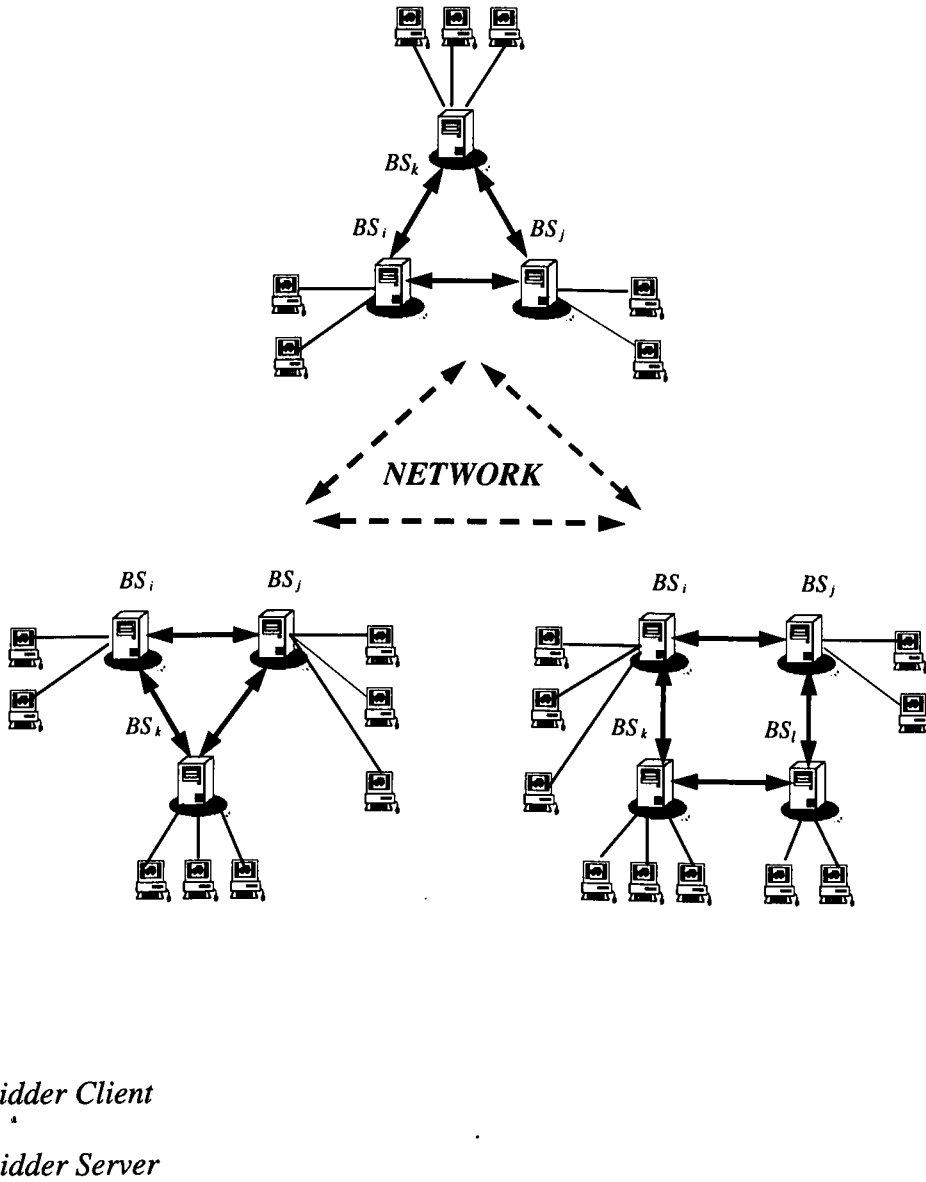


Figure 4-6

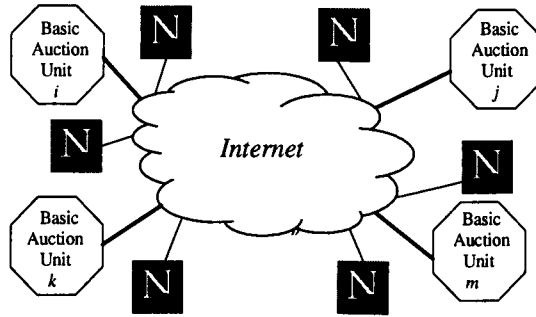


Figure 4-7

The combination of figure 4-6 and 4-7, depict in figure 4-8 .As this figure shows, basic auction units are connected together through the privately-owned high band network and build the cluster of auction. These clusters of auction are communicated to each other through the Internet.

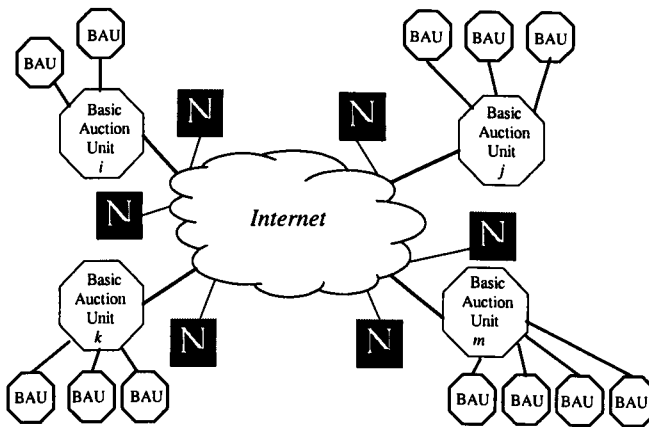


Figure 4-8

4-3 Pure Hierarchical Architecture

It is well known that a multilevel and tree based architecture is a good preparation for scalability requirements. Therefore developing a system, which allows a user to bid at any logically nearest server (latency time), is our principal way of achieving scalability as the total load is shared amongst many servers. It means interaction among a very large number of customers and supplier of goods, regardless of the number of those users and their geographical location. This requires the service to satisfy users requirements in the presence of large number of buyers and sellers.

The structured is arranged in a tree, rooted on main auction server. This structure is made up of several levels of basic auction units connected to each other via the internet /intranet or privately-owned, high-bandwidth network. Figure 4-9 shows basic auction units arranged in multilevel (as a tree), with the root being main auction server (MAS). In this regards, basic auction units can be combined hierarchically to support a finitely large number of bidders, geographically wide apart, to take in an auction.

Recall that basic auction units can directly communicate with each other as has been shown in figure 4-5 and 4-6 and this tree structure is a logical one imposed in an attempt to make the inter- basic auction units communication scalable; also, that each basic auction units caters for a local set of clients and has its own (local) bidders registered directly with it.

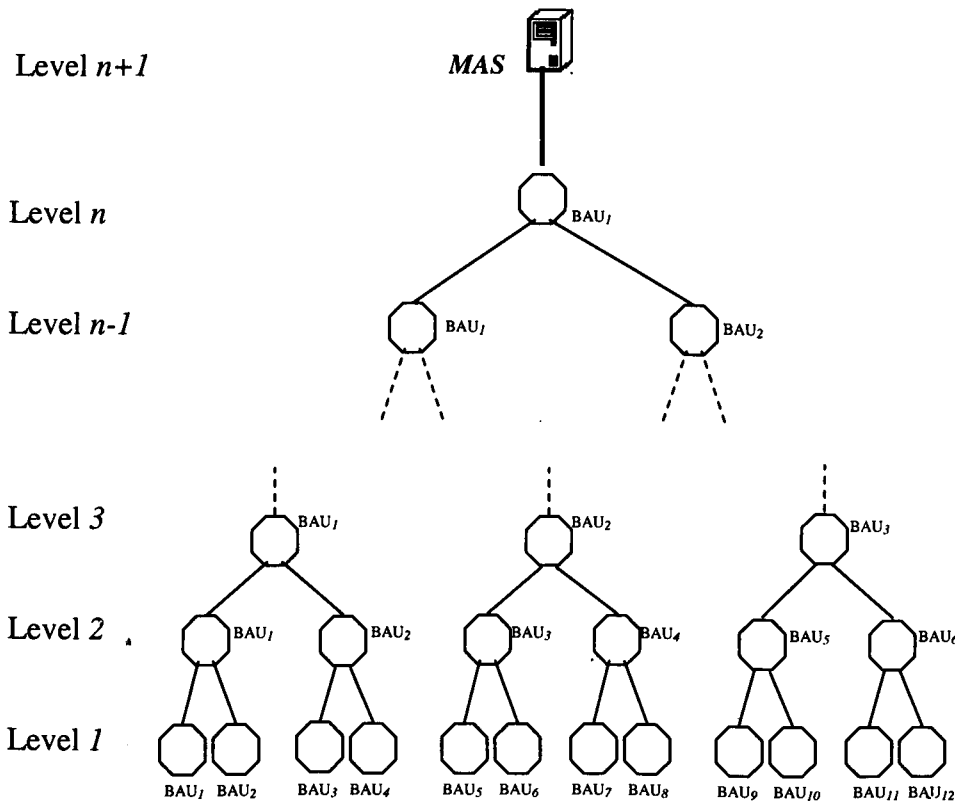


Figure 4-9

Main auction server (MAS), which is allocated in the root if auction servers are organised as a hierarchical architecture and use tree structure to connect together, is playing the role of the seller, and also is at the top-most level of the tree. A basic auction unit is termed the parent of all those basic auction unit that are directly connected to it and are one level below; the lower level basic auction unit are termed the child basic auction unit of the parent. A basic auction unit that has no child is called a leaf basic auction unit. We do not require the tree to be a balanced one (though such a tree would improve the communication efficiency) or a binary one as shown in the figure. What we do require is that the root bidder server be connected to every other bidder server either directly or via sequencing parent basic auction unit and that every non-root basic auction unit have only one parent.

Refereeing hierarchical architecture and tree structure, which has been described, and regarding fundamental fairness property of scalable auction to capable of providing its end user with satisfactory Quality of Service (QoS), regardless of the number of those users and their geographical distance, we therefore investigate ways of enabling widely distributed, arbitrary large number of auction servers to cooperate in conducting an auction. In this regard, disseminating latest updated information, which are called episode messages (will be referred to later), will be necessary. Therefore servers are partitioned into *Multicast groups*, to communicate together in order to disseminate information to each other and as a result users will be allowed to bid at any one of the auction bidder servers. In this way scalability and fairness properties of an auction will be preserved.

A group consist of one parent and all its children are in the multicast group. As shows in figure 4-9 BAU's will be divided into numbers of multicast groups: e.g. {BAU₁ and BAU₂ in level 1 ith BAU₁ in level 2}, {BAU₃ and BAU₄ in level 1 with BAU₂, in level 2}, {BAU₁ and BAU₂ in level 2 with BAU₁, in level 3} and ... {BAU₁ and BAU₂ in level n-1 with BAU₁, in level n}. Within a multicast group, servers know each other's identifier and periodically multicast the latest auction information it has received so far to every other server in the respected group. These multicast messages are called episode messages, as their contents are used by each server to form the history of client requests accepted (so far) in the global system. The episode messages generated by a

given server obey the following rule: every local client request accepted is referred to in one of the episode messages, and no two-episode messages refer to the same client request. This is necessary to ensure that the global history constructed by each server represents any given bid exactly once.

Every BAU_i is in at least one group and a parent server, except the root (BAU_1 in level n in figure 4-9) is present in two groups. For a parent server (e.g. BAU_i in level 2), the group that contains its children is called its down-tree group and denoted as G_d ; e.g., G_d of BAU_1 in level 2 is $\{BAU_1$ and BAU_2 in level 1 and BAU_1 in level 2 $\}$ or G_d of BAU_2 in level 2 is $\{BAU_3$ and BAU_4 in level 1 and BAU_2 in level 2 $\}$. (Alternatively for a non-root server, the group that contains its parent is called its up-tree group and denoted as G_u ; e.g. G_u of BAU_1 in level 1 is $\{BAU_1$ and BAU_2 in level 1 and BAU_1 in level 2 $\}$ or G_u of BAU_1 in level 2 is $\{BAU_1$ and BAU_2 in level 2 and BAU_1 in level 3 $\}$).

Partitioning the servers into multicast groups based on a tree structure facilitates dissemination of episode messages, which has already been explained in above, in the following recursive manner. A non-root parent server periodically (in appropriate time , e.g. whenever receives new bid) aggregates its own episode message with messages received from its children during the past period and forwards (possibly multicasts regarding the architecture) the aggregated episode message in its G_u . Thus, in its up-tree group, it represents the bids received by every server of the sub-tree rooted on itself. The downward propagation of episode messages also works in the same way but in the downward direction: each non-root parent server periodically (in appropriate time which will be defined by the main auction server with considering bidder servers' conditions in each level of hierarchy architecture which will be explained in figure 4-13) aggregates its own episode message with the messages received from the parent of its G_u during the past period, and multicasts the aggregated episode message in its G_d ; the root server periodically multicasts only its own episode message in its G_d . Recall that the formation and aggregation of episode messages are done in such a way that any given client request (sent to any server in the global system) is represented exactly once in the global history computed by every server.

In this hierarchy architecture that has been introduced, scalability, which is necessary for auction, will be met.

4-4 Implementation Framework

To explain our implementation clearly, we simplify the figure 4-9 to figure 4-10.

