TRADING SYSTEM DESIGN AND IMPLEMENTATION IN OCEAN (OPEN COMPUTATION EXCHANGE AND ARBITRATION NETWORK)

By

SRIRAMKUMAR NALLANCHAKRAVARTHULA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002

TO MOM, DAD, MY SISTERS MADHAVI AND SRIDEVI

ACKNOWLEDGMENTS

I must first and foremost express my deepest appreciation to my committee chair, Dr. Michael Frank, for presenting me with the opportunity to work on such a fascinating project. His continuous guidance played a significant role in bringing this project to fruition. I would also like to thank my other committee members, Drs. Stanley Su and Joachim Hammer, for their meticulous review of this thesis.

I would also like to express my gratitude to all members, past and present, of the OCEAN research project. Through painstaking collaboration, their contributions produced meaningful results that appear in this thesis. And on a personal note, I must express my most sincere thanks to our project lead, Erwin Jansen, whose unending support and encouragement were a constant source of inspiration throughout this experience.

TABLE OF CONTENTS

page

LIST OF FIGURES

Figure page

Abstract of Thesis Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Master of Science

TRADING SYSTEM DESIGN AND IMPLEMENTATION IN OCEAN (OPEN COMPUTATION EXCHANGE AND ARBITRATION NETWORK)

By

SRIRAMKUMAR NALLANCHAKRAVARTHULA

December 2002

Chairman: Michael P Frank
Major Department: Computer and Information Science

The growth of the Internet has provided us with the largest network of interconnected computers. Many of these systems are often under-utilized and are not used in compute-intensive tasks. The time-zone difference accentuates this fact as busy hours and idle hours tend to be different in different zones. The computing world can potentially be revolutionized if we can fully utilize these systems. Users who find it difficult to allocate resources for their massive applications can use these remote-computing resources via the Internet and this would allow us to perform the computation in cooperation with many processors worldwide. The owners of these processors can be motivated to participate in such a computation by paying for their CPU time. This idea led to the development of computational markets where the computing power can be traded as a commodity. OCEAN (Open Computation Exchange And Arbitration Network) is such a computational market which aims to support the automated commercial buying and selling of dynamic distributed computing resources over the Internet. Applications within

OCEAN can purchase and utilize remote distributed computational resources on demand, while compensating the providers of these resources via a free-market mechanism.

This thesis discusses how a trade occurs in OCEAN between potential trading partners. In particular it discusses how buyers find potential sellers who offer their required computational resources and how buyers make a contract with sellers to procure those resources. In order to guarantee maximal cross-platform operability the OCEAN system was implemented in Java and hence can span across many architectures and operating systems. The trading system in OCEAN is implemented using XML and SOAP technologies.

CHAPTER 1
INTRODUCTION

Computational Markets and OCEAN

It has been determined by numerous sources that many of the systems that are connected to the vast Internet are idle most of the times and are not involved in compute-intensive tasks. People who do not have necessary computational resources can use this idle time for executing their jobs. The owners of these systems can be motivated to participate in remote computations by paying for their CPU time. This led to the idea of developing computational markets where CPU time is the tradable commodity. OCEAN is such a computational market that provides services to discover and procure resources across the Internet using an open market mechanism. It was our belief that many of the existing computational grids can use the services provided by OCEAN to trade computational resources to perform Internet-based distributed computing. OCEAN research group aims to produce an automated trading system that can buy and sell dynamic distributed computing resources over the Internet. This thesis discusses the design and implementation of a trading system built for Internet-based distributed computational markets like OCEAN. In order to guarantee maximal cross-platform operability the OCEAN trading system was implemented in Java. This system, which was built using XML and SOAP technologies in Java, can span across many architectures and operating systems.

An Overview of Trading in OCEAN

Users of OCEAN system are classified as either buyers or sellers. The buyers typically have a computation that needs to be performed, while the sellers have access to idle resources that can execute the computation in question. Buyers who don't have the necessary computational resources trade with Seller to use their idle resources. When buyers and sellers have made a contract, the buyer's program requiring the computational resources will be transported to the seller's machine, and the computation will be performed there, the results of which are sent back to the buyers. The sellers are compensated for the resources they provide and the buyers are able to perform the task without purchasing expensive hardware. The open market in OCEAN provides an environment in which these interactions may take place in an automatic way.

The host allocation in OCEAN is based on machine profiles describing the computational resources they provide and a credit-based mechanism is used for compensating the sellers. If we want to distribute a task to a different node on the OCEAN network we must be able to specify the requirements of our task. That means a buyer must be able to extract the minimum requirements for its task to execute. This information will be used to find all the potential Sellers' resources that are compatible with the buyer's requirements.

Buyers prepare documents identifying their requested computational resources such as CPU, Memory etc, the maximum price they are willing to bid for these resources, the Grid structure they use to transfer their job, payment type they wish to use and contract terms such as compensation they will make for not honoring the lease etc. These documents stored at Buyers' machines are later used to prepare search requests and contract requests during trading process. In the same way Sellers also identify the

minimum price they are willing to accept for lending their computational resources, payment type acceptable to them, the Grid structure they use to receive the Buyers' tasks etc which are stored in documents at Sellers' machines.

Search requests are generated from buyers' resource requirements and are propagated in OCEAN network to search for matching Sellers. Sellers report buyers of a match if their actual resource descriptions match with the buyers' search requests. Buyers can then choose a best seller from the list of matches they receive and finalize the deal.

For choosing a best Seller the OCEAN group identified three possible ways:

- Based upon the resource description in XML document we assign a quality to the resource. The matches that are received can then be ordered on these numbers. This will take a limited amount of overhead but may not produce accurate result. We think this scheme is enough for simple jobs.
- We can include the results of the seller's machine on a well-known benchmark. This can be used by the buyer to get an indication of the quality of the resources of the seller. But there is overhead of forming these benchmarks. We propose to use this scheme for complicated jobs.
- Buyer can run a probe on Seller's machines to get accurate indication of the capabilities of the resources offered. The probe can be tuned to represent the task at hand. An example of this would be a probe that renders only 1 frame of a movie scene. The task is clear representation of the entire, but will consume only a limited amount of time. This gives an accurate result but there is too much overhead of running these probes on all sellers' machines. We propose to use this scheme for extremely complicated jobs.

The current thesis implemented the first approach in ordering the Sellers. Standard Benchmark results can be easily incorporated in the search requests. Once we have an ordering a buyer can contact the best seller and make a deal with him. Both buyer and seller can then sign a contract, which specifies price, payment, and contract terms, which is logged to a central accounting system for payment and reference.

Structure of the Thesis

In this section I will briefly describe the contents of each chapter.

- In chapter 2, I will give an overview of OCEAN network structure, OCEAN Trading System Architecture and its GRID neutral Architecture.
- In chapter 3, I will discuss the overview of trading process in OCEAN. Here I will discuss how buyers find the resources they need and how they procure these resources by making contract with a seller.
- In Chapter 4, I will discuss design and implementation of Trading System. Here I will discuss the sample implementation of a Trading system in java using XML and SOAP technologies.
- In chapter 5, I will show the run time GUI and messages of the simple system made using the above design and discuss a little bit about the testing of the system.
- In Chapter 6, I will give a conclusion and identify future improvements that can be made in the current system.
- At the end I will give the appendix, references and documentation.

CHAPTER 2
OCEAN SYSTEM ARCHITECTURE

Peer-To-Peer system

The task of locating a suitable Seller in OCEAN network can be done in three ways:

(1) A client-server type of architecture for OCEAN would employ a powerful server to run a collection of server processes to perform the trading with buyers and sellers sending a series of requests to perform a trade. The central server can provide a directory and event service supporting the discovery of trading partner. But this is not feasible in OCEAN system as the potential volume of users that are able to participate is enormous since anyone with a computer may join the OCEAN system. The scalability becomes a huge factor in such a client-server design with so many clients. The performance of the system will be affected as the number of clients increase and server itself becomes a single point of failure for the whole system.

(2) We can employ a distributed network of market servers coordinating with each other to implement the market mechanisms. This approach definitely improves the fault tolerance over client-server approach but even this doesn't achieve maximum scalability. If the number of users increased dramatically which is possible in OCEAN system because of its nature, it may so happen that there are not enough market servers to keep the market operating efficiently.

(3) The last approach is by using peer-to-peer based architecture. In this approach there is no concept of central registry and peer-to-peer messaging is used to locate Trading partners. We feel this approach ideally suits the OCEAN system. It is not easy to co-ordinate actions in pure peer-to-peer based systems as they lack central registry but the scalability and the robust features they provide outweigh this disadvantage in OCEAN System. Any peer node in OCEAN network can perform any function that is required for trading.

OCEAN Trading System Architecture

The below figure depicts a typical Trading System architecture at an OCEAN node.

An overview of each component follows:

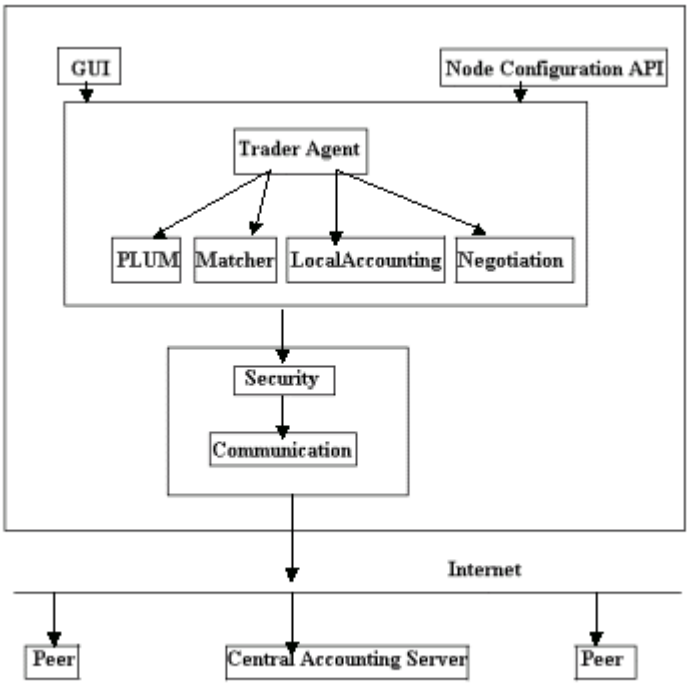

Figure 2-1. OCEAN Trading System Architecture. Trader Agent uses the services of PLUM, Matcher, Negotiation and Local Accounting System to perform a trade. These OCEAN services use the security for authentication and communication for messaging. This figure also shows the Central Accounting server. All transactions that happen between peers in OCEAN are logged to this Central Accounting Server.

GUI. This GUI is used to prepare the Trade Proposal documents. This is just one way of preparing the XML documents. We also provide a Trade Proposal API that can be used to prepare the documents. The current system uses this GUI to prepare XML documents for testing purpose.

