

PEER LIST UPDATE MANAGER (PLUM) IMPLEMENTATION
IN OPEN COMPUTING EXCHANGING AND ARBITRATION NETWORK
(OCEAN)

By

HEE YONG PARK

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

Copyright 2002

by

Hee Yong Park

ACKNOWLEDGMENTS

I would like to thank my parents for all of their support and encouragement over the years. My success certainly builds on the foundation that they have provided.

My thanks go to my advisor, Dr. Michael P. Frank, for his help, advice and guidance during my graduate career at the University of Florida. I learned a great deal, and had fun doing it too.

TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGMENTS	iii
LIST OF TABLES.....	vi
LIST OF FIGURES	vii
ABSTRACT	viii
CHAPTER	
1. BACKGROUND.....	1
2. ABOUT OCEAN.....	3
Related Projects and Works.....	4
Architecture of OCEAN	5
Scenario	6
Components of OCEAN.....	8
3. PLUM	13
Typical Implementation Approaches.....	13
PLUM Approach of Implementation.....	15
Philosophies of the PLUM Implementation	15
4. PLUM ARCHITECTURE.....	17
5. PLUM FUNCTIONAL SPECIFICATION	19
Role of Components in OCEAN System Architecture	19
Functional Needs	20
Needs of the Local Auction Service	20
Needs of the Node Operator	20
Needs of PLUMs at Other Nodes	21
Needs of the infrastructure promoter	21
Needs from security and communication layer	22
Functional Requirements	22
By Auction Component	22
For Performance	22
Maintenance goal.....	22
Background tasks.....	23
Ranking System.....	23
Node Operator	24

Persistent Object Storage.....	25
Communication with Other Components	25
Other OCEAN Nodes	25
Auction Node.....	26
Secure Communication.....	26
6. PEER SELECTION ALGORITHM.....	27
7. XML SCHEMA FOR PLUM.....	29
XML Schema for Peer List.....	29
XML Schema for Peer Requests	30
XML Schema for Responses	31
A Scenario for Expanding Peer Information	32
8. EXPERIMENTAL RESULTS	34
Thread Number in Ping Utility.....	34
Maximum Number of Peer Transfer in Expand Query	35
9. PLUM LIFE CYCLE.....	37
Registration Stage.....	37
Maintenance Stage.....	37
Termination Stage.....	37
10. FUTURE WORKS	38
Naming and Directory Service	38
SOAP (Simple Object Access Protocol)	38
Lightweight Persistent Object Storage Service	38
APPENDIX	
XML SCHEMAS FOR PLUM.....	39
XML Schema for Peer List.....	39
XML Schema for Request	39
XML Schema for Response.....	41
LIST OF REFERENCES.....	42
BIOGRAPHICAL SKETCH.....	44

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1 Types of Requests.....	30
2 Types of Responses	31

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1 Architecture of OCEAN System	6
2 Scenario for Trading and Negotiation	7
3 Components of OCEAN	9
4 Three-Tier Architecture of Core PLUM.....	18
5 Architecture of PLUM.....	18
6 Architecture View of PLUM's Role.....	19
7 XML Schema for Peer and Peer List.....	29
8 XML Schema for Request Message Format.....	31
9 XML Schema for Response Message Format	32
10 Scenario for Exchanging Peer Information	33
11 Ping Simulation	34
12 Performance of Ping Test	35
13 Efficiency of Peer Transfer Number	36

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

PEER LIST UPDATE MANAGER (PLUM) IMPLEMENTATION
IN OPEN COMPUTING EXCHANGING AND
ARBITRATION NETWORK (OCEAN)

Hee Yong Park

May 2002

Chairman: Michael P Frank

Major Department: Computer and Information Science and Engineering

Peer-to-Peer technology and distributed computing are often considered the next generation of computing. These technologies use computing resources while they are idle or almost idle. For example these technologies could use a PC as a word processor, which uses very low CPU power. Internet computing, such as Grid computing, is an emerging technology. It is often considered the future of computing. There are still many parallel applications no single computer system is able to solve. The Open Computing Exchange and Auctioning Network (OCEAN) project attempts to solve this problem by providing a new distributed computing framework with low barriers to entry based on a fully automated commodities market for computational resources (such as CPU cycles, memory, disk space, etc.).