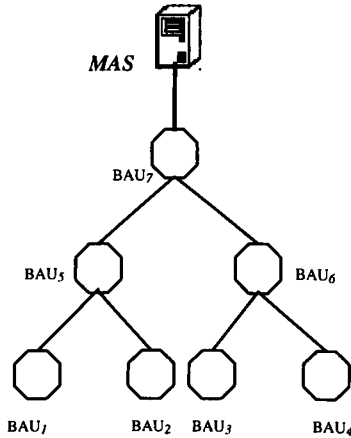


Figure 4-10

In figure 4-10, seven *BAUs* are arranged in a tree, with the root being server *BAU7* which connect to *MAS* with which the seller is assumed to be registered.

The main idea for processing the request in any of the bidder servers is that the client will send its requests through bidder clients to its local server (bidder server in the *BAU*) for processing. Bidder servers periodically report the requests they have received so far to every upward bidder servers that are directly connected to them in the tree structure in the distributed auction server.

The seven bidder servers in the *BAUs* in figure 4-10 can be configured into three multicast groups as shown in figure 4-11. In this figure three multicast groups has been shown: $G1: \{ BS_1, BS_5, BS_2 \}$, $G2: \{ BS_5, BS_6, BS_7 \}$, $G3: \{ BS_6, BS_3, BS_4 \}$. The details of how multicast episode messages between the groups and report the messages to bidder servers, will describe later as shown in figure 4-11.

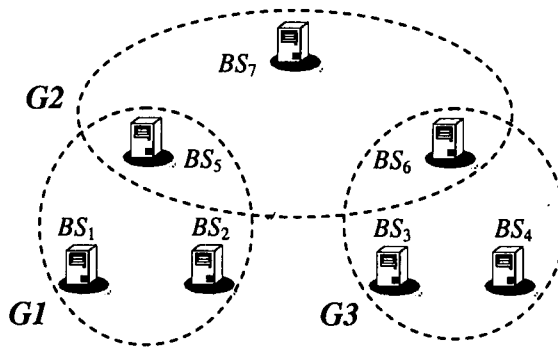


Figure 4-11

As the figure 4-11 shows, every bidder server is in at least one group and the parent server, except the root bidder server (BS_7), is present in two groups.

Regarding down-tree group (G_d) partitioning the servers into multicast groups based on tree structure facilitates dissemination of episode messages in the following recursive manner. A non-root parent bidder server periodically aggregates its own episode message with the messages received from other members of its G_u during the past period, and multicasts the aggregated episode message in its G_d (e.g. BS_5 to BS_1 and BS_2 as G_d of BS_5 is $G1$ which include BS_5 , BS_1 , BS_2); the root bidder server periodically multicasts only its own episode message in its G_d (BS_7 to BS_5 and BS_6). The upward propagation of episode messages also works almost in the same way but in the upward direction. A non-root parent server periodically aggregates its own episode message with messages received from its children during the past period, and forwards the aggregated episode message in its parent (e.g. BS_1 and BS_2 to BS_5 and BS_5 to BS_7 , also the same scenario exists for BS_3 and BS_4 to BS_6 and BS_6 to BS_7). Thus, in its up-tree group, it represents the bids received by every bidder server of the sub-tree rooted on itself. The details of algorithm for multicasting in down tree group as well as forwarding in its parents will be explained in figure 4-14.

The server at level $n+1$ is designed as the head or main auction server (MAS in figure 4-9) which is playing the role of the seller and has the responsibility of determining the auction rule, disseminating latest episode messages to other bidder servers through down-tree group which are connected together based on tree structure, selecting winner and so on regarding the bids and other information it has received. In fact, parent of each bidder server acts as the head bidder server for its child bidder servers (e.g. BS_5 head bidder server for BS_1 and BS_2 , also BS_6 head bidder server for BS_3

and BS_4) while the bidder server which is allocated in level $n+1$ in the root of the tree is designed as the head or main auction server in distributed auction system. Therefore, episode messages normally move upward to the main auction server and move downward from main auction server to the all bidder servers and as a result, the bidders who registered with each of the bidder servers can be informed by the results of their bid and then bidder has been given sufficient time to outbid the highest bid in the auction.

Figure 4-12 shows the details of a sample of connection between bidder clients and bidder servers in BAU_1 , BAU_2 and BAU_5 as has been shown in GI in figure 4-11.

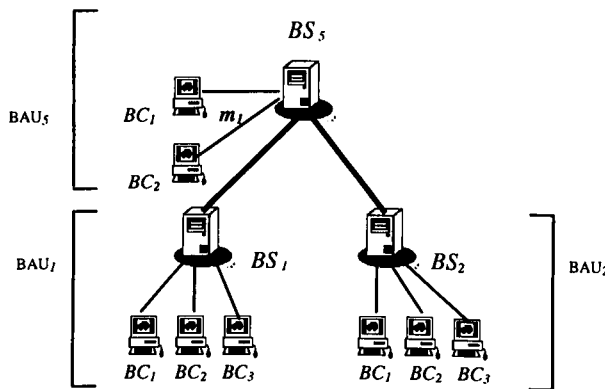


Figure 4-12

The procedure of bidder clients is the same as procedure of clients which has been explained in figure 4-3. The only difference between the conventional clients in figure 4-3 and bidder clients in figure 4-12 is that each bidder client sends the determined bid to the *Client's Selection Bidder Server* which is allocated in the nearest location to client or client might wish to send its bid to the specific bidder server which is placed in the whole distributed auction system.

The procedure of bidder servers and how the necessary aggregate information is passing between $BAUs$ will be described in the next section.

4-4-1 Bidder Servers

There are four tasks which are executing concurrently in each bidder server as shown in figure 4-13 . *Task 1* and *Task 2* handle messages from bidder clients and other bidder servers respectively, *Task 3* handle messages to BS_i 's parent bidder server and *Task 4* handle messages to BS_i 's child bidder servers. (*Task 4* will not be executed for bidder servers in level 1, see figure 4-9, since there is no child bidder server in this level. Also, *Task 3* will not be executed for the main auction server since there is no parent bidder server for this sever).

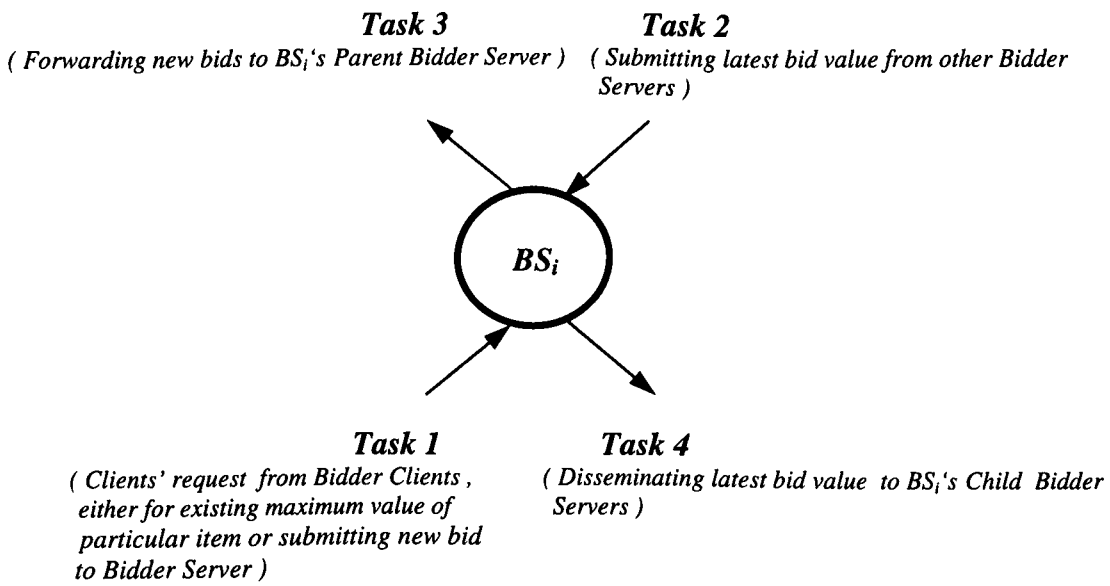


Figure 4-13

Suppose bidder client (BC_1) sends a bid (say m_1) to the bidder server (BS_5) (see figure 4-12). The following procedure will be held in bidder servers (e.g. BS_3) :

4-4-1-1 Bidder Servers' Procedure

```
// Initialization
val = init_val;
action_result:String;

// As long as auction is running, the following procedure which is includes " 4 " concurrent tasks will
be held on the Auction Bidder Servers :
while (auction_held)
{
    cobegin /* Start of Task 1 */
```



```

/* Task 1 : Executed by the Bidder Servers , whenever they get request either for existing
max-val or submission new bid value
// Bidder Server replies, whenever it receives message from clients

// Executing “ Servers’ Procedure” in figure 4-4 ;

coend /* End of Task 1 */

cobegin /* Start of Task 2 */

/* Task 2 : Executed by the Bidder Servers , whenever they receive submit message from
other bidder servers , either its parent or children */

get_operation_request ( ) ; // Bidder server receives a message
case request_type of :

submit_bid (bid_val) ; // Receives submit message from other bidder server
{
    bid_val = BidderServer (bid_val) ; // Bid value from bidder server
    if (bid_val > val)
    {
        val = bid_val ; // Auction Bidder Server updates current value with new
bid_val if it is higher than the current bid_val
    } // End of “ if (bid_val > val) “
    action_result = “ Accepted ” ;
    return (action_result) ; // New bidding has been accepted (Auction Bidder
Server sends “ Accepted ” to the corresponding sender)
} // End of “submit_bid (bid_val)”

other ; // The receiving message is not about submitting bid value
{
    ignore the message ; // Ignoring the message
}
coend ; /* End of Task 2 */

cobegin /* Start of Task 3 */

/* Task 3 : Executed by the Bidder Server as long as parent bidder servers are existing in
order to forward new bid to the next parent bidder servers */
while ( Parent_BidderServer ) // While parent bidder servers are existing and time is
appropriate, forwarding the new value of items to the
parent bidder server will be done
{
    if ( Time_to_Forwarding ) // Time is appropriate for forwarding
    {
        do // Repeating following statement till getting “ Accepted ” message
        {
            ChildBidderServerSend_bid (ParentBSi. submit_bid (newbid_val)) ;
        }
        while ( action_result == “ Accepted ” ) ; // End of do ... while ;
        Forwarding child bidder server’s bid
value has been accepted by Parent
auction bidder server therefore
Task 3 is completed

    } // End of “ if ( Time_to_Forwarding ) ”
} // End of “ while (Parent_BidderServer) “

```

```

coend ; /* End of Task 3

cobegin /* Start of Task 4 */
/* Task 4 : Executed by the Bidder Server , while auction is running and time is appropriate
to disseminate */
while ( Gd_Bidder Server ) // As long as there are down- group bidder servers and time
is appropriate to disseminate to these bidder servers, sending
the latest max value of item to the down- group bidder servers
will be continued
{
if ( Time_to_Disseminate ) // Time is appropriate for disseminating
{
do // Repeating following statement till getting “ Accepted ” message
{
ParentBidderServerSend_bid (ChildBSi. submit_bid(latestbid_val)) ;
}
while( action_result == “ Accepted ” ); // End of do ... while ;
// Parent's dissemination of latest
// bid value has been accepted by
// child auction bidder server
// therefore Task 4 is completed

} // End of “if ( Time_to_Disseminate )”
} // End of “ while ( Gd_Bidder Server ) “

coend ; /* End of Task 4 */

} // End of “ while (auction_held) ”

```

Figure 4-14

As figure 4-14 shows the bidder server sets initialisation of values and executes four tasks concurrently which are shown by task 1 to task 4 while the corresponding auction is held.

Task 1 shows the action of bidder servers whenever they get a request from clients. If a request for the maximum value information or submission new bid value comes from a client, the server returns the current value (for the maximum value information) or “Accepted ” / “Rejected ” message (for submission a new bid value)to corresponding client. Task 2 shows if the server receives a submission bid (from parent or child bidder servers), a bidding value (*bid_val*) will be obtained and compared with the current value information which has been updated in the whole system through task 4. If the bidding value (*bid_val*) is higher than the current one , the out of bid message will be sent to the client whose bid is now out of bid only if the client has been registered by this bidder server and the current one will be updated and

“Accepted“ message will be send to the corresponding server through the *action_result*.

Task 3 will be executed as long as there are parent bidder servers in the system , and time is appropriate (which will be explained later) in order to forward new bid value to the next parent bidder server. This process will be continued until “Accepted“ message being received (after repeating this task). It means corresponding parent bidder server sends the updated bidding value of each item to its parents in order to update the information (e.g. higher bid, etc) in all bidder servers till the main auction server receives them.

Task 4 illustrates the action of bidder servers while disseminating latest updated information to others. In this regards, bidder servers will periodically disseminate the max value of each item only when the time is appropriate for disseminating (will be explained later) and also the bidder server is not in level 1 (see figure 4-9) of the tree structure. It means that the dissemination of *max_value* will be held to *Child_Bidder_Server* till a “Accepted“ message will be obtained, while a bidder server is existed in down tree group.

Time for forwarding new bid value to parent bidder servers (task 3) and disseminating of latest updated information to child bidder servers (task 4) could be defined in several methods. These times are very important and depend on several parameters (e.g. type of auction, hierarchical architecture of bidder servers, how they are connected together and so on) which will be explained in next section .