Node configuration API. This API is used to tune various parameters that affect the trading process in OCEAN. All parameters that can affect trading process like the Number of Legs that Trade Proposal can travel before they die, the time the Agents wait to accumulate matches before they start negotiation etc can be modified using this API.

Trader Agent. The Trader Agent acts on behalf of the human users to perform the whole trading process. The Trader Agent can be used either as a buyer or as a seller.

Buyers and sellers submit Trade Proposal Documents prepared using the above GUI or by some other means to these agents. Agents then prepare Bids/Offers from these documents and use the services of Matcher, PLUM, Negotiation and Local Accounting Component to perform the trading.

Matcher. All OCEAN Nodes contain matchers but they are active in only Seller Nodes. The matcher at each seller performs the matching process. It verifies whether the Search requests that arrives from buyers match with its actual resource description.

Peer list update manager component (PLUM). PLUM component determines a list of peer nodes in a priority queue like data structure, with which a node will communicate. The PLUM maintains addresses of other peers and their associated priority information. PLUM periodically updates its peer list by sending queries to other peers.

Security component. Security is a major concern in computational markets. Both the parties to a transaction should have confidence in the integrity of the documents. The Security Component performs the authentication of parties by providing digital certificates. All sensitive messages like Contracts are digitally signed by the parties involved in trade.

Communication component. The Communication Component is responsible for any communication to and from any OCEAN node. All Trader Agents at an OCEAN node use their Communication Component for sending and receiving SOAP messages.

Local accounting component. Local accounting component logs the contracts that are made by the user to a Central Accounting Server that transfers the funds between buyers and sellers accounts. It also maintains the node's own transaction history.

Central accounting server (CAS). This is not a part of OCEAN Node. Every trader on the OCEAN system has an account on the CAS. CAS maintains all the accounts like a conventional bank. All contracts made in OCEAN are logged in CAS database. CAS uses this information in database to compensate all the sellers that are involved in a contract. Many transactions may be too small to be worth executing on traditional financial networks. Such micro-payments are accrued to or deducted from the trader's account each time a successful transaction takes place, without accessing the external financial networks. This account information is used to make real world transactions with the existing financial networks to debit or credit the traders only periodically or when the balance in the trader's account exceeds a particular limit.

The OCEAN also provides task mobility and communication using a separate GRID architecture. The OCEAN Grid is completely independent of the OCEAN Trading System. The OCEAN Grid consists of four components and they are Task Migration Component, Security Component, Communication Component and Naming Component. Upon successful negotiation of a contract for the execution of a computation, the Task Migration component spawns or migrates client tasks to remote nodes in a secure fashion, using services provided by the Security and Communication components. The Security component's role in this context is to ensure a safe environment within which an OCEAN task executes, incapable of unexpected or hostile activity while running on an OCEAN host. The Communication component is primarily responsible for data delivery to the other OCEAN node(s), and configurable through the Node Configuration / Maintenance Operator Interface. The Naming Component is used to locate a task that is transferred to

at a particular OCEAN Node. We plan to provide a simple Job Maker API through which a programmer can write OCEAN aware applications easily.

Existing Grid Architectures and Compatibility with OCEAN

There are many projects that provide infrastructures for distributed computing like OCEAN. GriPhyN, Globus, Tryllian, Beowulf, SETI@Home, Distributed.Net, Gnutella, IBM Aglets, Sun Grid Engine, XtremWeb, Condor, Legion, and Ubero are all examples of distributed computing systems that currently lack a market mechanism to compensate resource providers. United Devices, Entropia, Parabon, Porivo are examples of ventures that do sell distributed computing resources, but they are closed markets. The markets in these systems are controlled by those companies themselves and often may not give the best possible compensation to the resource provider and they don't provide a free, open, standard API that any developer can target and test like OCEAN. These systems are also less flexible than OCEAN, since resources must be typically leased in advance, and cannot be purchased on-demand dynamically by arbitrary applications. They are also less scalable than OCEAN, because of their centralized architecture unlike OCEAN that uses peer-to-peer architecture to locate trading partners. OCEAN is being designed to be compatible with all the Grid technologies in the market that offer different solutions to certain basic problems, such as transfer of code to new hosts, and communication between distributed components of an application. Developers using any of the above said GRID technologies can access OCEAN to find trading partners, perform trading and for reimbursing sellers and then use their own GRID technology for task migration.

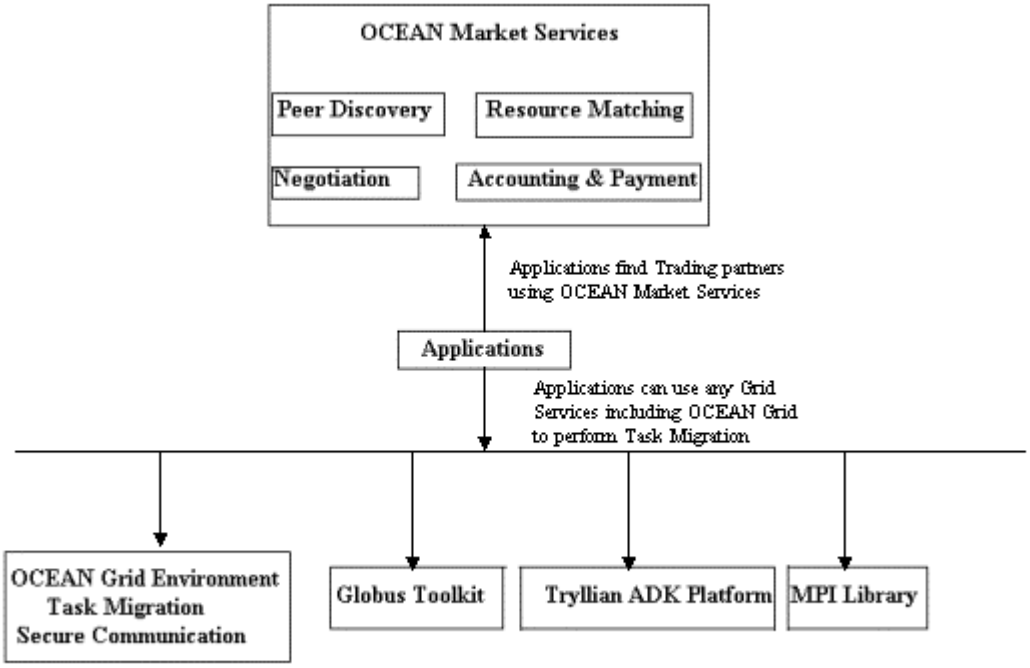

Figure 2-2. OCEAN Grid-Neutral architecture. At the top of the layer are the OCEAN services that make up the open market to trade the computational resources. Any Grid or Mobile-agent application can use this OCEAN service to find the potential trading partners who can run their job and then use any of the Grid services including the OCEAN Grid to transfer the job to that place for remote execution. The OCEAN's Accounting and Payment System compensates all the intermediate sellers in the path.

CHAPTER 3
TRADING SYSTEM OVERVIEW

Registration

Any client interested in using the OCEAN trading system services must first register with the Central Server maintained by the OCEAN authority by filling out a registration form on which he or she must provide his information such as real name, postal address and email address, and a desired username and password. During installation of the OCEAN software, a client, on behalf of the local user will send the user's information to the Authentication Server and requests to be registered. The server will check whether the requesting user has already been registered or not. If not, the Authentication Server will return the client's unique identity. After successful registration the Unique ID sent by the server is stored locally and is used for identification purposes in further communications in OCEAN system. After registration users must open an account in Central Accounting System, which serves in a way similar to a conventional Bank Account. Once this registration is completed, the client becomes a member of the OCEAN trading network.

Generating Trade Proposal Documents from requirements

Buyers who want to execute their task in OCEAN network should first specify the requirements for their task. Buyers should indicate their desired machine profiles which can execute their task. The parameters that buyers specify are:

(1) Desired minimum computational resource requirements for his/her application. The computational resources that are specified include the CPU Power, Fast/Primary Memory, Slow/Secondary Memory, Operating system, and Software Library etc.

(2) Type of task they want to execute. At present OCEAN supports only Java Tasks. We plan to support .NET tasks in future.

(3) The maximum amount of price/credit per second he/she is willing to pay for the task.

(4) The underlying Grid Environment they will use to transfer the code to Sellers.

(5) The schedule time and the minimum amount of usage time for the jobs without further notification.

(6) The Payment type and mode to be used.

(7) The credit granted/demanded for not honoring the agreement.

(8) The average performance of the system (determined from benchmarking certain performance characteristics such as integer and float point arithmetic and their network connection speed etc) that is acceptable. The current system doesn't support this and we plan to include this in future versions.

In the same way Sellers Task and Grid Information, Price and Payment Terms, Benchmark Performance etc. Trade proposal documents will be prepared from these specified parameters. These documents are used later by the Trading System to create bids and offers that are used during matching and negotiation.

It is obvious that Buyers and sellers should express the requirements in their trade proposals in a standard format so that all participants can understand. Both the parties must agree on many discrepancies in specifying these documents before they can be matched by any matching system. So we need some standardization. The language used to specify these documents should also be extensible to specify more parameters that may be needed for some computations. We need expressive and powerful data definition languages to deal with this standardization and complexity.

The OCEAN group proposed to use standardized XML documents for this purpose. By standardizing the languages using XML to describe buyers and sellers, it will be possible for markets and users to benefit. Only in this way a computational market can be

truly “open”. Use of XML documents to describe trade proposals of buyers and sellers also offers some distinct advantages. First of all, XML documents are plain text, and thus are platform independent. Any platform that can read ASCII text has the ability to read an XML document. This is critical for computational markets like OCEAN as many different computing platforms may be used in a market. XML provides a way of conveying metadata. So when a document is written using XML all the data and information describing the data is in one place. And also XML is designed to run over http connections on the Internet which makes it an obvious choice for e-business. Also we have plenty of tools to parse and retrieve information from XML documents.

XML Schemas were developed to provide languages for expressing trade proposals and contracts in the OCEAN system. These schemas provide the constructs used to describe the documents in OCEAN market. These constructs include describing the formats for expressing Computing Resources, Performance Scores, Grid Technologies Used, Price, Payment Terms and Contract issues for Trade Proposals. The benefit of using schema specification is that schemas are easily extensible and this feature contributes to the openness property of OCEAN. From applications point of view the major benefit of using a schema to define the structure of documents is that applications, which depend on the data, have a well-established set of rules to follow when generating or parsing the XML documents. For example in OCEAN system GUI to create these XML documents or the agents, which parse these XML documents, know how a document is structured.