4-4-1-2 Forwarding / Disseminating time

The time for forwarding new bid value from the bidder servers to their parent and disseminating of the latest updated information from parent bidder servers to their children could be defined as the following methods:

The first method is called round-base method. In this method main auction server initiates an auction round, including start and terminate time for the bargain auction in auction rounds. Then informs each bidder server and so bidders are invited to place bids to bidder servers through the bidder clients.

Each bidder server therefore starts the auction round; regarding the base round which has been defined by the main auction server, and continues this round till termination time will be met. During the auction round in each bidder server, the highest bid is known to other bidders who have registered with this bidder server and as a result there is opportunity for the bidders to place the highest bid in this bidder server. Termination time is the appropriate time for forwarding and disseminating to and from parents. For instance, if we suppose t_i is the starting time of the auction and t_{i+1} is the terminating time, so bidders could place the bid during the $(t_i + \Delta)$ and $(t_{i+1} - \Delta)$, which has been called *Auction Round* and forwarding / disseminating will be done during the $(t_i - \Delta)$ and $(t_i + \Delta)$ or $(t_{i+1} - \Delta)$ and $(t_{i+1} + \Delta)$ in this auction round (Δ is clock delay).

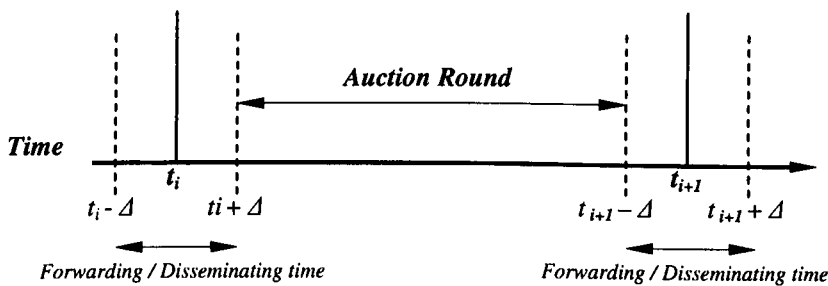


Figure 4-15

This method can probably tempt the bidders to place their bids in the last minute of the round $(t_i - \Delta)$ in their bidder servers, so that their bids are less likely to be known to others before the terminating time and therefore less likely to be out-bid in this round. Such last minute bidding can lead to winner's curse in this round of auction as the winner regrets that he/she placed a far higher bid only because he had no sure way of guessing the bidding intentions of his/her competitors due to the scope for last-minute placement of bids.

In second method, bidder servers define their forwarding time independently as main auction server initiates an auction round, excluding start and terminate forwarding time for each bidder server. In this method, each bidder server, or at most the bidder servers which has been allocated in the same level in our architecture (figures 4-9 and 10) have same forwarding time. It means when bidder server receives the new bid

value and waits to receive other possible new bid values from another bidder client or its child, forwards the maximum new bid value of item to its parent bidder server (the appropriate waiting time depends on the bidder clients and servers' specifications and their communication which could be different in each BAU's in the system).

Main auction server disseminates the latest updated information to child bidder servers (*Task 4*) in a particular time and this process will be continued from each bidder server to its children. Bidder server's dissemination time could be synchronies for all bidder servers which are allocated in the same level. So the dissemination time of BAU_1 and BAU_2 in level n-1 would be the same, BAU_1 to BAU_6 in level 2 have the same dissemination time and so other BAUs in other levels (levels which are shown in figure 4-10) .

4-4-2 Main Auction Server

The procedure for main auction server which is allocated in the root of tree structure (see figure 4-10) and is shown in figure 4-16 is depicted in figure 4-17.

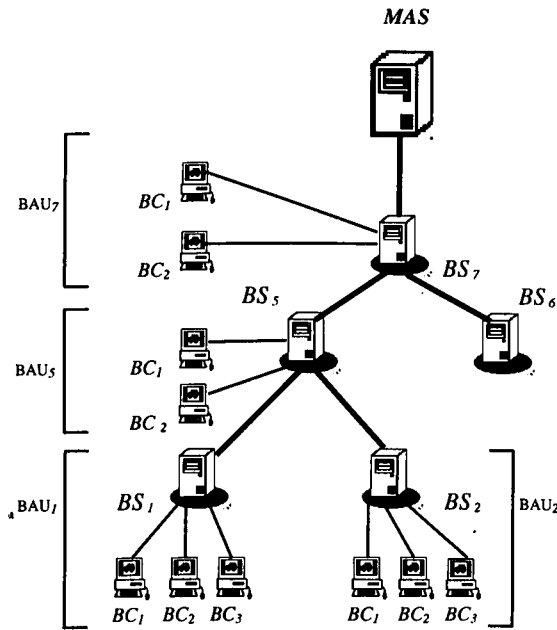


Figure 4-16

4-4-2-1 Main Auction Server's Procedure

```

// Initialization
val = init_val;
result_msg:Boolean;

// As long as the auction is running , the following procedure will be held on the main auction server
(MAS )
while (auction_held)
{
    /* Execute Task 1 of figure 4-14 */

    /* Execute Task 2 of figure 4-14 */

    /* Execute Task 4 of figure 4-14 */

} // End of " while (auction_held) "

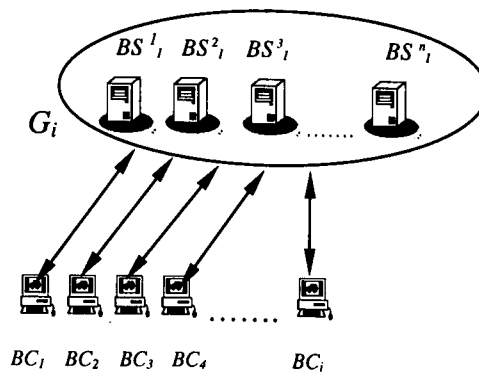
```

Figure 4-17

Regarding figure 4-16, the main auction server sets an initialisation value (*init_val*), for each item as receives the bid from the bidder servers that has been allocated in down level of main auction server (*BS*, in figure 4-16) and executes task 1, 2 and 4 concurrently which has been explained in figure 4-14 . The dissemination time has also explained in forwarding and disseminating time in above.

4-5 Reliability and Fault Tolerance

The basic auction unit which has been described in section 4-2 and figure 4-2, has three subsystems: bidder server (BS_i), bidder client (BC_i) and the communication network that interconnects bidder server to bidder clients. A bidder server (and bidder client) can fail, usually in various ways, and must be built reliably using internal redundancy so that a service remains available. Using well-known redundancy management techniques, reliable server can be built and we achieve the goal of reliability and fault tolerant by replicating the bidder server. We would adopt a replication strategy to build reliable servers, as it would enable a replicated server BS_i to provide fast responses in the absence of fault. Figure 4-18 shows the details of replicated bidder servers in basic auction unit if we suppose there are n replica servers exist for BS_i , which are gathering in specified group and are connected with BC_i . The details of the established connection between BC_i and BS^{n_i} inside the group will be described later.



○ G_i : Group consists of Bidder Servers

Figure 4-18

The communication network between bidder clients and bidder servers is not owned or maintained by the auction service provider, this “should be built reliable” approach does not work for the network, especially in the case of the Internet. The internet generally provides a reliable communication (in the sense that what is sent is received, perhaps after a few retries), so the asynchronous network assumption will be met if BS_i in BAU_i and BS_j in BAU_j are correct (see figure 4-12), a message sent by one to the other is eventually delivered. In fact this assumption requires that communication path between any BS in two BAU if broken, be eventually restored. This enable the bidder

server communication to be reliable but not synchronous, a bound on how long messages can take to reach the destination can not be known with certainty.

A process can fail in many ways, and there are two extreme fault models: Byzantine model, that is a faulty processor, can fail in arbitrary ways and Crash model, which is a faulty processor, fails only by stopping to function. We assume the second fault model in our system.

In reality Client/Server or Communication Network can fail. We therefore need to deploy redundancy to cope with failure. We will consider two approaches of incorporate redundancy:

Approach 1 - Redundancy internal to servers

Approach 2 - Redundancy external to servers

In the first approach, we assume there are more than one server exist for every BS_i server in the BAU_i in our architecture which are gathering together in one group (BS_i 's Group). Figure 4-19 shows the internal structure of each bidder server (e.g. BS_i).

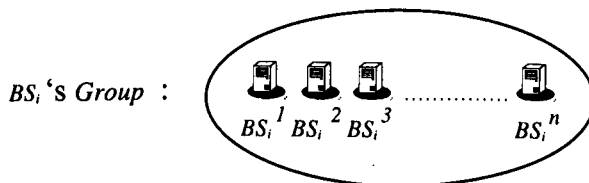


Figure 4-19

A replica auction server group can present each component, which is a number of replicas grouped together that cooperate to provide the same service. The replication protocol must mask failures which occur so that the replicated service can continue to function. Objects with high reliability requirement are replicated (forming an object group) on several nodes, employing some suitable replication protocol to ensure replica consistency. There are basically two classes of replications, passive replication and active replication. In active replication every functioning member of the replica group must receive, process, and reply to other level of replica group. In passive

replication requires only one member of replica group, called primary, must receive, process and reply to other level of replica group. In this approach we have highest redundancy in the system as each server consists of a set of servers.

Second approach for increasing the availability of service is through multicast messages to other existing bidder servers (BS) in other BAU as back up in the distributed auction systems. So instead of a set of servers and using replication method inside the BS_i which explain in the first approach and named redundancy internal to servers, bidder server multicast to other existing bidder servers by themselves. It means that every bidder server which receive messages will multicast them to other bidder servers which will determine by the protocol and in this regards, we can consider this approach as a redundancy external to servers.

The first approach has higher redundancy in the system in comparison to the second approach. However, as the replica is internal in this approach, its architecture therefore is very closed to the non-fault tolerance model and as a result the main up/down protocols and procedures have not been effected.

We will discuss and implement this approach in next sections.

4-5-1 Replication – Based Reliability

A widely used computational model for constructing fault-tolerant distributed applications employs atomic transactions for controlling operations on persistent objects. In order to increase the availability of persistent data which is manipulated under the control of transactions there has been considerable work on data replication techniques. Process group with ordered group communications has also emerged as a model for building available distributed applications. High service availability can be achieved by replicating the service state on multiple processes managed by a group communication infrastructure. These two models are often seen as rivals [Little'99]. Our distributed auction will be implemented with replicated and process group.

Replication is intended at protecting computational resources through the use of redundancy: if a processor fails, then another processor can take over the processing of the failed processor.

Regarding the two best-known replication styles (active and passive) a replicated object is often represented by an object group, with the replicas of the object forming the members of the group. The object group membership may be static or dynamic. Static membership implies that the number, and the identity, of the replicas do not change over the lifetime of the replicated object; on the other hand, dynamic replication allows replicas to be added or removed at run-time.

With active replication, all of the replicas of the object play the same role: every active replica receives each request, processes it, updates its state, and sends a response back to the client. Because the client's invocations are always sent to, and processed by, every server replica, the failure of any of the server replicas can be made transparent to the client. With primary-backup replication, one of the server replicas is designated as the primary, while all the other entire replicas server as backups. A client typically sends its request only to the primary, which executes the request, updates it own state, updates the states of the backups, and sends the response to the client. The periodic state updates from the primary to the backups serve to synchronize the states of all of the server replicas at specific points in their execution. Replication implements *roll-forward* recovery mechanisms that promote liveness by continuing processing where it had been left at the time of the failure. In active replication, in the event of a fault (one of the active replicas crashes), the other replicas continue processing the current request, regardless, thereby implicitly implementing a roll-forward mechanism. In primary-backup replication, in the event of a fault (the primary replica crashes), one of the backup replicas takes over as the new primary and re-processes any requests that the previous primary was performing before it failed. If a backup replica crashes, then, there is no loss in processing. Thus, the roll-forward mechanism is explicitly implemented in the re-election of a new primary replica, and the re-processing of requests by the new primary. Consistency is maintained for both active and primary-backup replication by guaranteeing that partial request execution will not harm since the request will be eventually completed (by "rolling forward").

With roll-forward reliability strategies, invocations are traditionally sent using reliable multicast (also known as reliable group communication), so that all of the replicas of an object receive every request. This is evident in an active replication configuration, where a client does not need to re-issue the request if one of the active server replicas fails (in fact, the client is typically not even aware of this failure). In a primary-backup setting, when the primary has finished processing a request, it multicasts both the response and a state update to the backups before returning the response to the client. The state update allows the backups to synchronize their state with that of the primary. The response is also cached by the backups for retrieval, should the primary fail. If the primary fails, then a backup assumes the role of the new primary transparently. If the primary fails before returning a response to the client, the client will re-issue the request to one of the backups (now the new primary); if the new primary has a cached response and the last state update of the old primary, it can readily return a response; if it doesn't have the cached response, it will re-process the request.

4-5-2 Implementation and Reliability Issues

A group is defined as a collection of distributed auction server in which a member server can communicate with other members by multicasting to the full membership of the group (the property of atomic has been considered).

An additional property of interest is guaranteeing total order: all the functioning members are delivered messages in identical order. Clearly, these properties are ideal for replicated server in distributed auction architecture: each server manages a copy of data, and given atomic delivery and order, it is easy to ensure that copies of data do not diverge.