The generation of XML documents to describe the trade proposals and contracts should require minimal effort on the part of users who use the OCEAN system. In general

humans do not easily write XML and OCEAN Documents are no exception. We should provide an API to generate these documents which should be called by user-friendly GUI interfaces. All of the API implementations to generate XML documents in this thesis are done using JDOM and the GUI is provided using Java swing API. JDOM is a Java-based solution for accessing, manipulating, and outputting XML data from Java code. JDOM [25] is both Java-centric and Java-optimized. It behaves like Java, it uses Java collections, it is completely natural API for current Java developers, and it provides a low-cost entry point for using XML. While JDOM interoperates well with existing standards such as the Simple API for XML (SAX) and the Document Object Model (DOM), it is not an abstraction layer or enhancement to those APIs. Rather, it seeks to provide a robust, lightweight means of reading and writing XML data without the complex and memory-consumptive options that current API offerings provide.

Matching Phase

Propagating Search Requests to Find Sellers

We need to provide a mechanism so that buyers can locate Sellers in OCEAN network. For the reasons discussed in section 2.1 the OCEAN system is designed as a peer-to-peer system. There is no central registry and the only way the Buyers can locate the Sellers is by sending messages to their peers. Buyers send Search Messages to their peers, which are propagated in OCEAN in a peer-to-peer fashion. OCEAN follows query-flooding scheme for broadcasting of these Search messages. Buyers ignore these Search messages. Seller node sends match messages back to buyers if they match with search requests.

For optimal maintenance and generation of peer list we need another sophisticated software component. This component is termed Peer List Update Manager (PLUM),

which generates and maintains a list of peers. The current system has a basic implementation of this component. All the users get an implementation of PLUM component when they download the services of OCEAN system during registration time. The PLUM component maintains a certain pre-determined number of peer's information in its persistent storage at any time. It is not a good idea to broadcast every search request to every node in the OCEAN network as it leads to so much traffic. To prevent this a parameter called "Number Of Legs (NOL)" is included in the search messages. This parameter controls the number of hops a trade proposal can travel in OCEAN network. Node operators can configure this parameter through Node Configuration API. When a node receives a search message it decreases the number of legs by one and only if it is not zero it propagates the search message. So by setting NOL to maximum number of hops to be traveled we can control the depth of propagation in the OCEAN network. Since it is possible to receive redundant messages in a system like this we assign a unique Identifier for each proposal so that matching systems can ignore the duplicate messages. This identifier is generated by concatenating system time in milliseconds to the Unique Identifier generated from the registration process. The Search Messages add the path information as they travel to various nodes. This path is used later to reimburse all the nodes along the path for propagating the search requests. This reimbursement scheme provides an incentive for all node operators to tune their nodes for optimal distribution of these Search messages.

Matching of Proposals

We say a Buyer's requests match with Seller's profile when Bids and Offers prepared from Buyers' and Sellers' Trade Proposal documents match with each other. We say a Bid and Offer are matched when all the fields in Bid match with respective

fields in Offer. In particular we say a Bid is matched with Offer if they are resource compatible i.e. when the computational requirements and Grid Environment specified by buyer in his/her bid are matched with those specified by seller in his/her Offer, Price compatible i.e. when the price terms specified in Bid agree with price specified in Offer, Contract Terms Compatible i.e. when the payment terms and other contract issues specified in Bid agree with those specified in Offer. In current OCEAN system based on resource listing in XML documents it is assumed that Buyers list all the minimum necessary rigid computational requirements in their Bids. Sellers whose offers provide those minimum necessary resources match with them. That is a Bid "B" is said to be resource compatible with a Offer "O" if every resource requirement 'X' stated in B is less than or equal to every resource requirement 'X' stated in O in their respective unit of measurements. They are price compatible if Bid Price "B.P" is at least some 'DIF'% greater than Offer Price "O.P". This DIF is specified by the Central Accounting System, which gets some percentage of the credit transferred between buyers and sellers. That is if ((B.P-O.P)/B.P * 100) >= DIF then B and O are said to be price compatible. Payment terms, Grid specification and some resources such as Operating System in Bid "B" should exactly match with those specified in Offer "O" otherwise they are not compatible. Only if a bid and offer are compatible in all these conditions they are returned as matched by the matching system.

Negotiation Phase

Sorting Matches

We need to define a function that can sorts the matches and finds the best seller for our purpose. This is not easy to design as the XML documents of trade proposals contain so many parameters and some of them are related to each other. To make the design

simple we require that buyers indicate the factors that are important for them in choosing an offer and the weights attached with each factor. Offers can then be sorted according to these weighted factors chosen by the Buyer. The current System only considers price and sorts all offers according to the increasing order of price assuming that the systems that pass the threshold requirement during matching phase are roughly same from performance perspective. The system can be made more accurate and reliable if we can provide performance profile of the machines. For this purpose we are planning to introduce the concepts of benchmarks and probes in later versions as said earlier in the introduction.

Making a Contract

Once the Agent at Buyer Side sorts all the matches received from various sellers the Contract Making phase starts. An optional negotiation can be done at this stage to negotiate on price. This phase is done in a one-to-one fashion unlike matching which is done in a distributed fashion. Buyers and Sellers exchange messages and make a final contract. The protocol for this is very simple and can be stated in four steps as:

(1) Buyer initiates the phase by sending the "Request For Contract" XML documents constructed from the price, payment type, contract terms specified in the original Trade Proposal Documents A Status element is attached to this XML document, which is changed by the Seller as Negotiation goes.

(2) Seller verifies this document to see whether the specified parameters match with its proposal. If everything in the document sent by the buyer match with seller's he/she updates the status element of the document to "Accepted" and sends it back to buyer. If something in the document doesn't agree he/she updates the status element of the document to "Rejected" and sends it back to buyer.

(3) The Buyer now will receive a document in which the status is either "Accepted" or "Rejected". If the status is "Accepted" he digitally signs the contract and sends it to Seller. If the status is "Rejected" the buyer will pick the next best seller in the list and repeats the process.

(4) If Seller receives signed contract document he /she also signs the contract and a deal is made. Both parties then indicate the Central Server about the deal through their Local Accounting component, which logs it for future payment and reference purposes.

Accounting Phase

The Local Accounting Component keeps track of the Clients transactional history. A database to record this transaction history is maintained. The Central Accounting System accrues or deducts all the micro payments from the trader's account each time a successful transaction takes place without accessing the external financial networks. This account information is later used to make real world transactions with the existing financial networks to debit or credit the traders. Though most of OCEAN is distributed, the CAS is centralized to simplify the financial operation of the system.

Exposing Trading System as SOAP Service

Our aim is to develop OCEAN as a web based service, which the existing GRID computing technologies like Globus, Parabon etc can use. As said earlier existing GRID applications like Globus and Parabon have their own communication, security and Code Migration systems. But these technologies lack a market-based architecture like OCEAN. These technologies can then use the web based market service of OCEAN to find potential trading partners and then use their own internal systems for communications and code migration. To achieve this end we can use technologies like SOAP, WSDL, and UDDI in OCEAN. We propose to use SOAP as the messaging mechanism. We want to design the different layers in OCEAN system as web services that are accessible via SOAP. In particular we want to design the communication component that receives search messages, match messages and contract messages for the purpose of the Matching and Negotiation as web based service that accepts SOAP Messages.

SOAP [18] is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. It is a fundamentally a stateless, one-way message exchanging paradigm, but applications can create more complex interaction patterns such as (request/response and request/multiple pattern) by combining such one-way exchanges with features provided by an underlying transport protocol and application-specific information. The Java API for XML Messaging (JAXM) [26] Optional Package is used in this thesis to send and receive document oriented XML messages. JAXM implements Simple Object Access Protocol (SOAP) 1.1 with Attachments messaging so that developers can focus on building, sending, receiving, and decomposing messages for their applications instead of programming low level XML communications routines.

Anatomy of a Trade at an OCEAN node

The various steps involved in a Trading Process at an OCEAN Node are:

(1) Buyers and sellers prepare their machine profiles using a user-friendly GUI forms. XML documents containing desired/required resources, Grid and Task types, Price and Payment terms etc are prepared by submitting the forms.

(2) Application asks Trading System for a location where it can execute its task by providing the resource document.

(3) Trading System creates a Bid object and then a search message from the document.

(4) Search Messages (SOAP based) are propagated in the peer-to-peer OCEAN network to different matching systems.

(5) The SOAP server on the peer receives these search requests. The peers (buyers/sellers) propagate the search messages. Seller peers also prepare Bids from

the search messages and enter them into their Matching System in addition to propagation.

(6) Seller nodes send match messages back to buyers that contain their entire resource description to Buyers if Bids entered in their matching systems match with their offers.

(7) The SOAP server on buyer side receives match messages from all the sellers that have a match. Offers are created from these messages and are submitted to the Local Negotiation System.

(8) The Local Negotiation/Contract-Making system then sorts the matches according to the criteria chosen by the buyer.

(9) Local Negotiation System then picks the best seller and sends a contract message consisting of the price and payment terms and asking sellers if they are ready to make a deal with it.

(10) The SOAP server on Seller side receives the contract message. It then verifies the items of contract and whether they agree with its terms. If so it signs the contract indicating its willingness to make a deal. Otherwise it rejects the contract.

(11) The SOAP server on Buyer side now receives either the rejected contract message or signed contract message. If it is a signed contract message it also signs the contract and sends it back to the seller for verification. If it is a rejected contract message it starts the contracting process with the next best seller in the sorted offers until a contract is made with one seller.

(12) If the contract phase succeeds both Seller and Buyer logs the contract in their Local Accounting System, which submits the contract message to the Central Accounting Server.

(13) If the contract phase fails buyer starts the trading process all over again by distributing search messages.

(14) The Central Accounting Server accrues all the micro payments and transfers the funds between buyer's account and seller's account. It also reimburses all the nodes in the path by inspecting the contract message and keeps some amount for itself.

Buyers and Sellers State Diagrams

In this section I will depict the Buyers and Sellers state diagrams. These diagrams show how the buyers and sellers perform a trading process in OCEAN system. It is assumed that Buyer has already prepared an XML document of the desired machine profile and it is stored somewhere in his hard disk. OCEAN software Agents that

performs trading for buyer loads these XML documents and for now this is done by passing an URL of the document in command line.