A multicast made by a server can be interrupted due the crash of that process; this can result in some connected servers not receiving the message. Server crashes should ideally be handled by a fault tolerant protocol in the following manner: when a server does crash, all functioning servers must promptly observe that crash event and agree on the order of that event relative to other events in the system. In an asynchronous

environment this is impossible to achieve: when servers are prone to failures, it is impossible to guarantee that all non-faulty servers will reach agreement in finite time [Fischer'85]. This impossibility stems from the inability of a server to distinguish slow servers from crashed ones. Asynchronous protocols can circumvent this impossibility result by permitting servers to suspect server crashes and to reach agreement only among those servers which they do not suspect to have crashed.

A server group therefore needs the services of a membership service that executes an agreement protocol to ensure that functioning servers within any given group will have identical views about the membership. The membership service also ensures that the sequence of views installed by any two functioning member servers of a group that do not suspect each other are identical.

In the simplified model, which is shown in figure 4-10, seven $BAUs$ are arranged in a tree, with the root being server BAU_7 that connect to MAS with which the seller is assumed to be registered. We could adopt a passive replication strategy to build reliable servers, as it would enable a replicated bidder server BS_i to provide fast responses in the absence of faults. Figure 4-20 shows the completed details of internal structure of BS_i in figure 4-19, if we suppose there are n replica server for BS_i , so in this scenario, the bidder server can provide services despite at most $(n-1)$ replica crashes.

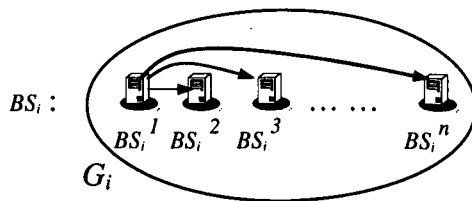


Figure 4-20

In passive replication, only the highest ranked replica, which called the primary (BS_i^1 in figure 4-20), processes, and responds to the requests; for every received request, it multicasts to other replicas the state changes effected and any response produced due to processing of the request. If the primary crashes, the highest ranked among the non-crashed replicas becomes the new primary and continues with the processing of

incoming requests (according the group membership service which will be explained later , the BC_i 's request will be sent to the properly server). In this regard, in passive replication, while every replica may receive the inputs, only the primary sends the server output to other servers. We would describe how the (passively replicated) servers exchange episode messages. Regarding figures 4-19 and 4-20, we will consider a single multicast group also assume that each bidder server in this example, is internally replicated and BS_i^j is the primary of BS_i^j ($j=1$ to n) . The details of internal structure of bidder servers are shown in figure 4-21 (for simplicity in this figure, we assume that $n = 2$, and in this case, at most one replica can crash within each BS_i).

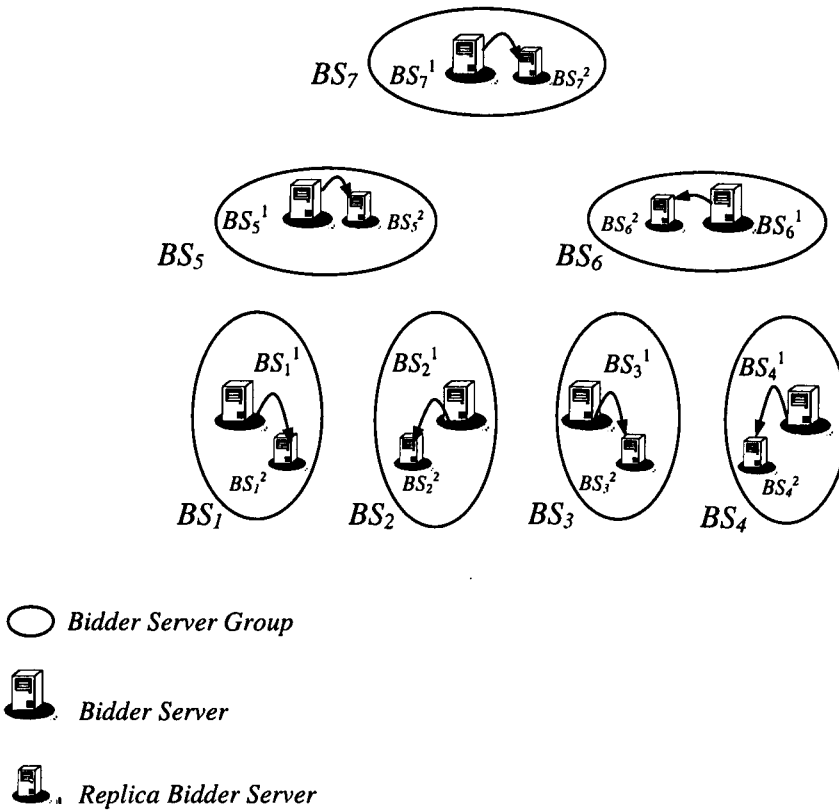


Figure 4-21

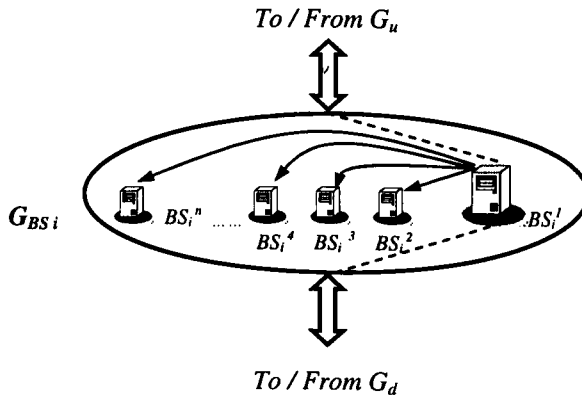
We use group management technique to facilitate the servers to exchange messages and cooperate with each other. Two basic services are assumed: reliable multicast and group membership, which many of the existing group management middleware system can readily provide.

An implementation of passive replication is done using the following services within S_i , a reliable FIFO multicast service which ensures that if the primary crashes during a multicast, either all functioning replicas (only one replica in here) or none of them receives that multicast, and a group membership service which promptly informs the functioning replicas of suspected replica crashes and the order in which these crashes must be viewed with respect to message delivery order. This property of synchronising crash notifications with message delivery order is known as view or virtual synchrony. These services facilitate prompt selection of a new primary after the existing one crashed, and guarantees that the survivors are in agreement on the last multicast the old primary made before it crashed so that the transfer of the processing role from the old to the new primary remains correct.

Regarding to partitioning (not disjointing) the servers into multicast groups based on a tree structure facilitates dissemination of episode messages; a group consists of one parent and its children. These multicast messages which are called episode messages, as their contents are used by each bidder server to form the history of client requests accepted (so far) in the global system. Within a multicast group, bidder server knows each other's identifier and periodically multicast the episode message. We describe how the passively replicated servers exchange episode messages. We will consider multicast groups and assume that each server BS_i is internally duplicated ($n = 2$) and BS_i^1 is the primary of BS_i . The seven bidder servers in the BAUs in figure 4-21 can be configured into three multicast groups ($G1: \{ BS_1^1, BS_5^1, BS_2^1 \}$, $G2: \{ BS_5^1, BS_6^1, BS_7^1 \}$, $G3: \{ BS_6^1, BS_3^1, BS_4^1 \}$) which containing only the bidder server primaries. The configuration is identical to that shown in figure 4-11, if we replace BS_i with BS_i^1 and the direction of multicasting the episode messaging is from up to down (from parent to children in each group). The details of bidder servers' procedure to implement fault tolerance model and reliability issues regarding server replicated will be described in section 4-5-2-1.

Figure 4-22 shows bidder server group (G_{BS_i}), inside of each replica bidder server group and communication between the primary bidder server for incoming/outcoming request to/from G_{BS_i} , if we suppose BS_i^1 is primary of BS_i^j and BS_i^2 is a replica of BS_i^j

($j=1$ to n). The reliable FIFO multicast and group membership services will still be considered in G_{BS_i} .





-  BS_i^1 Primary Bidder Server
-  BS_i^j Replica Bidder Server

Figure 4-22

According to the bidder server group (G_{BS_i}) and down/up -tree group (G_d and G_u), figures 4-11 and 4-22, a non-root parent bidder server periodically aggregates its own episode message with the messages received from other members of its G_u during the past period, and multicasts the aggregated episode message in its G_d , which contains groups of bidder servers (each group includes primary and at least one replica server), e.g. BS_5 to G_{BS_1} and G_{BS_2} as G_d of BS_5 is G_1 which includes BS_5 , BS_1 , BS_2 ; the root bidder server periodically multicasts only its own episode message in its G_d (BS_7 to G_{BS_5} and G_{BS_6}) and etc. The upward propagation of episode messages also works almost in the same way but in the upward direction and forwards episode messages to its parent bidder server instead of multicasting them, which has been done in downward propagation in G_u . So a non-root parent server periodically aggregates its own episode message with messages received from its children during the past period, and forwards the aggregated episode message in its parent: e.g. BS_1 and BS_2 to G_{BS_5} and BS_5 to G_{BS_7} , also the same scenario exists for BS_3 and BS_4 to G_{BS_6} and BS_6 to G_{BS_7} . Thus, in its up-tree group, it represents the bids received by every bidder server of the sub-tree rooted on itself. It is worth mentioning that, each

bidder server consists of one group in which there are primary and replica bidder servers.

The details of bidder servers' procedure to implement fault tolerance model and reliability issues regarding server replicated will be described in next section.

4-5-2-1 Fault Tolerance Bidder Servers' Procedure

The main idea for processing the request in any of the bidder servers is that the client will send its requests, either for existing maximum value or new bid value, through bidder clients to its local server (bidder server in the *BAU*) for processing. Bidder servers process the request, sending back the maximum value or updated new bid value, and replicate to their replica bidder server and periodically forward the latest bid that they have received to their parent bidder server. The same scenario will be done for disseminating the latest updated information to other down ward bidder servers when the time is appropriate for disseminating, through a group consists of one parent and its children (G_i in figure 4-11). Then these bidder servers will replicate this information to their replica bidder server.

The procedure of bidder servers to implement of fault tolerance model and reliability issues described in figure 4-23. There are still 4 tasks which executing concurrently in each bidder server as shown and explained in figure 4-13.

Since the replica is internal (approach 1 in section 4-5), all the up/down main protocols and procedures have not been effected and the same as figure 4-14 that has been explained when fault tolerance is not being supported , however the new operation will be provided by bidder server (RBS_i): This operation will be executed whenever bidder server receives new bid value from client/bidder servers and this value is higher than existing value, so bidder server update the current value with new bid value and will submit its to replica bidder server(s) :

*(RBS_i. submit_bid(value)) : //Bidder Server submits a bid value to its Replica Bidder Server
(Each bidder server is internally replicated and BS_i¹ is the primary of BS_i^j)*

The other operations are the same as operations which explained in figures 4-3 and 4-4).

For clarifying of RBS_i's operation, the bidder servers procedure in fault tolerance model (Approach 1 - Redundancy internal to servers) will be shown in figure 4-23.

4-5-2-1-1 Bidder Servers' Procedure

// Initialization

*val = init_val;
action_result:String;*

// As long as auction is running, the following procedure which includes 4 concurrent tasks will be held on the Auction Bidder Servers :

while (auction_held)

{

cobegin / Start of Task 1 */*

/ Task 1 : Executed by the Bidder Servers, whenever they get request either for existing max_val or submission new bid value // Bidder Server replies, whenever it receives message from clients*

// Bidder server replies, whenever receives message from clients

get_operation_request (); // Bidder server receives a message from client

case request_type of :

get_bid_val (max_val); // Bidder server has been asked by client for existing max_val

*{
return (max_val); // Bidder server replies existing max_value to corresponding client*

}

submit_bid (bid_val); // Bidder server receives new bid value from client

*{
bid_val = clientbid(bid_val); // bid value from client*

if (bid_val > val)

*{
val = bid_val; // Bidder server updates current value with new bid_val which has been received from client if it is higher than the current bid_val*

RBS_i. submit_bid(val); // Bidder Server submit the bid(val) to Replicated Bidder Server (each bidder server is internally duplicated and BS_i¹ is the primary of BS_i^j)

action_result = " Accepted " ;

return (action_result); // Client's bidding has been submitted by bidder server and sends " Accepted " to the corresponding client

} // End of " if (bid_val > val) "

else

```

    {
        action_result = "Rejected";
        return (action_result); // Client's bidding has not been accepted due to new
                                bid_val was not higher than current bid_val ,
                                therefore "Rejected" will be returned to the corresponding client
    }
other ; // The receiving message is not about existing max_val nor new bid_val
{
    ignore the message; // Ignoring the message if bidder server receives
                        messages neither about existing max_val nor new bid_val
}

coend /* End of Task 1 */

cobegin /* Start of Task 2 */

/* Task 2 : Executed by the Bidder Servers , whenever they receive submit message from
other bidder servers , either its parent or children */

get_operation_request () ; // Bidder server receives a message
case request_type of :

submit_bid(bid_val) ; // Receives submit message from other bidder server
{
    bid_val = BidderServer (bid_val) ; // Bid value from bidder server
    if (bid_val > val)
    {
        val = bid_val ; // Auction Bidder Server updates current value with new
                        bid_val if it is higher than the current bid_val

        RBSi. submit_bid(val); // Bidder Server submit the bid( val) to Replicated
                                Bidder Server ( each bidder server is internally
                                duplicated and BSi1 is the primary of BSi1 )

        } // End of " if (bid_val > val) "
        action_result = "Accepted" ;
        return (action_result) ; // New bidding has been accepted (Auction Bidder
                                Server sends " Accepted " to the corresponding sender)
    } // End of "submit_bid(bid_val)."

other ; // The receiving message is not about submitting bid value
{
    ignore the message; // Ignoring the message
}
coend ; /* End of Task 2 */

cobegin /* Start of Task 3 */

/* Task 3 : Executed by the Bidder Server as long as parent bidder servers are existing in
order to forward new bid to the next parent bidder servers */

// Executing Task 3 in " Bidder Servers' procedure " in figure 4-14

coend ; /* End of Task 3

cobegin /* Start of Task 4 */

```

```
/* Task 4 : Executed by the Bidder Server , while auction is running and time is appropriate  
to disseminate */  
  
// Executing Task 4 in “ Bidder Servers’ procedure “ in figure 4-14  
  
coend ; /* End of Task 4 */  
  
} // End of “ while (auction_held) ”
```

Figure 4-23

4-6 Conclusions

In this chapter we have described the Internet auction service requirements and structure of existing Internet auction systems which is based on centralised auction bidder servers.