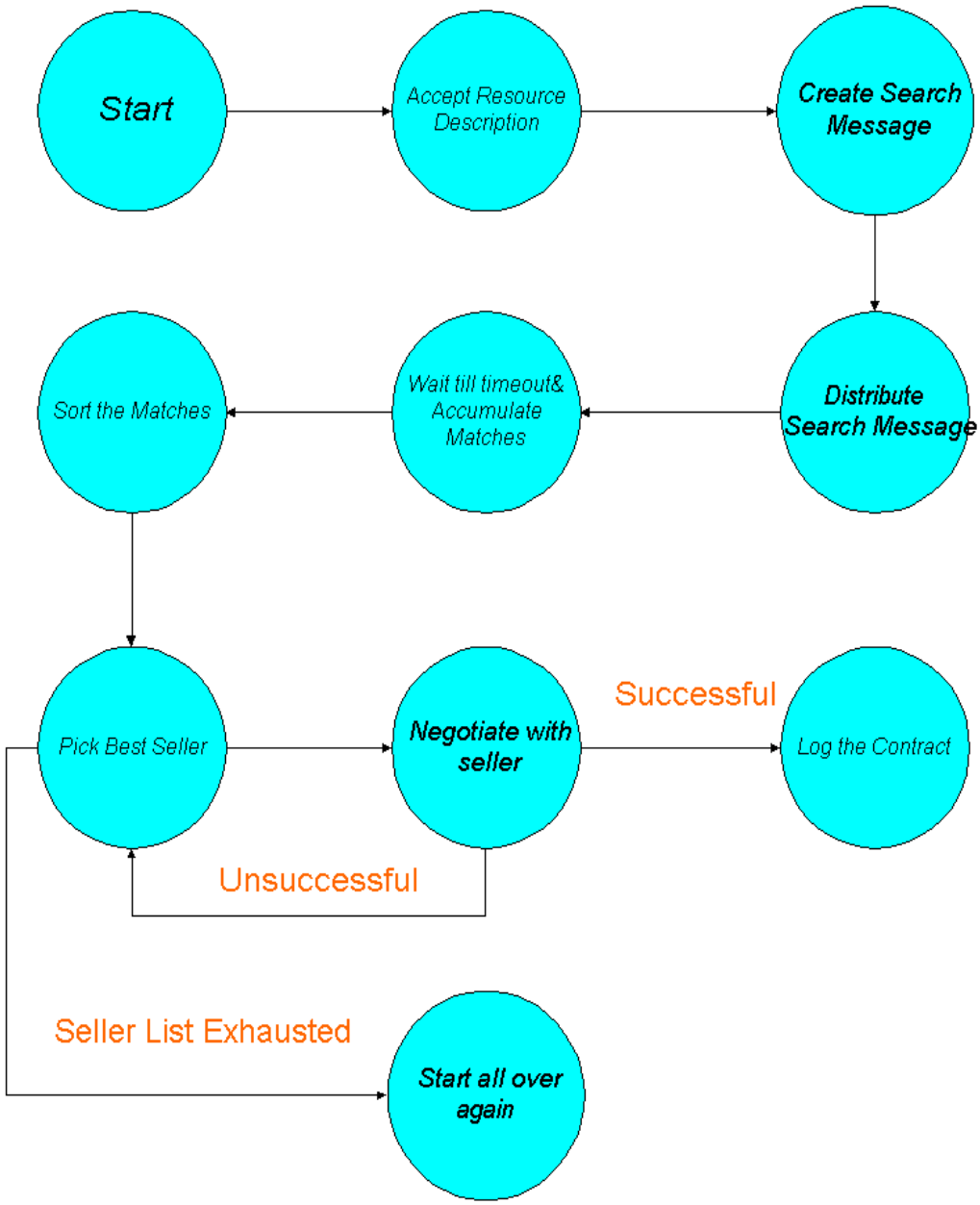

Figure 3-1. Buyers state diagram. Buyer Agents initially take a Uniform Resource Locator of the desired Machine Profile and create a search message from it. This Search Message is then propagated in a peer-to-peer fashion in OCEAN network. The agents then wait for some predetermined amount of time to accumulate all the matches from various sellers in network. After the time out they sort the matches and negotiate with the best Seller.

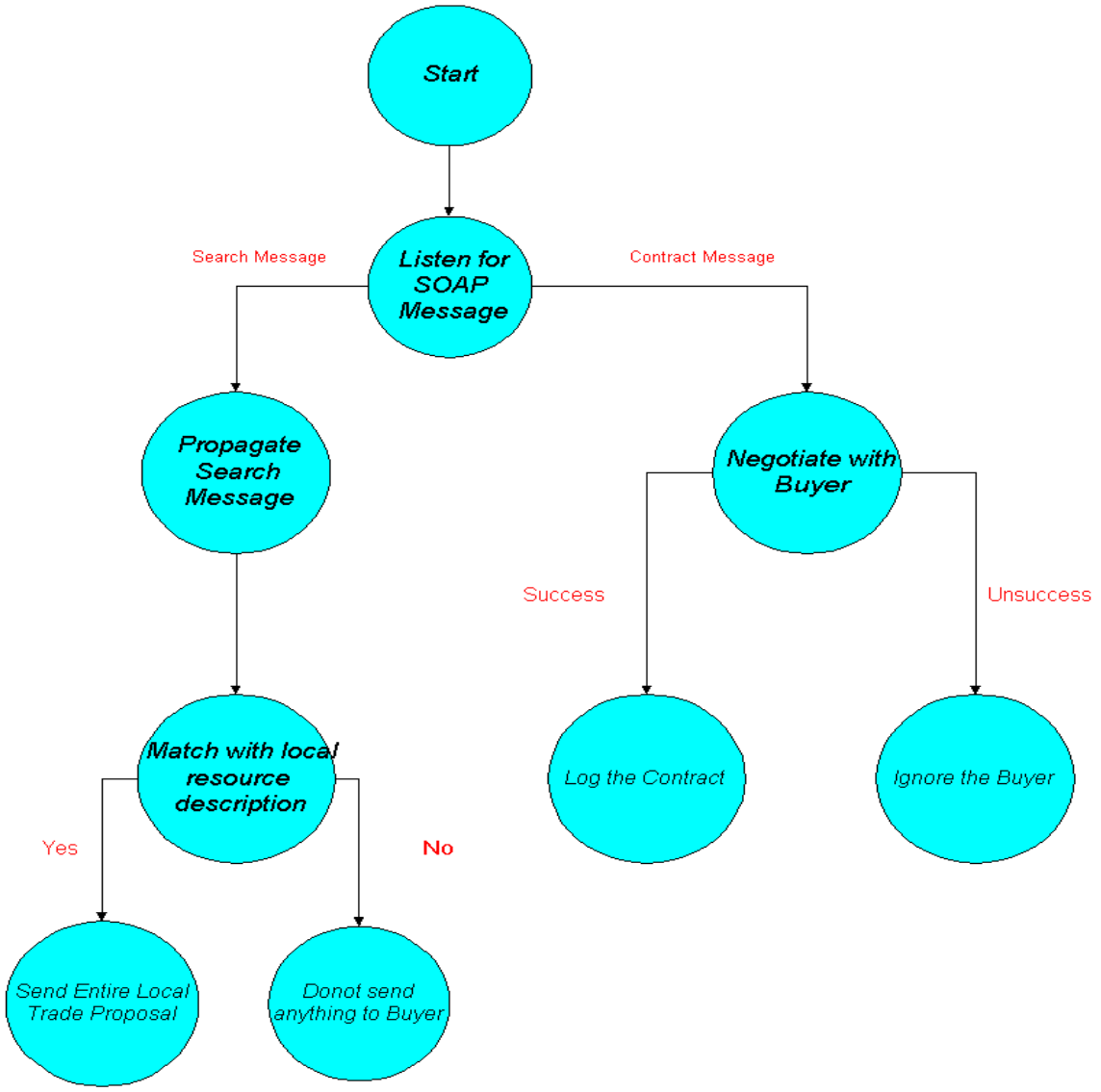

Figure 3-2. Sellers State Diagram. Sellers receive search messages from buyers that contain their desired machine profiles. Sellers send match messages back to buyers if their machine profile matches with the desired machine profiles of buyers. Sellers also receive Contract Messages if the buyers choose to negotiate with them. They negotiate with the buyers if they receive request for contract messages and log the contract if a successful deal is made.

CHAPTER 4
TRADING SYSTEN DESIGN AND IMPLEMENTATION

Major Components in Design

Trading in OCEAN is done in two phases. Those two phases are Matching Phase and Negotiation Phase.

(1) **Matching Phase:** This is the time when Bids submitted by buyers are distributed in the OCEAN network to find matching Offers. A Matching System is designed for this purpose, which matches the Bids and Offers.

(2) **Negotiation Phase:** This is the time when all the offers sent by sellers are sorted and a contract is made with the best seller. Logic to perform this ordering and making contract is coded in trading agents, which control the whole trading process. An optional Negotiation can be done for high-valued transactions before making a contract.

The Trading system in OCEAN includes the following major components:

(1) User-friendly GUI to generate Trade Proposals (Bids and Offers), which include required/actual machine profiles along with price and contract terms for buyers and sellers respectively.

(2) Matching Component where bids and offers are matched. The matching system matches the bids with the Seller's Offer. It decides whether the actual machine profile of seller can satisfy the machine profile requested by the buyer. It also considers minimum price difference and Grid compatibility issues.

(3) Trader Agent that controls all the trading process. Trader agents submit the Bids to remote Matching Systems at different Sellers using SOAP based Communication component. They sort the matches received from various sellers in the network according to utility function chosen by the buyers and make contracts with the best sellers. Trading agents are created for each and every Bid submitted to the OCEAN Market. These Trading agents control all the Trading process from submitting search messages to Matching Systems, sorting the Matches received from various Matching Systems and negotiating with each Seller to make a deal with the best possible seller. Agents register with Matching Communication Component for sending and receiving messages.

(4) Matching Communication Component is mainly responsible for propagating and receiving searches and matches respectively. All trading agents register with this

component for sending their search requests and for receiving match messages from various sellers in OCEAN network.

(5) Messaging part based on SOAP, which deals with the communication of Ocean Documents between the Peer nodes. The OCEAN SOAP messages are broadly classified into three types. (1) Search Messages that are sent by buyers to remote matching systems to find Sellers. (2) Match Messages that are sent by Sellers when their bids received from buyers match with its Offer. (3) Contract Messages that are exchanged between buyers and sellers during contract making phase.

(6) PLUM component, which generates and maintains a peer list. The current implementation has some stubs written to represent "Peer Structure" and "PLUM Table" in OCEAN Trading System. The "Peer Manager" is still being designed.

All an OCEAN user should do is to create the Trade Proposal Document consisting of machine profile and contract terms using the GUI and submit it to the Trader Agent, which performs all the trading process and gives a location where the buyer can execute their job. He doesn't need to know anything about the underlying system. But he/she can affect the way the OCEAN Trading system works by modifying the various configurable parameters such as Hop Count, Matching Time etc.

XML Documents Creation

Since it is generally not easy to write XML documents by hand GUIs are created to produce Trade Proposal documents. Classes responsible for creating these documents are named as BuyerDocCreator and SellerDocCreator. Both these classes use JDOM API calls, which builds and manipulates the structure of XML documents by representing an XML document as a node tree. By accessing the tree structure new nodes can be added and existing nodes can be searched and changed. Trade Proposal documents can be created by calling CreateProposal proposal class and by supplying two arguments. The first argument identifies the type (Buyer/Seller) and the second argument specifies the URL. CreateProposal class takes these two arguments and fires a GUI for entering the proposal. The current GUI provides a form for entering the items of a trade proposal

such as CPU Power, High-Speed Primary Memory, Persistent Secondary Memory such as Hard Disk, Operating System, Grid type, Task type, Schedule Information, Price, Payment Terms, Contract Terms such as Compensation for not honoring the lease etc. We plan to include lease information and benchmark information in the future versions. All the data items entered in these forms for various bid/offer items are collected and are used as input for various JDOM API calls to create the document at the specified URL.

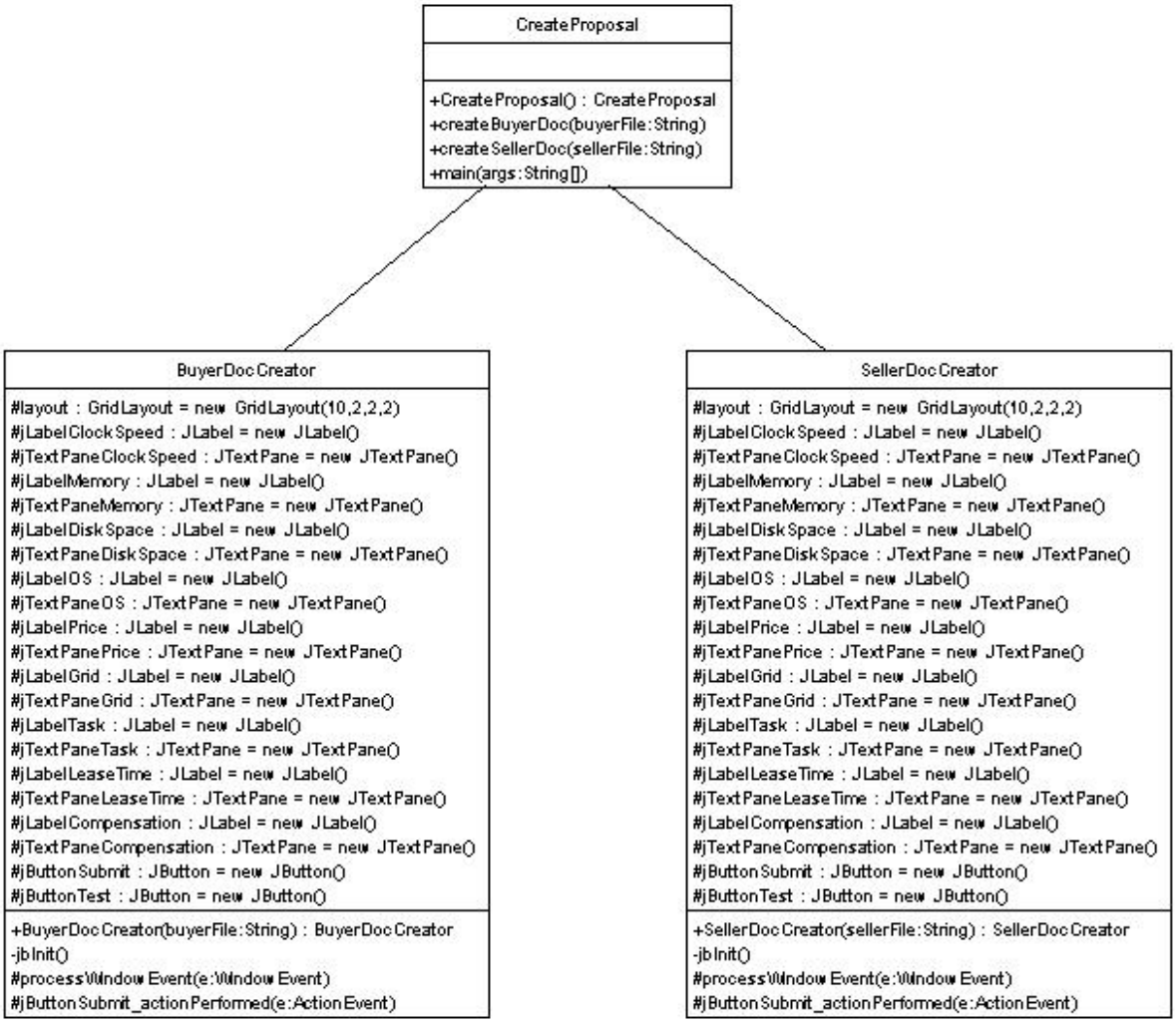

Figure 4-1. Class Diagram of the GUI for creating XML proposals. Create Proposal gives a GUI that is used to create XML documents.

Bid and Offer Object Creation

The classes Bid and Offer represent the Bids generated by buyers and Offers generated by Sellers in OCEAN system. Objects generated from these classes include all the functionality needed to represent Bids and Offers in OCEAN market. Bids and Offers are created from the XML documents by calling some static utility functions in XMLUtility class. Both Bids and Offers extend an abstract class called TradeProposal. The TradeProposal class implements comparable interface so that the Agent can sort Offers later during Negotiation phase.

All the Bids in OCEAN system can be uniquely identified using Unique ID member. Both bids and Offers encapsulate all the information to identify the Buyers required and Sellers actual machine profiles respectively. In addition to this Offers also contain the functionality to determine whether a particular bid is compatible with it or not. These methods hold the following signature:

- **int priceComaptible (Bid):** This method in Offer takes a Bid created from search request and sees whether they are compatible in price terms. A Bid is compatible with Offer only when the price offered by a bid is at least 'DIF'% greater than actual Offer price where DIF is determined by the OCEAN authority. This difference in price is set to compensate the sellers along the path and also to pay for the transaction to be done by the Central Accounting Server.
- **int resourceCompatible (Bid):** This method in Offer takes a Bid and checks whether its machine profile can satisfy the desired profile in Bid that is if the Offer can satisfy all the rigid/minimum requirements stated in Bid.
- **int contractTermsCompatible (Bid):** This method in Offer takes a Bid and checks whether its contract terms agree with its terms.

Of these methods the first two are checked during matching phase and the last one is checked during negotiation/contract-making phase.

Figure 4-2. Class Diagram of the TradeProposal class. It implements comparable interface so that the classes that extend it (Bid and Offer) can be later sorted.

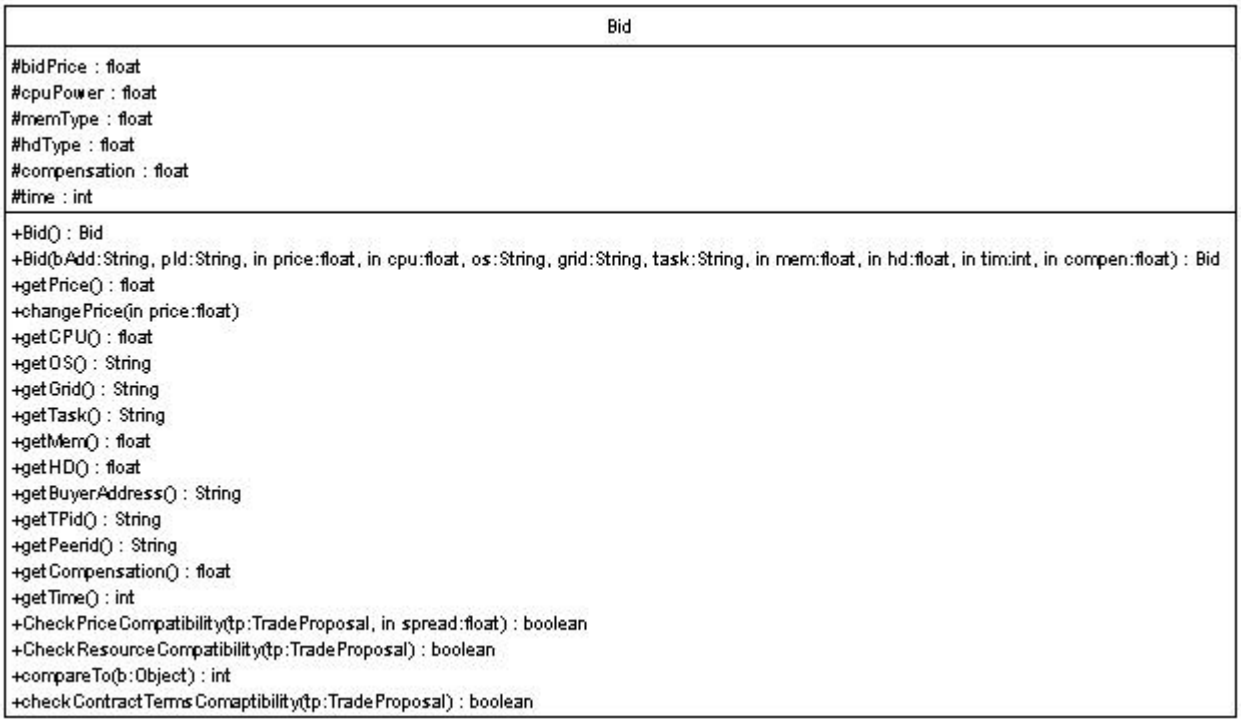

Figure 4-3. The Bid Class that encapsulates all the fields that should be in a search message to find the appropriate OCEAN node that could do its task. It also contains all the functionality so that we can find a particular Offer prepared by Sellers in OCEAN system cam match with it. Matching Systems use this functionality. Offers in OCEAN also look similar to this.

Automated Trader Agent

This is the main interface for OCEAN traders. Buyers submit their Trade Proposal XML Documents to the remote Matching Systems using Trader Agents that control the

whole trading process. Buyers and sellers don't need to understand any details of underlying trading mechanism. Trader Agent at a buyer does the following:

(1) Creates Bids from the XML documents using XMLUtility class.

(2) Register itself with the Matching Communication Component to receive matches by providing the unique id of the proposal.

(3) Created Search messages based on these bids and propagate them in OCEAN network. These search messages are SOAP messages created using the XMLUtility class.

(4) Receive Match Messages from its local Communication Component.

(5) Creates Offers from these SOAP Messages and put them in a sorted list.

(6) Create Contract Messages and perform one-to-one negotiation with the Sellers in this ordered list until a successful contract is made with one seller.

(7) Log the contract to Central Accounting System using Local Accounting Component functionality.

Each Bid in OCEAN network has an associated Trader Agent. This Trader Agent works in close with Matching Communication Component to perform a trade in OCEAN market as explained in the above process.

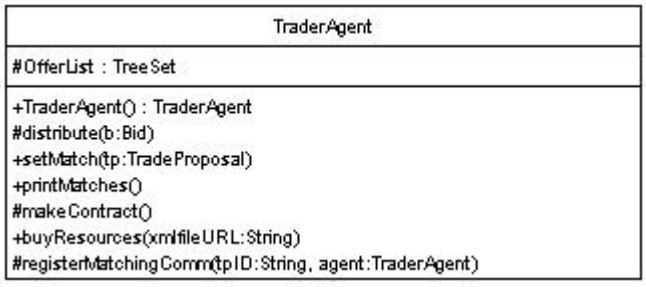

Figure 4-4. Class Diagram of the Trader Agent class that controls all the trading in OCEAN. Trader Agents distribute Search requests using Communication component. They register with communication component and accumulate all the matches that occur at various Seller nodes. Then they sort that seller list and make a contract with the best seller.