Hierarchical architecture and Tree-based recursive design approach is known to be one of the best methods to achieve Internet auction’s scalability and responsiveness requirements. The architecture of centralised and distributed Internet auction has been presented and the replicated bidder servers has also been introduced in order to achieve the reliability and fault tolerance in centralised and distributed system architecture for Internet auctions .

Furthermore, the new algorithms and protocols which have been supported by centralised and distributed system architecture for Internet auction have been presented.

According to this architecture, implementation of the distributed system architecture for Internet Auctions will be shown and performance results obtained from experiments are presented in next chapter.

Chapter 5

In this chapter implementation of the distributed system architecture for Internet Auctions described in chapter 4 and performance results obtained from experiments are presented. Firstly the details of implementation will be discussed, including NewTOP reliable group communication system as a tool used in our implementation and finally performance results will be presented.

5-1 Implementation of Distributed Auction System

As explained in chapter 4 and regarding the Bidder Servers' procedures which have been discussed in 4-4-2-1 and 4-5-2-1 (in non fault tolerance and fault tolerance models) the auction system is structured as a number of basic auction units (BAU_i) that may be geographically separated over the Internet. A single bidder server (BS_i) and a number of bidder clients (BC_i) represent each basic auction units (BAU_i); bidder servers (BS_i) are structured hierarchically (chapter 4, section 4-3), with bidder clients (BC_i) placing bids and/or advertising items for sale via their local bidder server (BS_i). Bidder servers (BS_i) are semi passively replicated (in fault tolerant model, see chapter 4 , section 4-5-2) , locally, to improve reliability within a single basic auction units (BAU_i).

Each bidder server (BS_i) consists of four basic types of component: Auction object, Reliability object, Distribution object and Group communication and each component is implemented as a CORBA object.

The details of these four components are as follows and the relation of them will also be shown in next sections:

5-1-1 Auction Object

The auction object implements the auction logic of the system via a number of services:

- *Seller* – Allows the registration of seller details within the Bidding service. Contact details of a seller to enable buyers to trade with sellers are a minimum requirement and are managed by the seller service.
- *Buyer* – Allows the registration of buyer details within the auction service. As with the seller service, the primary purpose of the buyer service is to provide sellers with the relevant details of a buyer to enable trading between buyer and seller.
- *Product* – A product is made available to buyers by sellers via the product service. All the information required by buyers to enable an auction of a product is supplied by the product service (e.g., selling price, product description).
- *Trader* – Manages the bidding process. Buyers may place bids and buyers/sellers may enquire about the bidding status of products (if the auction in use allows this). The trader service also enforces the appropriate style of auction (e.g., English auction ,Dutch auction).

5-1-2 Reliability Object

Reliability objects enable the basic auction unit to tolerate servers' crash. Each auction object replica is provided with a reliability object. A reliability object provides two basic functions: accepts requests from bidder clients and implements semi passive replication [D'efago'98] policy (chapter 4, section 4-5-2). Figure 5-1 describes the handling of a bidder client request; M_1 is the initial bidder client request. On receiving M_1 , R_1 multicasts this request (M_2) to other reliability objects (two in the diagram) and forwards M_1 to its own auction object (M_3). On receiving M_2 , R_2

forwards M_2 (M_4) to A_2 . As this is a semi passive replication scheme, only replies from A_1 are returned to the bidder client via the reliability object of R_1 and replies from A_2 through the reliability object of R_2 are ignored (see procedures 4-2-1 and 4-5-2-1-1 in chapter 4).

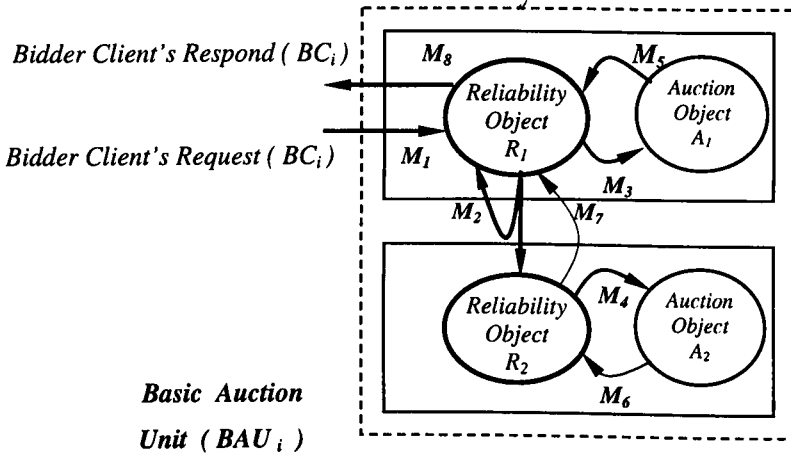


Figure 5-1 Reliability Objects in a Basic Auction Unit

The summary of the above actions are as follows:

- 1- [Bidder Client's Request to R_1 (Say M_1)]
- 2- [R_1 Multicast M_1 to R_1 and R_2 (Say M_2)] and [R_1 forward M_1 to A_1 (Say M_3)]
 // The two above operations (in step 2) will be accomplished in parallel
- 3- [R_2 forward M_2 to A_2 (Say M_4)]
- 4- [Reply from A_1 to R_1 (Say M_5)] and [Reply from A_2 to R_2 (Say M_6)]
 // The two above operations (in step 4) could be accomplished in parallel
- 5- [Reply from R_2 to R_1 (Say M_7)]
- 6- [Reply from R_1 to Bidder Client's (Say M_8)]

According the figure 5-1, the Reliability Object (R_i) may be co-located in the same addressable space as an Auction Object (A_i); R_1 and A_1 or R_2 and A_2 , or may be in a different addressable space. In the latter case, the reliability object and its associated auction object may fail independently of each other (R_i and A_i). In this scenario, when an auction object fails (A_1) the reliability object (R_1) instantiates a new auction object, gaining state for the new replica from existing replicas (A_2) (via other reliability objects (R_2)). When a reliability object fails (R_1), a correctly functioning

reliability object (R_2) instantiates a replacement and associate this new reliability object with the existing auction object (A_1).

The failure of the semi passive reliability object R_2 will require R_1 to buffer all bidder client requests (BC_i) until a new version of R_2 is instantiated, after which, such requests may be forwarded to R_2 . The failure of R_1 will result in R_2 assuming the role of primary. R_2 will buffer all bidder client requests until a replacement for R_1 can be instantiated, forwarding such requests when the replacement for R_1 is available.

5-1-3 Distribution Object

According hierarchical architecture and its implementation (sections 4-3 and 4-4 in chapter 4), each auction object is assigned a distribution object in our implementation. The distribution object is responsible for imposing the hierarchical structure of the global auction system and managing the distribution of episode messages throughout this structure (chapter 4 , section 4-4-1-2 and procedures in sections 4-4-1-1 and 4-5-2-1-1).

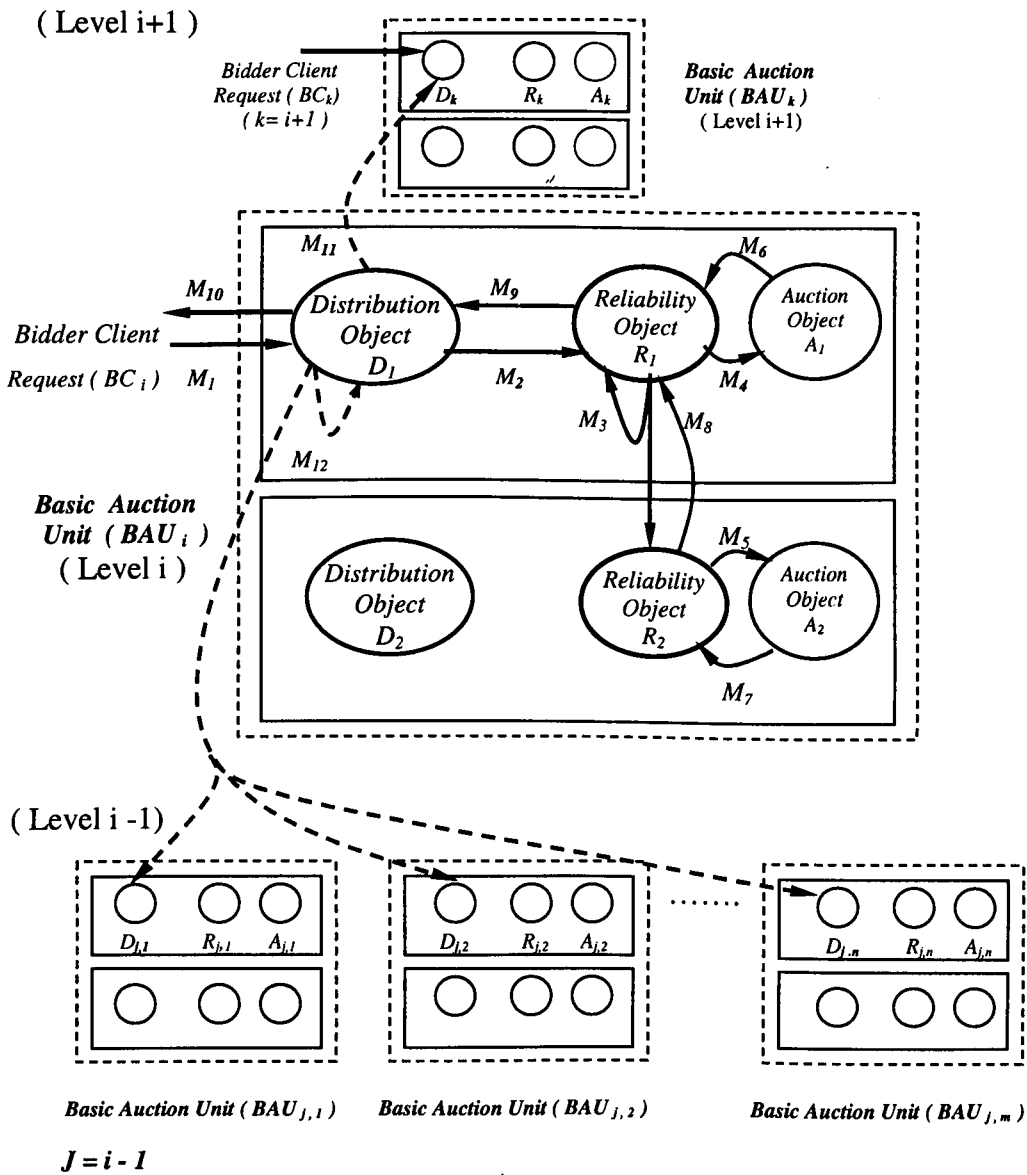


Figure 5-2 Distribution, Reliability and Auction Objects in the Basic Auction Units

When auction objects are replicated, the distribution object (D_i) is placed between bidder clients (BC_i) and a reliability object (R_i) (see figure 5-2). The distribution object (D_1) accepts bidder client requests (M_1) and forwards them to its local reliability object (M_2 to R_1). The system administrator may determine the frequency a distribution object forward (child bidder server to its parent) and multicast episode messages (parent to their child bidder servers); i.e. from D_1 to D_k and $D_{j,m}$ ($m=1$ to n) say M_{11} and M_{12} respectively. Episode messages received by a distribution object

(apart from its own) are disassembled into the original bidder client requests; each request is forwarded to the local reliability object (as M_2).