Matching System

All Seller nodes have a Matching System where they enter their Offer. The Communication Component submits all the bids it receives from various nodes to this matching system. The Matching system in OCEAN matches these Bids submitted to the system with its Offer. If a Bid is matched with an Offer the matching component sends a match message back to buyer using the Communication component.

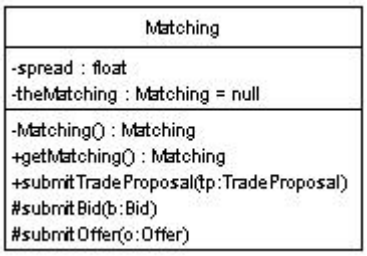

Figure 4-5. Class Diagram of Matching Component at Seller nodes matches the bids and Offers. Communication Components submit the bids to Matching from search requests they receive. The matching component indicates the communication component if it has a match for a particular bid.

Communication System

Matching Communication class is responsible for sending and receiving Search requests and Match messages on behalf of Trader Agents. It uses the PLUM component for trade search message propagation. On the Seller end it receives search messages and enter them into their local matching system. All Trader Agents at a particular OCEAN node register with their communication class to receive matches.

All Trader agents use their own Matching Communication to propagate and receive search requests and match messages by registering with the Trade Proposal Unique ID. This registration is done is done using a Hash Map, which stores the pairs of Trade Proposal objects and their associated Trader Agents. This hash map is later used by the Communication component to submit the matches to correct Trader Agents.

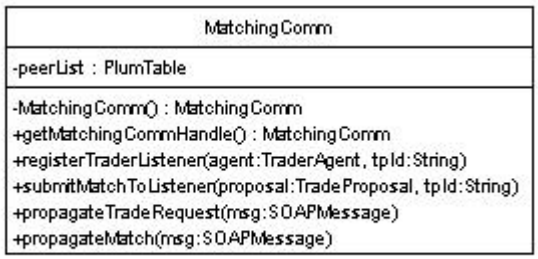

Figure 4-6. Class Diagram of MatchingComm that represents the communication component of the Trading System. All Trader Agents at a particular node register with it with a particular Trade Proposal ID. MatchingComm propagates the search requests in a Peer-To-Peer fashion using the PLUM table. They submit all the matches they receive to Trading Agents listening for them.

PLUM Table and Peer

All the information associated with a peer is encapsulated in Peer Class. Each Peer in OCEAN contains a Unique ID (Got from Registration), Address (DNS address) and Priority (High priority peers are contacted first for matching). Of these fields priority field is constantly changed according to the successful/unsuccessful contracts it made with other systems. All peers implement “comparable interface” so that they can be ordered according to priority in the PLUM Table, which maintains a list of peers. PLUM tables give the peer lists to Trader Agents whenever they need to send search messages to find Sellers in OCEAN network. For full implementation of PLUM component we need an intelligent and adaptive Peer Manager class, which creates, and maintains this peer list.

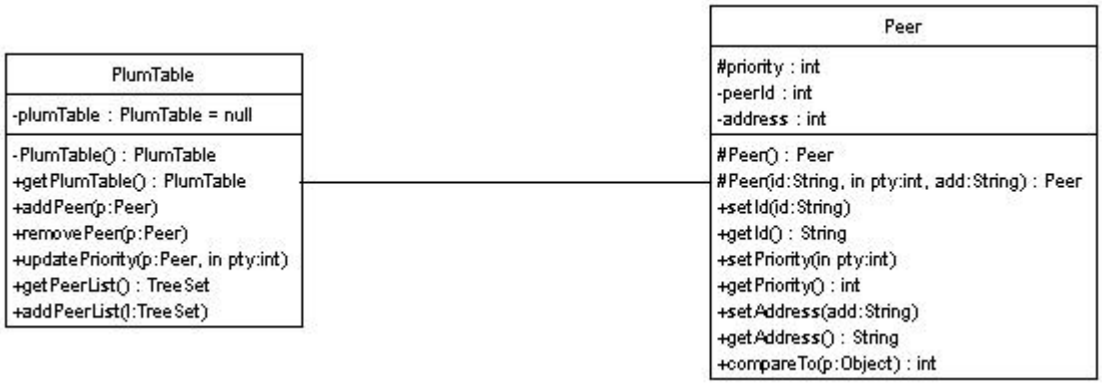

Figure 4-7. A Peer Node in OCEAN is identified by its Identification, IP address, and the priority. PLUM table holds the list of peers it knows about. It provides functionality to add, remove, update peer node information. Communication Component uses this Peer Node List in PLUM table for Search Request distribution.

SOAP Messaging

All messaging in OCEAN nodes (Search, Match, Contract) is done using SOAP. SOAP messages are transferred over http. JAVA SOAP Messaging API is used for sending and receiving SOAP Messages. The two classes used for this purpose are:

(1) **SoapMsgSender:** It takes a SOAP Message and a URL of receiving end and sends the message. This is done using a secured connection socket.

(2) **SoapMsgReciever:** It is a server process (a Java Servlet), which keeps listening on the end machine for SOAP Messages. This server process is started when the Matching System and Matching Communication are started and it runs as long as the OCEAN system is alive. It parses the SOAP Message and submits it to the Matching Communication after identifying the type of message (search request/match/contract request).

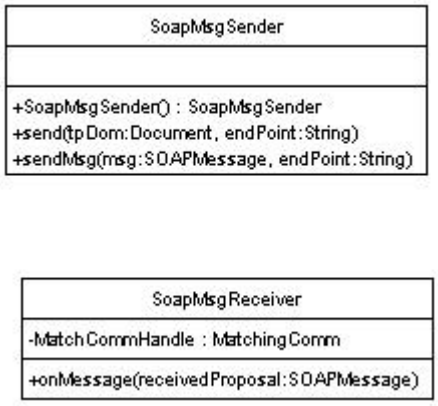

Figure 4-8. SOAP Messages are created and received in OCEAN peer nodes using SoapMsgSender and SoapMsgReceiver classes. SoapMsgSender takes SOAP message and End Point and sends the message. SoapMsgReciever is a java servlet that listens for the SOAP Messages. SoapMsgReceiver submits all the messages it receives to Communication Component for further consideration.

The SOAP server at each peer listens for SOAP messages. These messages can be either search messages or match messages or contract messages. When the message received is search message it creates a Bid from it and enters it into Local Matching system. When the message received is a match message it creates an Offer from it and enters it into the Sorted List of the Trader Agent. When the message received is a contract message it sends accept or reject message by verifying the contract terms. The XML Schemas of SOAP messages are given in appendix.

XML Utility Class

This Class contains all the logic to create and parse XML/SOAP messages. It provides some static methods to:

(1) Create Bid and Offer objects by parsing XML documents.

(2) Create SOAP based Search/Contract/Match messages from Bids and Offers.

(3) Create Bids and Offers from SOAP Messages.

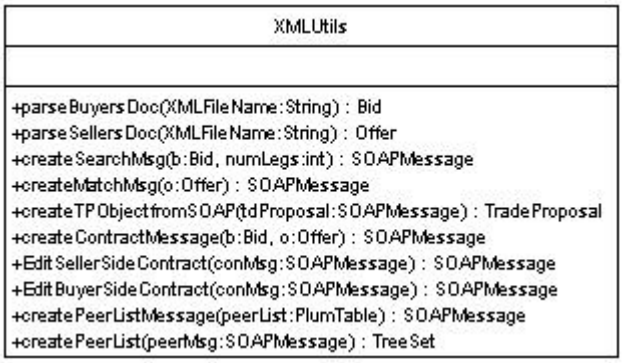

Figure 4-9. XMLUtils is the utility class used to create XML documents and SOAP messages from Files, Bid/Offer Objects respectively. XMLUtils class is used to create Search Messages and Match Messages from Bid and Offer Objects and vice versa.

CHAPTER 5
RUN TIME VIEW OF A SIMPLE SYSTEM

Testing

At the time of this writing some of the major parts of the Trading System such as Peer List Update Manager, Registration System and Security System are still not developed. Also there are no users registered with the OCEAN system. The first major version is expected to be released by December 2002 for people at University of Florida. Because of lack of OCEAN aware applications we just did Unit testing of the major components in the OCEAN Trading System. Some normal Java Classes and some JUnit [27] classes are written to test the system. All these classes have one Static Main method that test the functionality of particular component. Below is a description of various classes that are written and what they do:

TestMatch. This class takes hard coded URLS of some Buyer XML documents and URL of one Seller XML document and creates Bids and Offer respectively from them. It submits the Offer to the Matching System first and then randomly submits all the bids. The Matcher does the matching and prints all the Bids that are matched.

TestOrdering. This class takes a set of Seller XML documents and creates Offers from them. It also gets an instance of the TraderAgent and submits all these Offers to them. It then calls the method to sort the matches and finally prints all the Offers and their price. We can see that Offers are sorted according to the increasing order of price.

TestSOAPMessages. This class tests all the functionality of XMLUtils class. It creates Search Messages from Bids and Match Messages from Offers and vice versa. It

prints all the messages to some files in the implementation directory to check whether the messages are properly created are not. It also tests the creation of PeerList Messages and Contract Messages.

TestSoapPropagation. This class tests the transfer of SOAP messages between the Systems. Various servers in Computer Science Department at UF are used to test this functionality. This test is actually performed by using two classes TestSoap and TestSoapPrint that creates and sends Search Message and that receives and prints the Search Message into a file respectively.

We plan to do more automated testing before the first release of OCEAN software as this kind of Unit Testing is not enough for testing the functionality of the Final System. More Integrated and semi-real testing will be done in feature. In the nest two sections I am going to show a run time view of a simple system implemented using the design shown in previous chapter.

GUI And Trade Proposal

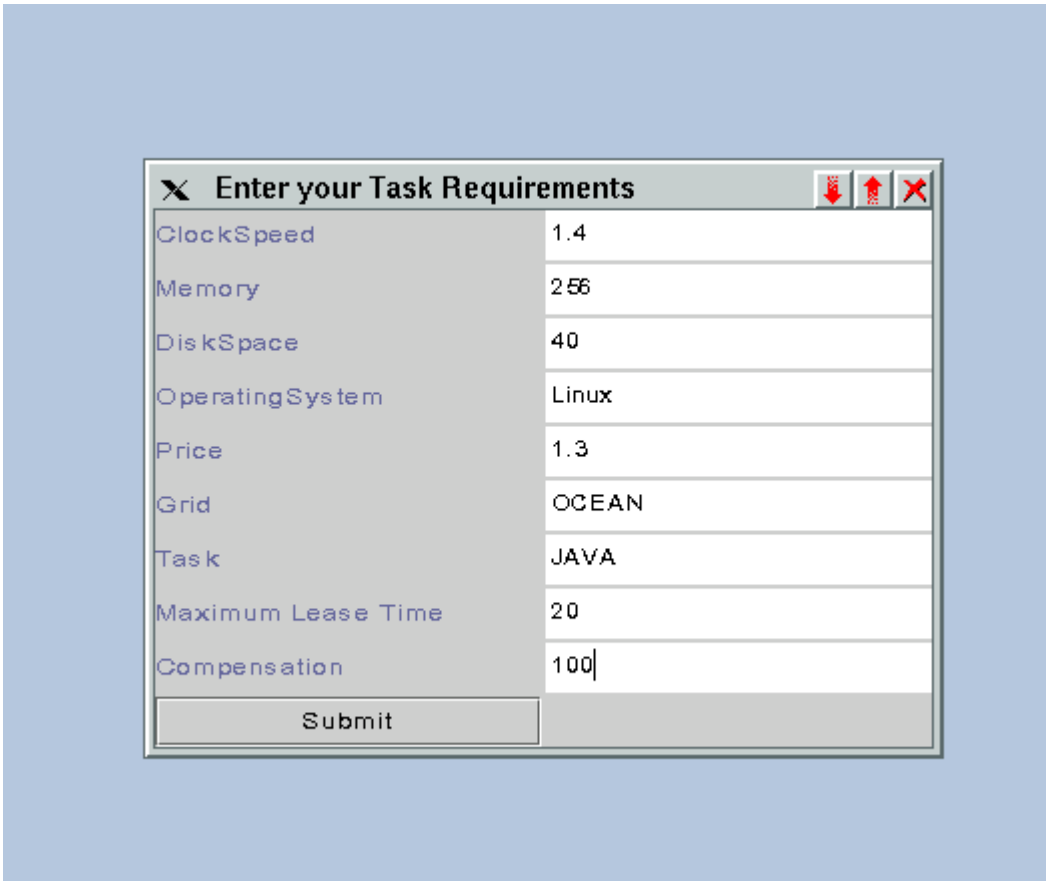

Figure 5-1. A very simple GUI built using Java Swing API. All the fields entered will take their respective default units

Trade Proposal XML document created from the above GUI form looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BuyerResourceDoc>
  <CpuSpeed>1.4</CpuSpeed>
  <Memory>256</Memory>
  <HardDisk>40</HardDisk>
  <OperatingSystem>Linux</OperatingSystem>
  <Price>1.3</Price>
  <Grid>OCEAN</Grid>
  <Task>JAVA</Task>
  <LeaseTime>20</LeaseTime>
  <Compensation>100</Compensation>
</BuyerResourceDoc>
```

Soap Messages

Simple Search, Match, Contract Messages created in the simple system looks as follows:

Search Message

```
<soap-env:Envelope
 xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ocean="http://www.cise.ufl.edu/research/ocean">
<soap-env:Header>
  <ocean:UniqueID>Sriram7b</ocean:UniqueID>
  <ocean:purpose>search</ocean:purpose>
  <ocean:numLegs>3</ocean:numLegs>
</soap-env:Header>
<soap-env:Body>
 <ocean:UniqueID>Sriram7b</ocean:UniqueID>
 <ocean:CpuSpeed>1.4</ocean:CpuSpeed>
 <ocean:Memory>256.0</ocean:Memory>
 <ocean:HardDisk>10.0</ocean:HardDisk>
 <ocean:OperatingSystem>Linux</ocean:OperatingSystem>
 <ocean:Grid>ocean</ocean:Grid>
 <ocean:Task>java</ocean:Task>
 <ocean:Price>2.4</ocean:Price>
</soap-env:Body>
</soap-env:Envelope>
```

Match Message

```
<soap-env:Envelope
 xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ocean="http://www.cise.ufl.edu/research/ocean">
 <soap-env:Header>
  <ocean:purpose>MatchMessage</ocean:purpose>
  <ocean:UniqueID>SriramKumar</ocean:UniqueID>
 </soap-env:Header>
 <soap-env:Body>
  <ocean:CpuSpeed>1.6</ocean:CpuSpeed>
  <ocean:Memory>512.0</ocean:Memory>
  <ocean:HardDisk>40.0</ocean:HardDisk>
  <ocean:OperatingSystem>Linux</ocean:OperatingSystem>
  <ocean:Price>1.2</ocean:Price>
  <ocean:Compensation>100</ocean:Compensation>
  <ocean:LeaseTime>5</ocean:LeaseTime>
 </soap-env:Body>
</soap-env:Envelope>
```

Contract Message

```
<soap-env:Envelope
 xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ocean="http://www.cise.ufl.edu/research/ocean">
 <soap-env:Header>
  <ocean:purpose>MatchMessage</ocean:purpose>
  <ocean:UniqueID>SriramKumar</ocean:UniqueID>
 </soap-env:Header>
 <soap-env:Body>
  <ocean:CpuSpeed>1.6</ocean:CpuSpeed>
  <ocean:Memory>512.0</ocean:Memory>
  <ocean:HardDisk>40.0</ocean:HardDisk>
  <ocean:OperatingSystem>Linux</ocean:OperatingSystem>
  <ocean:Price>1.2</ocean:Price>
  <ocean:Compensation>100</ocean:Compensation>
  <ocean:LeaseTime>5</ocean:LeaseTime>
 </soap-env:Body>
</soap-env:Envelope>
```

The current system can be easily extended to follow more appropriate and well-defined schemas shown in Chapter 7. The schemas are developed using XML SPY software.

CHAPTER 6
CONCLUSION AND FUTURE IMPROVEMENTS

Conclusion

The objective of the thesis is to design and implement a Trading System to trade Computational resources in Distributed Networks like OCEAN. This thesis discusses how to build such as system in a distributed network like OCEAN using a peer-to-peer based system. Peer-to-peer system is employed to increase scalability and to provide robustness against failures. And all the peers are classified as either Buyers or Sellers.

The thesis mainly focuses on design of such a system using two services:

(1) Matching Service to find all potential Sellers that match the Search request made from Buyers resource requirements.

(2) Negotiation Service to make a contract with the best Seller among those potential Sellers. This thesis identifies three ways of finding this best seller nodes which are based on resource description, benchmarking and probe. It then examines how the XML based resource description can be used to find the best seller.

The thesis shows how to build Matching and Negotiation as Web Services using SOAP technology. It discusses a sample design and implementation of a simple Trading System using XML and SOAP technologies in Java that can be easily extended to build a more robust System in the future. In the next section I will discuss all the features that this System lacks or it needs to implement to become a more complete system.

Future Enhancements

The current system is a very simple one that doesn't restrict any structure on the SOAP messages. It doesn't explain some of the security issues that arise when we use

SOAP messages. All these improvements are currently being considered and

implemented for future versions like:

(1) **Offline access.** The current System doesn't support offline access. That is a buyer or seller has to be online to get the matches when he submits a trade proposal, which is not the case in traditional auction systems. Future systems should allow Buyers and Sellers to submit their proposals indicating the time at which they want the matches and go offline. After they come back they should be able to just query the peers to get their matches. This feature might be useful for massive applications to procure the resources in advance.

(2) **Security.** There are so many security issues that are not dealt in this thesis. Security module is currently being designed to provide features such as non-repudiation.

(3) **More generic proposals.** We plan to support more generic proposals using consistent nesting of tags. The Buyers may have some strict requirements for their task such as some software libraries etc, which must be declared in the resource descriptions to find appropriate Sellers.

(4) **More tasks.** The current system only supports Java Tasks. Future systems will be able to support more types like .NET etc.

(5) **Matching system.** Future systems should be able to build matching systems based on benchmarks and probes which are more reliable for more complex jobs.

(6) **Negotiation.** The current system does not support any negotiation. We plan to include negotiation that might be useful for high-valued transactions. Also we think it makes sense to include a negotiation system to renegotiate the resources for lease extension.

APPENDIX
XML AND SOAP MESSAGE SCHEMAS

This chapter gives the XML schemas of the Trade Proposal Documents and SOAP messages used in OCEAN. All Trade Proposal documents that are created when a user submits a form in the GUI should follow the XML schema shown below. A Trade Proposal description contains information about Trader, his/her Resource Description and Price Description, Schedule and Contract Terms Description. Search Messages in OCEAN are SOAP messages and they contain the buyers desired Resource Description along with his identification and the Number of hops information. Match Messages are similar to Search Messages but contain the whole Trade Proposal Description including Resource, Price and Contract Terms Information. Contract Messages that are logged to Central Server are SOAP messages that contain the Buyer and Seller Identification information and their deal information such as the Grid Mobility, Price and Contract terms that both partied agreed upon. The XML schemas given below are self-descriptive.

Trade Proposal Schema

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <!--
 Namespace declarations
  -->
- <xsd:schema
    targetNamespace="http://www.cise.ufl.edu/research/ocean"
    version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.cise.ufl.edu/research/ocean"
    elementFormDefault="qualified">
  - <!--
   Trade Proposal description consists of TraderInfo,
    ResourceInfo, PriceInfo, Contract TermsInfo
    -->
  - <xsd:element name="TradeProposal">
    - <xsd:complexType>
      - <xsd:sequence>
```

```
        <xsd:element name="TraderInfo" type="TraderType"
          />
        <xsd:element name="ResourceInfo"
          type="ResourcesType" />
        <xsd:element name="PriceInfo" type="PriceType"
          />
        <xsd:element name="LeaseInfo" type="LeaseType"
          />
        <xsd:element name="CompensationInfo"
          type="CompensationType" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
- <!--
 Trade Info consists of kind(buyer or seller), UniqueID and URL
  of server
  -->
- <!--
 Trader Inforamtion Defintion
  -->
- <xsd:complexType name="TraderType">
  - <xsd:sequence>
    - <element name="kind">
      - <xsd:simpleType>
        - <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Buyer" />
            <xsd:enumeration value="Seller" />
          </xsd:restriction>
        </xsd:simpleType>
      </element>
      <element name="urlInfo" type="xsd:string" />
      <element name="UniqueID" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
- <!--
 end TraderInformation Defintion
  -->
- <!--
 Resource list to be
  considered(CPU,OS,Memory,NetworkConnection,Grid,Task
  -->
- <!--
 Resource Definitions
  -->
- <xsd:complexType name="ResourcesType">
  - <xsd:sequence>
      <xsd:element name="processor" type="CPUType" />
      <xsd:element name="operating" type="OSType" />
      <xsd:element name="memory" type="MemoryType" />
      <xsd:element name="network"
        type="NetworkConnectionType" />
      <xsd:element name="grid" type="GridType" />
      <xsd:element name="taskDesc" type="TaskType" />
    </xsd:sequence>
  </xsd:complexType>
- <!--
 end resource Defintions
```

```
  -->
- <!--
 CPU definition
  -->
- <xsd:complexType name="CPUType">
  - <xsd:sequence>
      <xsd:element name="Architecture" minOccurs="0"
        maxOccurs="1" type="xsd:string" />
    - <xsd:element name="ClockSpeed" minOccurs="0"
        maxOccurs="1">
      - <xsd:complexType>
          <xsd:attribute name="value"
            type="xsd:decimal" />
          <xsd:attribute name="units"
            type="ClockUnits" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
- <!--
 end CPU definition
  -->
- <!--
 Clockunits Definition
  -->
- <xsd:simpleType name="ClockUnits">
  - <xsd:restriction base="xsd:string">
      <xsd:enumeration value="MHz" />
      <xsd:enumeration value="GHz" />
    </xsd:restriction>
  </xsd:simpleType>
- <!--
 end Clockunits defintion
  -->
- <!--
 Operating System description
  -->
- <xsd:complexType name="OSType">
  - <xsd:attribute name="value" use="required">
    - <xsd:simpleType>
      - <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Windows" />
          <xsd:enumeration value="Linux" />
          <xsd:enumeration value="Solaris" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="version" type="xsd:string"
      use="optional" />
  </xsd:complexType>
- <!--
 end Operating System description
  -->
- <!--
 Memory Description
  -->
- <xsd:complexType name="MemoryType">
```

```
  - <xsd:sequence>
      <xsd:element name="SecondaryMemory" type="MemType"
        />
      <xsd:element name="PrimaryMemory" type="MemType" />
      <xsd:element name="CacheMemory" type="MemyType" />
    </xsd:sequence>
  </xsd:complexType>
- <!--
 MemoryType definition
  -->
- <xsd:complexType name="MemType">
    <xsd:attribute name="value" type="xsd:decimal" />
    <xsd:attribute name="units" type="MemoryUnits" />
  </xsd:complexType>
- <!--
 End MemoryType definition
  -->
- <!--
 Memoryunits definition
  -->
- <xsd:simpleType name="MemoryUnits">
  - <xsd:restriction base="xsd:string">
      <xsd:enumeration value="TB" />
      <xsd:enumeration value="GB" />
      <xsd:enumeration value="MB" />
      <xsd:enumeration value="KB" />
    </xsd:restriction>
  </xsd:simpleType>
- <!--
 end Memoryunits defintion
  -->
- <!--
 Network Connection definition
  -->
- <xsd:complexType name="NetworkConnectionType">
  - <xsd:sequence>
      <xsd:element name="Bandwidth" type="BWType" />
    </xsd:sequence>
  </xsd:complexType>
- <!--
 end Network Connection definition
  -->
- <!--
 Bandwidth measurement definition
  -->
- <xsd:complexType name="BWType">
    <xsd:attribute name="value" type="xsd:decimal" />
  - <xsd:attribute name="units">
    - <xsd:simpleType>
      - <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Mbps" />
          <xsd:enumeration value="Kbps" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
- <!--
```

```
 end Bandwidth measurement defintion
  -->
- <!--
 Grid defintion
  -->
- <xsd:complexType name="GridType">
  - <xsd:simpleType>
    - <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Globus" />
        <xsd:enumeration value="MPI" />
        <xsd:enumeration value="Tryllian" />
        <xsd:enumeration value="OceanGrid" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:complexType>
- <!--
 end Grid defintion
  -->
- <!--
 TaskType Definition
  -->
- <xsd:complexType name="TaskType">
  - <xsd:simpleType>
    - <xsd:restriction base="xsd:string">
        <xsd:enumeration value="JavaByteCodes" />
        <xsd:enumeration value=".Net-CLR" />
        <xsd:enumeration value="ObjectCode" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:complexType>
- <!--
 end TaskType Defintion
  -->
- <!--
 Price should be time dependent price saying N$ per timeUnit or
  like that
  -->
- <!--
 price definitions
  -->
- <xsd:complexType name="PriceType">
  - <xsd:sequence>
      <xsd:element name="CurrencyAmount"
        type="CurrencyAmountType" />
      <xsd:element name="Timespan" type="xsd:decimal" />
    </xsd:sequence>
  </xsd:complexType>
- <!--
 end Time Dependent Price Defintion
  -->
- <!--
CurrencyAmountType Defintion
  -->
- <xsd:complexType name="CurrencyAmountType">
    <xsd:attribute name="value" type="xsd:decimal" />
    <xsd:attribute name="Currency" type="xsd:string" />
  </xsd:complexType>
```

```
- <!--
end CurrencyAmountType Definition
  -->
- <!--
 Lease tells the time for using resources without further
  notification
  -->
- <!--
 lease Defintion
  -->
- <xsd:complexType name="LeaseType">
  - <xsd:choice>
      <xsd:element name="Duration" type="xsd:decimal" />
      <xsd:element name="EndTime" type="xsd:date" />
    </xsd:choice>
  </xsd:complexType>
- <!--
 end Lease Definition
  -->
- <!--
 Compensation is the amount demanded or granted for not
  honoring the lease
  -->
- <!--
 compensation definitions
  -->
- <xsd:complexType name="CompensationType">
  - <xsd:sequence>
      <xsd:element name="AmountGranted"
        type="CurrencyAmountType" />
      <xsd:element name="AmountDemanded"
        type="CurrencyAmountType" />
    </xsd:sequence>
  </xsd:complexType>
- <!--
 end Compensation Defintion
  -->
</xsd:schema>
```