The detail actions of basic auction units which are shown in figure 5-2 are as follows:

- 1- [Bidder Client's Request to D_1 (Say M_1)]
 - 2- [D_1 forward M_1 to R_1 (Say M_2)]
 - 3- [R_1 Multicast M_2 to R_1 and R_2 (Say M_3)] and [R_1 forward M_3 to A_1 (Say M_4)]
// The two above operations (in step 3) will be accomplished in parallel.
 - 4- [R_2 forward M_3 to A_2 (Say M_5)]
 - 5- [Reply from A_1 to R_1 (Say M_6)] and [Reply from A_2 to R_2 (Say M_7)]
// The two above operations (in step 5) could be accomplished in parallel.
 - 6- [Reply from R_2 to R_1 (Say M_8)]
 - 7- [Reply from R_1 to D_1 (Say M_9)]
 - 8- [Reply from D_1 to Bidder Client's (Say M_{10})]
- // The following two operations (Forwarding the new bid value of items to bidder server's parent and / or Disseminating latest bid value to bidder server's children) will be done whenever time is appropriate for disseminating (by normal forwarding from bidder server to its parent bidder server and / or multicasting from bidder server to its children bidder servers respectively ; see figure 4-13 (Task 3 and Task 4) and procedure in figure 4-14 in chapter 4).*
- [D_1 forward to D_k (Forwarding from Child to Parent (Say M_{11}) to (BAU_k))]
- [D_1 multicast to $D_{j,m}$ ($m=1$ to n) (Multicasting from Parent to its Children (Say M_{12}) to ($BAU_{j,m}$ ($m=1$ to n))]

The distribution object of the semi passive replica (D_2) does not receive any bidder client requests nor does it send or receive episode messages. When the primary replica fails D_2 assumes responsibilities of D_1 .

5-1-4 Group Communication Object

The group communication requirements of the reliability objects in our implementation are satisfied by the Newtop service (a CORBA service) [Morgan'99]. The Newtop service is a distributed service and achieves distribution with the aid of the Newtop Service Object (NSO). Each group member (reliability object or distribution object) is allocated an NSO. Group related communications required by a member are handled by its NSO .

The Newtop service consists of three services implemented by corresponding objects within the NSOs: membership, invocation/multicast, and group management, also the management service provides members with create, delete and leave group operations and the invocation/multicast service provides four group invocation operations (wait for responses from all, from majority, from one and an asynchronous, no wait invocation). The membership service maintains the membership information and ensures that this information is mutually consistent at each member. This is achieved with the help of a failure suspector that initiates membership agreement as soon as a member is suspected to have failed.

The details of the NewTOP group communication services, structure including its components will be described in the next section.

5-2 The NewTOP Group Communication Service

The Newcastle Total Order Protocol (NewTOP) is a CORBA compliant, crash-tolerant, partitionable middleware system. The system implementation is centred on a CORBA node called the NewTOP Service Object (NSO). NSO is the pinnacle of the design of NewTOP in that it does total order, multicasts, inter group communication. There is one NSO for each group member. When application processes want to form a group with a common goal and to avail themselves of group communication services to this end, each process is allocated an NSO as shown in Figure 5.3. An application process A_i acts as a 'client' to its NSO in obtaining group communication services from the latter. The communication between client A_i and its NSO, and the communication between NSO's themselves are handled by an ORB [Mpoeleng'05].

An NSO and its application 'client' need not reside on the same host, for the reasons that NSO is a CORBA object and the communication between an NSO and its client is handled by the ORB (location independence); however, for performance reasons, they are normally hosted by the same node. Further, NewTOP requires an client A_i to be member of a group in which client A_i intends to multicast and permits client A_i to be a member of more than one group at the same time. Being a partitionable system, it does not however support merging of partitioned sub-groups.

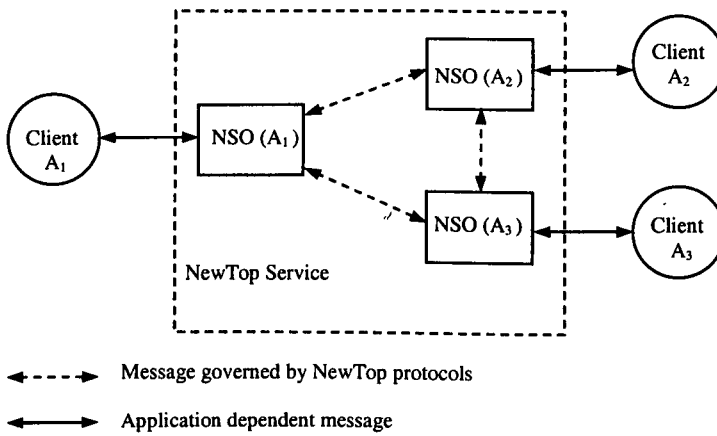


Figure 5-3: Clients of the NewTop service and associated NSO

An NSO comprises of two subsystems: Invocation service and Group Communication (GC) service (Figure 5.4). The former allows the application to specify the type of NewTOP service needed and marshals a multicast message accordingly. The latter implements protocols to provide a variety of services: symmetric total order, asymmetric total order, reliable multicast, simple (unreliable) multicast and (partitionable) group membership.

When client A_i multicasts a message to the group, the message is marshaled into a generic CORBA type any by the Invocation service and the relevant protocol of the group communication service is invoked to deliver the message. At the delivery end, the reverse happens. The Invocation service at a destination end unmarshals the delivered message (of type any) and delivers it to the client application A_j [Mpoeleng'05].

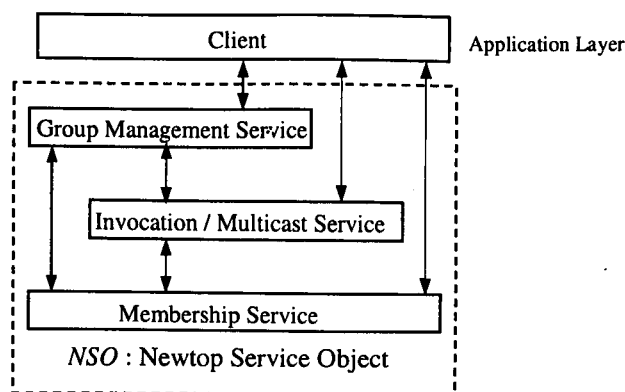


Figure 5-4 : The NewTop Services: Access and Structure

The management service provides clients with create, delete and leave group operations. The invocation/multicast service provides four group invocation operations (wait for responses from all, from majority, from one and an asynchronous, no wait invocation). The membership service maintains the membership information and ensures that this information is mutually consistent at each member. This is achieved with the help of a failure suspector that initiates membership agreement as soon as a member is suspected to have failed. The client can obtain the current membership information by invoking 'groupDetails' operation. Figure 5.5 summarizes the main operations provided by an NSO [Morgan'99].

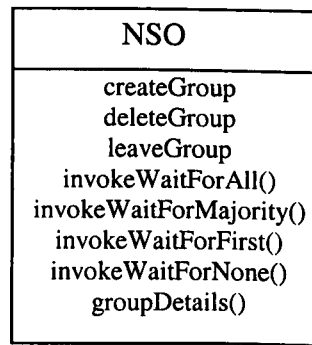


Figure 5-5 : Summary of NSO operation

A more details of each service are as follow.

5-2-1 Management Service

The management service manages the creation of new groups, the deletion of existing groups, and the change of membership for existing groups.

Creating a group – The creation of a group is initiated by a client of the NewTOP service; it is assumed that the relevant NSOs have already been created. The client is required to give the group an identifier that can aid the client and the NewTOP service in differentiating between groups. This identifier should be unique and consists of a string of ASCII characters. The client is also required to supply an initial member list containing the IORs of the NSOs. The objects identified by the list are considered group members at the start of a group's life.

Ordering - A client may specify what ordering guarantees are required for enabling message delivery (total, causal, arbitrary) and what style of ordering protocol is to be used (asymmetric or symmetric) within a group.

Type - A group may be designated as lively or event driven. In an event driven group the timesilence mechanism (used for detecting member failure and to advance message delivery) is only active in the presence of computational messages derived from a client. In a lively group the timesilence mechanism is active all of the time.

Deleting a group - A client may specify, at any time, that a group is to be deleted. The deletion of a group does not result in the deletion of the individual group members, but only of the abstract group entity. When a group has been marked for deletion all group members are told by the management service to voluntarily leave the group. The group membership service may, due to members leaving and/or members failing, indicate to the management service that the membership of a group has reduced to a singleton. This results in the management service deleting this group.

Leaving a group - At any time during the lifetime of a group a member may request to leave the group.

The underlying protocols do not support an explicit join facility. Since members are permitted to belong to several groups a similar effect can be obtained by members forming a new group and exiting the previous group. Joining a group in this manner results in the identifier of the group changing after each join is accomplished. As the joining of a group may result in computations that are application dependent (e.g., state transfer in replica groups), it is left to the application developer to decide how best to tackle any inconvenience associated with changing group identifiers.

5-2-2 Invocation / Multicast Service

The Invocation/multicast service manages all aspects of messages related to the delivery of client requests to object groups and server replies to clients.

Invocation - A client may issue a request in asynchronous or synchronous styles.

Server replies - A client issuing a synchronous request may dictate the number of server replies to wait for (all, majority or one) and how these replies are handled by an NSO in the event of failure within the server group.

Protocols - The invocation/multicast service provides the protocols for guaranteeing the ordering (total, causal or arbitrary) and delivery atomicity (with respect to group view updates) of messages.

The NewTOP service relies on the message passing capabilities of the ORB for enabling multicast communication between group members. Since, at present, ORBs only provide one to one communication, multicasting has been implemented by making invocations in turn to all the members. CORBA supports synchronous and asynchronous RPC. The asynchronous style RPC is termed *oneway* and allows an RPC to be sent while enabling the calling client to continue execution (no blocking of client). However, the CORBA specification indicates that a oneway RPC does not need to be attempted by an ORB. Some ORBs do implement oneway, some do not. For this reason, synchronous RPC has been chosen for use in the NewTOP Service. Multiple threads of execution are used to obtain parallelism and prevent client blocking[Morgan'99].

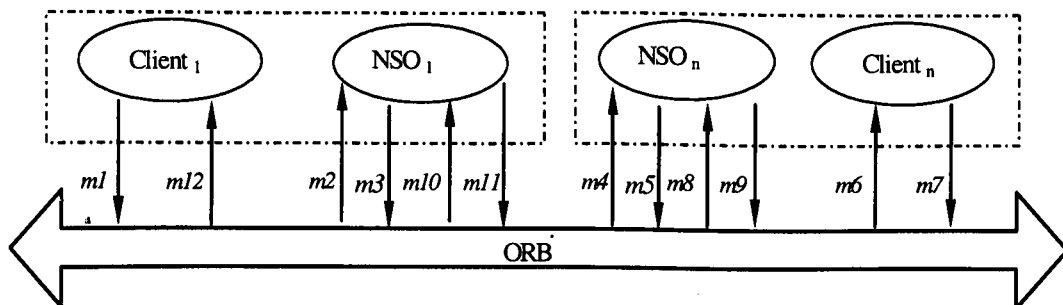


Figure 5-6 : Message interactions in a group multicast

The principal message exchanges involved in making a group invocation are now explained. Assume a group of n identical objects and *object1* wants to make a synchronous group invocation on some operation of the objects; this invocation will have to be made via the group service. Figure 5-6 shows two of these objects and their respective NSOs. The client of the NewTOP service making the invocation is required to marshall the invocation request, consisting of the name of the function and associated parameter list, into a single structure and send it to its NSO. Message 1 (*m1* for short) is such a message; *m2* is its reception. As a result, NSO_1 sends NewTOP specific messages to other NSOs; in figure 5.6, *m3* is such a message and *m4* is its reception at NSO_n . NSO_n responds by composing and sending the appropriate invocation message, *m5*, to its target object (*object_n*); *m6* is its reception at *object_n*. The response from *object_n* (*m7*) is received by NSO_n (*m8*); NSO_n then sends NewTOP specific message (*m9*), it is received at NSO_1 (*m10*), from here *m11* and *m12* indicate the final journey back to the invoker. An NSO (such as NSO_n) that is receiving an invocation on behalf of its target object must be able to compose the type specific invocation on the fly; this is made possible by making use of the *Dynamic Invocation Interface* (DII) feature of the ORB (in the figure 5.4, the invocation represented by the message pair *m5*, *m8* uses DII).

5-2-3 Group Membership Service

The group membership service maintains a mutually consistent view of a group membership for each member of a group.

Detecting member failure – The NewTOP service may suspect member failures with the aid of a timeout based failure suspicion protocol and/or exceptions thrown by the underlying ORB when attempting an RPC. Suspecting a member of failure results in the execution of the membership agreement protocol; the suspected member will be removed from the group or will remain in the group with all suspicions removed. Whatever the outcome of the protocol, group members will retain mutually consistent views of the group membership.

Single group membership – When membership of a group falls to singleton the group is marked for deletion, the management service is informed and all information relating to the group is removed.

Changes in group membership are reported to the invocation/multicast service to enable pending messages to be appropriately managed (i.e., delivered or disregarded). The group membership service also provides clients with a mechanism that enables clients to gain current group views of any group which the client is a member[Morgan'99].

5-3 Performance Evaluation of Distributed Auction System

To demonstrate the effectiveness of distributed auction system, we present performance figures related to a restricted implementation of the hierarchical architecture; only a single bidder server group is implemented, releasing the need for a root node. We experiment with centralized auction server and distributed two-server auction systems (both replicated and non-replicated versions considered). We measure the time it takes for a bid to be registered at all auction objects in the system from the moment a bid is sent by a client. These measurements should not be treated as 'absolute' figures, but rather as an aid to compare the effectiveness of our distributed over a centralized in auction system architecture. For a fair comparison, all experiments were conducted overnight during which load fluctuations over the Internet were small. Four different types of experiment were carried out :

- (i) Centralised Bidder Server Non-Replicated Auction Object,
- (ii) Distributed Bidder Servers (two) Non-Replicated Auction Object,
- (iii) Centralised Bidder Server and Replicated Auction Object,
- (iv) Distributed Bidder Servers (two) and Replicated Auction Object.