Search Message Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema
    targetNamespace="http:/www.cise.ufl.edu/research/O.C.E.A.N"
    xmlns:O.C.E.A.N="http:/www.cise.ufl.edu/research/O.C.E.A.N"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    elementFormDefault="qualified">
  - <!--
   include the Trade Proposal Schema
    -->
    <xsd:include
      schemaLocation="http://www.cise.ufl.edu/~sknallan/thesis/x
      ml/ProposalSchema.xsd" />
  - <!--
   Search or Match Message in OCEAN
```

```
    -->
  - <xsd:element name="Envelope">
    - <xsd:complexType>
      - <xsd:sequence>
          <xsd:element name="Header" type="HeaderType" />
          <xsd:element name="Body" type="BodyType" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  - <xsd:complexType name="HeaderType">
    - <xsd:sequence>
        <element name="Purpose" type="xsd:string" />
        <element name="UniqueID" type="xsd:string" />
        <element name="NumLegs" type="xsd:decimal"
          minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  - <xsd:complexType name="BodyType">
    - <xsd:sequence>
        <xsd:element name="ResourceInfo"
          type="ResourcesType" />
        <xsd:element name="PriceInfo" type="PriceType" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
```

Contract Message Schema

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema
    targetNamespace="http:/www.cise.ufl.edu/research/O.C.E.A.N"
    xmlns:O.C.E.A.N="http:/www.cise.ufl.edu/research/O.C.E.A.N"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    elementFormDefault="qualified">
  - <!--
    include the Trade Proposal Schema
    -->
    <xsd:include
      schemaLocation="http://www.cise.ufl.edu/~sknallan/thesis/x
      ml/ProposalSchema.xsd" />
  - <!--
    Contract Message in OCEAN
    -->
  - <xsd:element name="Envelope">
    - <xsd:complexType>
      - <xsd:sequence>
          <xsd:element name="Header" type="HeaderType" />
          <xsd:element name="Body" type="BodyType" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  - <xsd:complexType name="HeaderType">
    - <xsd:sequence>
        <element name="Purpose" type="xsd:string" />
        <element name="BuyerID" type="xsd:string" />
```

```
      <element name="SellerID" type="xsd:string" />
      <element name="ContractID" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
- <xsd:complexType name="BodyType">
  - <xsd:sequence>
      <xsd:element name="PriceInfo" type="PriceType" />
      <xsd:element name="LeaseInfo" type="LeaseType" />
      <xsd:element name="CompensationInfo"
        type="CompensationType" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

LIST OF REFERENCES

1. M.S. Miller and K.E. Drexler, "Markets and Computation: Agoric Open Systems," in B.Huberman (ed.) *The Ecology of Computation*, Elsevier Science Publishers, Amsterdam, The Netherlands, 1998, pp.133-176.

2. Y. Amir, B. Awerbuch and R.S. Borgstrom, "A cost-benefit framework for online management of a metacomputing system," *Proc. First International Conference on Information and Computation Economies*, 1998, pp 140-147.

3. M. Parameswaran, A. Susarla and A. Winston, "P2P Networking: An Information-Sharing Alternative," *IEEE Comm.,* vol. 34, no.7, July 2001, pp. 31-38.

4. Roxio, Napster Project, 2002, http://www.napster.com, accessed 7/19/2002.

5. OSMB, LLC, Gnutella Project, 2000, http://www.gnutella.com, accessed 12/7/2002.

6. H.Y. Park, "Peer List Update Manager Implementation in OCEAN (Open Computation Exchange And Arbitration Network)," Master's Thesis, 2002, University of Florida.

7. C. Chokkareddy, "Automated Negotiations in OCEAN (Open Computation Exchange And Arbitration Network)," Master's Thesis, 2002, University of Florida.

8. N. Chawla, "Registration And Authentication Protocol For OCEAN (Open Computation Exchange And Auctioning Network)," Master's Thesis, 2002, University of Florida.

9. SET@home, SETI@home project: The Search for Extraterrestrial Intelligence at Home, 2001, http://setiathome.ssl.berkeley.edu, accessed: 11/7/2002.

10. Distributed.net, Distributed.net project, 2001, http://www.distributed.net, accessed: 10/8/2001.

11. Entropia, Inc., Distributed Computing, 2001, http://www.entropia.com, accessed: 1/14/2002.

12. M.J. Tobias, "Resource And Requirement Schemas Applied To Auctioning In A Computational Market," Master's Thesis, 2001, University of Florida.

13. C.M.Sperberg-McQueen, Henry Thompson, Extensible Markup Language (XML), 2000, http://www.w3.org/XML/Schema, accessed 1/7/2002.

14. Jason Hunter, Brett McLaughlin, JDOM API beta 8, 2002, http://www.jdom.org, accessed 11/7/2002.

15. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, Simple Object Access Protocol (SOAP), 2002, http://www.w3.org/TR/SOAP/, accessed 11/7/2002.

16. Sun Microsystems, Inc., Java API for XML Messaging (JAXM), 2002, http://java.sun.com/xml/jaxm/, accessed 11/7/2002.

17. Object Mentor, JUnit Testing, 2002, http://www.junit.org/index.htm, accessed 11/7/2002.

BIOGRAPHICAL SKETCH

Sriramkumar Nallanchakravarthula was born in Hyderabad, Andhra Pradesh, India, on November 12th, 1978. Sriram attended Chaitanya Bharathi Institute of Technology, which is affiliated to Osmania University and located in Hyderabad, Andhra Pradesh, India, where he received a Bachelor of Science degree in computer science and engineering in 2000. In 2000, Sriram entered the computer and information science and engineering graduate program, in which he worked as a teaching assistant and attended school full time all the while completing the requirements for a Master of Science degree in computer and information science and engineering. His research interests include distributed systems, database implementation, and networking protocols.