Bidder Clients (BC_i) were configured to issue bids as frequently as possible; as soon as a reply is received another bid is issued. Bidder Client (BC_i) numbers were increased gradually from 100 to 200 in increments of 10. At each of these increments,

registering 100 bids for each bidder client (BC_i) in all auction objects is timed, and the average is taken.

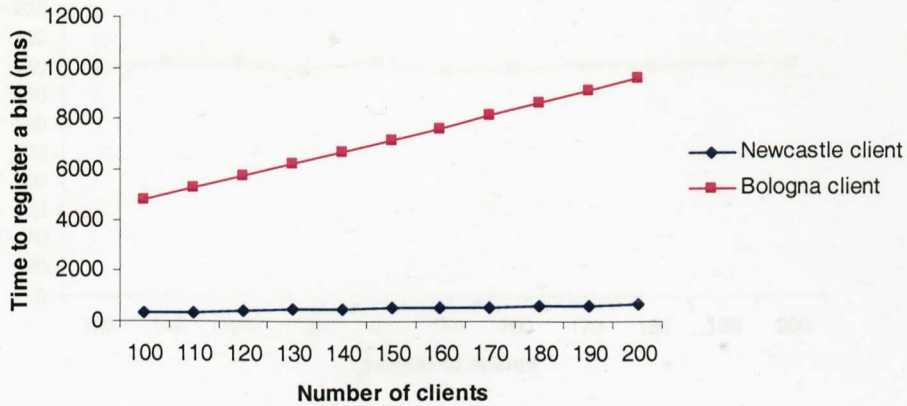
Communications between bidder clients and bidder servers were enabled via the Internet. Pentium Linux machines were used as hosts for bidder clients and bidder servers. Replicated bidder servers (when used) were located on different machines on the same LAN. All objects of a single bidder server were compiled into the same addressable space (e.g., D_I, R_I, A_I in figure 5-2).

The implementation language used was Java and the ORB used was ORBacus 4.0b3 [<http://www.orbacus.com/products/orbacus.html>]. Bidder clients and bidder servers were located at Newcastle (England) and Bologna (Italy). Bidder clients were always equally distributed between England and Italy. (That is, when we say the number of clients is 100, it is 50 in England and 50 in Italy.) In the single bidder server cases, only the Newcastle bidder server is operational which is accessed by both Italian and English bidder clients.

5-3-1 Centralised Server, Non-replicated Auction Object

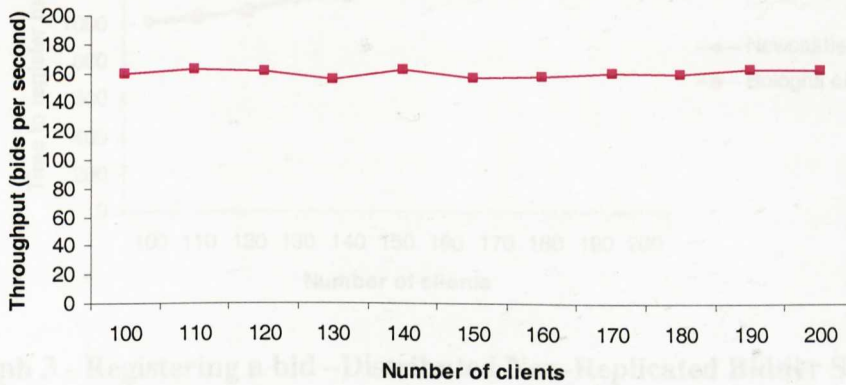
To enable comparative analysis of the performance figures, the CORBA RPC time of a bidder client in Italy communicating with a bidder server (without distribution or reliability objects) in Newcastle was 94 ms (and approximately the same for bidder client in Newcastle and bidder server in Bologna), the equivalent CORBA RPC between a single bidder client and a local bidder server (e.g., communicating over the same LAN) was approximately 6-7 ms for both Newcastle and Bologna.

The first experiment (Graph 1) to be made is the time for a bid sent from a local bidder client (Newcastle) to register in the auction object is far lower than the time taken to register a bid sent from a distant bidder client (Bologna).



**Graph 1 : Registering a bid – Single Non-Replicated Bidder Server
(in Newcastle)**

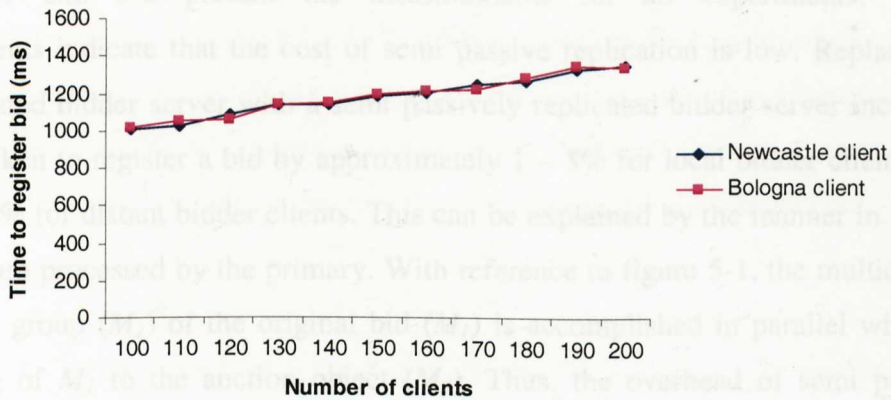
This is expected, as the latency between the bidder server and the distant bidder clients is approximately 15 times larger than the latency between the bidder server and local bidder clients. Both, local and distant bidder clients take longer to place bids when bidder client numbers are increasing (shown by the upward slope in graph 1); doubling the number of bidder clients doubles the time taken for bids to be registered. This indicates that the bidder server may be overloaded, and this assumption appears to be confirmed by the slightly decreasing slope in graph 2 (throughput: bids per second).



**Graph 2 : Throughput – Single Non-Replicated Server
(in Newcastle)**

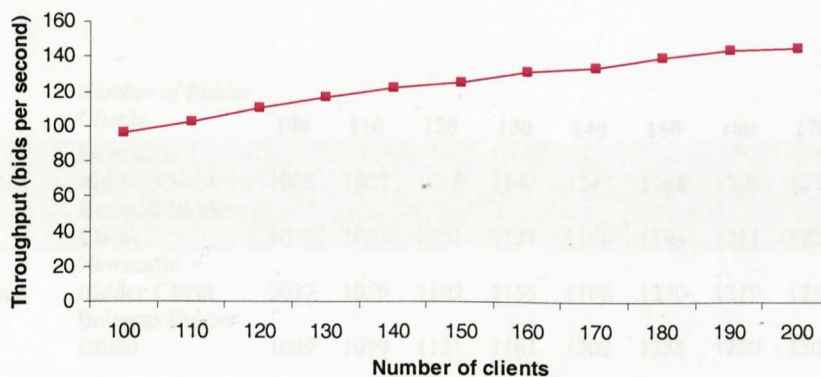
5-3-2 Distributed Bidder Servers, Non-replicated Auction object

Graphs 3 and 4 present the measurements when two bidder servers are present, each serving local bidder clients (Newcastle & Bologna). Due to the latency between these two distributed bidder servers, the time taken to register a bid at both bidder servers is approximately 3 times slower than a bidder client registering a bid at its local bidder server (comparing graph 1 with graph 3; 336ms in first and 1006ms in second experiment, both for 100 bidder clients in Newcastle bidder server). However, in this scenario there are no distant bidder clients. Therefore, no bidder client suffers the extremes of poor performance witnessed when only a single bidder server is present (distant bidder clients 15 times slower than local bidder clients) as shown in graph 1.



Graph 3 - Registering a bid –Distributed Non-Replicated Bidder Server (in Newcastle & Bologna)

Another experiment of interest is the degree of the slope of graph 3 compared to that of graph 1. Graph 1 shows a much steeper slope (for both Newcastle and Bologna bidder clients) compared to graph 3. When two bidder servers are present, increasing the number of bidder clients does not have the same adverse effect on time taken to register a bid as when only a single bidder server is present. In graph 4, the slope is still increasing when bidder client numbers are increased, indicating that the bidder server is not yet overloaded (as is the case when only a single bidder server exists – shown in graph 2).



Graph 4 : Throughput – Distributed Non-Replicated Bidder Serve (in Newcastle & Bologna)

5-3-3 Replicated Auction Objects (Centralised and Distributed Approaches)

Tables 5-1 and 5-2 present the measurements for all experiments. These measurements indicate that the cost of semi passive replication is low. Replacing a non-replicated bidder server with a semi passively replicated bidder server increases the time taken to register a bid by approximately 1 – 5% for local bidder clients and less than 1% for distant bidder clients. This can be explained by the manner in which messages are processed by the primary. With reference to figure 5-1, the multicast to the replica group (M_2) of the original bid (M_1) is accomplished in parallel with the forwarding of M_1 to the auction object (M_3). Thus, the overhead of semi passive replication is the processing of M_1 by the reliability object and the time taken for M_3 to be received and processed by the auction object. As the reliability and auction objects are compiled into the same addressable space, the cost of sending M_3 is very low.

		<i>Number of Bidder Clients</i>											
		100	110	120	130	140	150	160	170	180	190	200	
Non-Replicated	Newcastle Bidder client	336	362	397	445	460	512	541	565	603	622	655	
	Bologna Bidder client	4776	5261	5734	6225	6685	7173	7635	8126	8610	9073	9556	
Replicated	Newcastle Bidder client	341	377	413	443	471	504	566	581	619	664	689	
	Bologna Bidder client	4778	5263	5740	6230	6693	7172	7666	8127	8626	9085	9593	

Table 5-1. Performance of Replicated and Non-Replicated Auction Objects for Centralized approach

		<i>Number of Bidder Clients</i>											
		100	110	120	130	140	150	160	170	180	190	200	
Non-Replicated	Newcastle Bidder Client	1006	1027	1088	1143	1144	1188	1208	1254	1266	1328	1349	
	Bologna Bidder Client	1015	1051	1061	1138	1146	1194	1211	1223	1285	1346	1336	
Replicated	Newcastle Bidder Client	1027	1059	1102	1155	1168	1230	1219	1261	1314	1349	1365	
	Bologna Bidder Client	1049	1099	1121	1161	1209	1238	1250	1305	1340	1398	1427	

Table 5-2. Performance of Replicated and Non-Replicated Auction Objects for Distributed approach

5-3-4 Conclusions

In this chapter we have described a hierarchic architecture to enable Internet-based applications to satisfy the reliability requirements of a large number of geographically dispersed bidder clients. We have demonstrated the effectiveness of our architecture by implementing, and gaining performance measurements from, a distributed auction system.

According to the graphs number 1 to 4 and tables 5-1 and 5-2, the performance measurements presented indicate that our solution achieves scalability, as the total load is shared amongst many bidder servers (comparing figure 2 and figure 4). Furthermore, by presenting bidder clients with local bidder servers we ensure that bidder server access times are similar for all bidder clients, irrelevant of geographic location. This removes the unfair advantage that clients close to a bidder server can have over remote bidder clients in a centralised approach.

Chapter 6

Conclusions

This chapter summarises the material that has been covered in the thesis and gives an indication of the possible areas of future research.

6.1 Thesis Summary

Auctions involve competitive bidding among buyers and sellers of goods, and their provisioning places new demands from the underlying distributed computing infrastructure. The main objective of this research was the development of distributed systems architecture for dependable Internet based services for online competitive bidding and meeting the Internet auction's requirements. However, the types of bidding models which can be supported over distributed auction services was very much an open issue.

The main contribution has been the development of a hierarchic auction architecture that can scale arbitrary. In addition we have investigated various system architecture issues for ensuring dependability.

Current Internet-based auction services rely, in general, on a centralised auction server; it means the applications with large and geographically dispersed bidder client bases are currently supported in a centralised manner; bidder client requests (over the Internet) to systems located in a central place for processing. Such an approach is fundamentally restrictive as too many users can overload the server, making the whole auction process unresponsive. In addition, with a large number of geographically dispersed bidder clients, a centralised auction is not scalable.

Internet auction service requires meeting scalability, responsiveness, fairness and data integrity. A centralised auction server architecture can not deal adequately with issue of service availability and scalability. Typically, such an architecture can be vulnerable to server's failures, if not equipped with sufficient redundancy. Furthermore, server's overloading may occur, if an arbitrary large number of users concurrently access the service and a customer (a local bidder client) who is close to a central bidder server can have faster bidder server access than a remote bidder client, and thus may have an unfair advantage over the latter.

Therefore the ways of enabling widely distributed arbitrary large number of auction servers to cooperate in conducting an auction were investigated. The goals of responsiveness and scalability are achieved by replicating the auction/bidding service across a number of these auction bidder servers. Allowing a user to place a bid at any one of the bidder servers was our solution to achieve scalability and responsiveness, since the total load was shared amongst many bidder servers which has been distributed in our auction system , and users can interact with bidder server 'closest' to them. New algorithms and protocols based on hierarchical architecture and Tree-based recursive design were presented.

We described a hierarchic architecture to enable Internet-based applications to satisfy the Quality of Service (QoS) and reliability requirements of a large number of geographically dispersed bidder clients. The bidder clients and bidder servers procedures in both non-replicated and replicated were explained and the effectiveness of our architecture was demonstrated by implementing, and gaining performance measurements from a distributed auction system.

The implementation framework addresses the important issues: "Building a reliable/available bidder server through process replication"

The performance measurements presented in this thesis indicate that our solution achieves scalability, as the total load is shared amongst many bidder servers. Furthermore, by presenting bidder clients with local bidder servers we ensure that bidder server access times are similar for all bidder clients, irrelevant of geographic

location. This removes the unfair advantage that bidder clients close to a bidder server have over remote bidder clients in a centralised approach.

6-2 Future Work

The hierarchic architecture explained and developed in this thesis is very attractive for conducting auctions on a global scale, as it enables a federation of basic auction units to co-operate. We evaluated this architecture for supporting English auctions. This work can be extended to support distributed auction system for various other types of auction which are popular on the Internet. It would also be interesting to investigate if the idea presented here can be applied to other application areas such as Multiplayer Online Games.

The measured performances in chapter 5, confirm that auction's scalability requirement is achieved. We presented performances figures related to a restricted implementation of the hierarchical architecture by only a single bidder server group and experimented with centralized auction server and distributed two-bidder servers auction system (in Newcastle and Bologna) in both replicated and non-replicated version to consider reliability and fault tolerance in centralised and distributed system architecture for Internet auctions. Since auction's requirements are achieved in this architecture, so increasing the bidder servers in large and geographically dispersed basic auction units is likely to be needed in the real distributed auction system.

The number of bidder clients was increased gradually from 100 to 200 in increments of 10 and were always equally distributed between two different bidder servers. These initialisations could be changed and increased to more bidder clients and bidder servers and not equally distributed for implementing the real hierarchical architecture in distributed auction system in the real Internet auction world.

Further work can be carried out on the above suggestions. Furthermore, changing the Middleware technology, CORBA platform to EJB, Enterprise JavaBeans, or Microsoft.Net , in order to develop implementation and performance results could also be considered as future works.

Bibliography

- [96] *Going ... Going ... Gone! : A Survey of Auction Types*, 1996 available at: accessed:
- [99] *Online Bargain Hunting: The world wide shopping mall*. Time Magazine Europe, 1999.
- [Amoroso'03] Amoroso, A. and Panzieri, F. *A Scalable Architecture for Responsive Auction Services Over the Internet*, 2003.
- [Baldwin'99] Baldwin, R. "On-line Auctions: Just Another Fad?" IEEE Multimedia 6(3): 12-13, 1999.
- [Banatre'86] Banatre, J-P., Banatre, M., Lapalme, G. and Ployette, F. "The Design and Building of Enech'ere, a Distributed Electronic Marketing System." Communication of the ACM 29(1): 19-29, 1986.
- [Bernstein'87] Bernstein, P.A. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
- [Broughton'00] Broughton. *Devious sellers make a killing in net auction*. Telegraph, 2000.
- [Brown'96] Brown, N. *Distributed Component Object Model Protocol - DCOM/1.0*, 1996.
- [Clausing'00] Clausing, Jeri. *Government fights spread of online auction fraud*. The New York Times, 2000.
- [Collins'01] Collins, J., Corey, B., Ghini, M. and Mobasher, B. *Decision Processes in Agent-Based Automated Contracting*. IEEE Internet Computing: 61-72, 2001.
- [D'efago'98] D'efago, Xavier., Schiper, Andr'e. and Sergent, Nicole. *Semi-Passive Replication*. Seventeenth IEEE Symposium on Reliable Distributed Systems, 1998, IEEE Computer Society.
- [Dawe'00] Dawe, L. *Hammer out a deal at an online auction*. Times, 2000.
- [Deutsch'98] Deutsch, . and Claudia, H. *Businesses explore online auction for equipment parts*. The New York Times, 1998.
- [Ezhilchelvan'01] Ezhilchelvan, P.D., Khayyambashi, M.R., Morgan, G. and Palmer, D. *Measuring the Cost of Scalability and Reliability for Internet-Based, Server-Centered Applications*. 6th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2001), Rome, Italy, 2001, IEEE Computer Society Press 2001.
- [Ezhilchelvan'01] Ezhilchelvan, P.D. and Morgan, G. *A Dependable Distributed Auction System: Architecture and an Implementation Framework*. 5th International Symposium on

- Autonomous Decentralized Systems (ISADS 2001), Dallas, Texas, USA, 2001, IEEE Computer Society Press 2001.
- [Fay-Wolfe'00] Fay-Wolfe, V., DiPippo, L. C., Cooper, G., Johnston, R., Kortmann, P. and Thuraisingham, B. "*Real-Time Corba.*" IEEE trans. on Parallel and Distributed Systems **11**(10): 1073-1089, 2000.
- [Felber'02] Felber, P. and Narasimhan, P. Reconciling Replication and Transactions for the End-to-End Reliability of CORBA Applications. Distributed Objects And Applications 2002, UC Irvine, 2002.
- [Feldman'00] Feldman, S. "*Electronic Marketplaces.*" IEEE Internet Computing: 93-95, 2000.
- [Ferber'96] Ferbo, . Lecture Notes : Auctions, 1996 available at: accessed:
- [Fischer'85] Fischer, M.J. , Lynch, N.A. and Paterson, M.S. "*Impossibility of Distributed Consensus with one faulty Process.*" Journal of the ACM Vol. **32**, No. **2**: 374-382, 1985.
- [Franklin'66] Franklin, M.K. and Reiter, M.K. "*The Design and Implementation of a Secure Auction Service.*" IEEE Trans. on Software Engineering **22**(5): 302-312, 1966.
- [Friedman'91] Friedman, D. and Rust, J. The Double Auction Market: Institutions, Theories, and Evidence Proceedings of the Workshop on Double Auction Markets, Department of Economics, Vanderbilt University, Nashville, TN 37235, 1991.
- [Geun Lee'98] Geun Lee, H. "*Do Electronic Marketplaces Lower the Price of Goods?*" Comm. of the ACM **41**(1): 73-80, 1998.
- [Greenwald'01] Greenwald, A. and Stone, P. *Autonomous Bidding Agents in the Trading Agent Competition.* IEEE Internet Computing: 52-60, 2001.
- [Grossman'99] Grossman, Lev. *And to think that I saw it on eBay.* Time Digital, 1999.
- [Hardjono'96] Hardjono, T. and Seberry, J. Strongboxes for Electronic Commerce. 2rd USENIX Workshop on Electronic Commerce Proceedings, 1996.
- [Harkavy'98] Harkavy, M. , Tyger, J.D. and Kikuchi, H. Electronic Auction with Private Bids. 3rd USENIX Workshop on Electronic Commerce Proceedings, 1998.
- [<http://www.agorics.com/Library/Auctions/auction5.html>]
- <http://www.agorics.com/Library/Auctions/auction5.html>, available at: accessed:
- [<http://www.crime-research.org/articles/Wahab1>] <http://www.crime-research.org/articles/Wahab1>, available at: accessed:

- [<http://www.orbacus.com/products/orbacus.html>] <http://www.orbacus.com/products/orbacus.html>,
available at: <http://www.orbacus.com/products/orbacus.html> accessed:
- [Huhns'99] Huhns, N., Michael, . and Vidal, Jos´e. *On-line Auctions*. IEEE Internet Computing: 103-105, 1999.
- [Kaufman'99] Kaufman, Leslie. and Harmon, Amy. *Buyers flock to online auction*. The New York Times, 1999.
- [Knight'00] Knight, Jenny. *Buyers beware on the Internet*. Telegraph, 2000.
- [Kumar'98] Kumar, M. and Feldman, S.J. *Internet Auctions*. 3rd USENIX Workshop on Electronic Commerce Proceedings, Boston (MA),, 1998, IEEE.
- [Lin'03] Lin, N. and Shrivastava, S.K. *System Support for Small-scale Auctions*. Med-Hoc Net 2003 Workshop, Mahdia, Tunisia, 2003.
- [Lind'91] Lind, Barry., Plott, . and Charles, R. "The Winner's Curse: Experiments with Buyers and with Sellers." American Economics Review **81**(1): 335-346, 1991.
- [Little'99] Little, M.C. and Shrivastava, S.K. "Integrating Group Communication with Transactions for Implementing Persistent Replicated Objects." Advances in Distributed Systems, **1752**: 238-253, 1999.
- [Lucking-Reiley'99,] Lucking-Reiley, D. *Auction on the Internet : What's being auctioned, and how?*, Department of Economics, Vanderbilt University, Nashvil,, 1999,.
- [Macedo'95] Macedo, R.J.A. *Fault-Tolerant Group Communication Protocols For Asynchronous Systems*. School of Computing Science,. Newcastle upon Tyne, University of Newcastle upon Tyne, 1995.
- [Maxemchuk'01] Maxemchuk, N. F. and Shur, D. H. "An Internet Multicast System for the Stock Market." ACM Trans. on Comp. Sys. **19**(3): 384-412, 2001.
- [McGrane'00] McGrane, Sally. *Auction creep into all kinds of sites*. The New York Time, 2000.
- [Morgan'99] Morgan, G. *A Middleware Service for Fault-Tolerant Group Communications*. School of Computing Science. Newcastle upon Tyne, University of Newcastle upon Tyne, 1999.
- [Morgan'99] Morgan, G., Shrivastava, S.K., Ezhilchelvan, P.D. and Little, M.C. "Design and Implementation of a CORBA Fault-tolerant Object Group Service." Distributed Applications and Interoperable Systems: 361-374, 1999.

- [Mpoeleng'05] Mpoeleng, D. *From Crash Tolerance to Byzantine Tolerance: Fail Signaling Dependable Distributed Systems*. School of Computing Science. Newcastle upon Tyne, University of Newcastle upon Tyne, 2005.
- [Mullen'98] Mullen, T. and Wellman, M.P. *The Auction Manager: market Middleware for Large-Scale Electronic Commerce*. 3rd USENIX Workshop on Electronic Commerce, Boston (MA), 1998.
- [Panzieri'99] Panzieri, F. and Shrivastava, S.K. *On the Provision of Replicated Internet Auction Services*. In Proceedings of the 1999 IEEE Workshop on Electronic Commerce (WELCOM '99), part of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS '99), 1999.
- [Peng'98] Peng, C. , Pulido, J.M., Lin, K.J. and Blough, D. *The Design of an Internet-based Real Time Auction System*. 1st IEEE Workshop on Dependable and Real-Time E-Commerce Systems, Denver (CO), 1998.
- [Rachlevsky-Reich'99] Rachlevsky-Reich, B., Ben-Shaul, I., Chan, N.T., Lo, A. and Poggio, T. "GEM: A Global Electronic Market System." Information Systems 24(6): 495-518, 1999.
- [Rassenti'92] Rassenti, K. A. and Smith, V. L. "Designing Auction Institutions: Is Double-Dutch the Best?"" , 1992.
- [Robinson'85] Robinson, Marc S. "Collusion and the Choice of Auction." RAND Journal of Economics 16(1): 141-145, 1985.
- [Rosenberry'92] Rosenberry, W. *Understanding DCE, OFC Distributed Computing Environment*, Addison Wesley, 1992.
- [Samret'00] Samret, N., Liao, R. R.-F., Campbell, A. T. and Lazar, A. A. "Pricing Provision and Peering: Dynamic Markets for Differentiated Internet Services and Implications for Network Interconnections." IEEE J. on Select. Areas in Comm. 18(12): 499-513, 2000.
- [Sandholm'00] Sandholm, T. and Huai, Q. *Nomad: Mobile Agent System for an Internet-based Auction House*. IEEE Internet Computing: 80-86, 2000.
- [Schwartz'98] Schwartz, Evan I. *At online auctions, goods and raw, deals*. The New York Times, 1998.
- [Sun Microsystems'97] Sun Microsystems. *Java Remote Method Invocation Specification*. JDK 1.1, 1997.

- [Team'01] Team, TAC. *Design the Market Game for a TRading Agent Competition*. IEEE Internet Computing: 43-51, 2001.
- [The Object Management Group'95] The Object Management Group. *The Common Object Request Broker Architecture and Specification*, 1995.
- [Turban]Turban, Efraim. "Auction and Bidding on the Internet: An assessment", *International Journal of Electronic Markets*." Vol. 7 No. 4.
- [Vickery'61] Vickery, David. "Counter Speculation, Auction, and Competitive Sealed Tenders." Journal of Finance: 9-37, 1961.
- [Welkum'02] Welkum, G. and Vossen, G. *Transactional information Systems*, Morgan Kaufman, 2002.
- [Wellman'98] Wellman, M.P. and Wurman, P.R. *Real Time Issues for Internet Auctions*. First IEEE Work. on Dependable and Real-Time E-Commerce Systems (DARE-98), Denver (CO), 1998.
- [Wice'99] Wice, Nathaniel. *ebay keeps crashing*. Time Digital, 1999.
- [Wrigley'97] Wrigley, C. "Design Criteria for Electronic Market Servers." EM-Electronic Markets, 7(4): 12-16, 1997.
- [Wurman'01] Wurman, P.R. *Dynamic Pricing in the Virtual Market*. IEEE Internet Computing: 36-42, 2001.
- [Yuan'98] Yuan, Y. , Rose, J.B. and Archer, N. "A Web-Based Negotiation Support System." EM - Electronic Markets 8(3): 13-7, 1998.