The Peer List Update Manager (PLUM) is a component of the OCEAN node. As its name shows, it manages the peer list in order to provide promising peers for the Auction component, so successful trades are increased. This thesis shows how the PLUM component has been implemented, and explains the philosophy behind the component. This thesis will also show how the PLUM component helps the Auction components find more promising peers, and how the PLUM component interacts with

other components, including remote PLUMs, by taking advantage of XML technology and Java.

CHAPTER 1 BACKGROUND

Currently millions of users and organizations use computing resources that are connected via the Internet. Previous studies [1, 2, 3] show that these resources are frequently idle and underutilized. This is true of computer users who own computers 600 Mhz or higher, or those who own computers 100 Mhz or lower. It is true whether these users are word processing, emailing, surfing on the Internet, or listening to music. Users spend an increasing amount of money purchasing computing resources, but these resources are underutilized. An early project like SETI@Home, sponsored by the University of California at Berkeley, shows that all personal computers have the potential to be major computing resources, in addition to their limited role as word processors. After the Berkeley project, similar projects [4, 5, 6] attempted to integrate the underutilized computing resources in order to increase their utilization. These efforts have increased scholastic interest in computer fields, such as Peer-to-Peer and Grid computing.

Computers connected to the Internet are not involved in CPU-intensive tasks. They are only involved in word processing and Internet browsing, which consume very little computing power. The total computing power in an organization may often be underutilized, especially during off-peak business hours. In addition, users operating large parallel applications and services may have difficulty allocating the necessary resources in a cost-effective manner, especially if the level of resource utilization over time is highly variable. The demand for new computing resources and Internet computing technology is the reason why the computational market was created [4].

In order to utilize PDP-1, a computational market by auction was started as early as 1968, at Harvard University. Users bid for time slots, and the highest bidder received the time slot for the machine. Currently, there are several projects such as Distributed.net, Grid, and United Devices which use the auctioning system to buy and sell computing resources. The OCEAN project is one of the most recent projects which use auctioning. The OCEAN project is defined as a functional infrastructure supporting automated, commercial buying and selling of dynamic distributed computing resources over the Internet [5].

Java and eXtensible Markup Language (XML) are the core technologies used to implement the OCEAN project. Java helps to solve the compatibility of all hardware and operating systems. It provides rich built-in Application Programming Interfaces (API), which eliminates unnecessary work. XML is the universal format for structured documents and data on the Web. PLUM uses the XML technology for data communication especially XML schemas for describing the structure and constraining the contents of XML documents.

CHAPTER 2 ABOUT OCEAN

The OCEAN project is a major ongoing project at the University of Florida (UF) Computer and Information Science and Engineering Department. The goal of the project is to develop a functional infrastructure supporting the commercial buying and selling of dynamic distributed computing resources over the Internet.

The OCEAN project was started by a group of MIT students and Stanford alumni in 1997[5]. In 1999, Dr. Michael Frank reorganized the project, and transformed it into a major project at UF.

The goal of the OCEAN project is to build a marketplace in which CPU time, memory, and network bandwidth are tradable commodities. In order to facilitate trading, the major components used are computational resources and underlying market mechanisms.

The OCEAN project has three main goals. First, a user who has, or is able to purchase underutilized or cheap computational resources should be able to sell these resources on the market. Second, a competent developer should have no difficulty writing a distributed application. And finally, a user with a valid credit card, or another means of automatic payment should be able to bid for affordable computational resources.

An OCEAN user does not necessarily need to be present at all times. A user may be a programmed agent acting as a user [6, 7]. Users may be buyers or sellers, or act as both. Buyers need computational resources to solve a computational problem, or to run programs. Sellers want to sell computational resources. When buyer and seller agents have negotiated a contract, the program which needs to be executed will

be transported to the seller's machine(s). It will be executed in the seller's machine, and then the result will be returned to the seller. The computational market's role is to provide a means of payment and an environment in which these interactions may take place even without the user's interaction.

Related Projects and Works

There are many other projects similar to the OCEAN project. These projects attempt to provide infrastructures for distributed computing. Most of these projects are different than the OCEAN project. Unlike other projects, OCEAN provides an automated, open market intended for widespread adoption, with a Peer-to-Peer auctioning mechanism. Similar projects may also be distinguished by their for-profit or not-for-profit status.

Well known projects from not-for-profit organizations are SETI@home [8], Distributed.net [1], and Legion [9]. These projects use only the computing resources donated by users for research purposes. For example, SETI@home is a scientific experiment which uses computers connected to the Internet to facilitate the search for extraterrestrial intelligence. Users donate computing resources to join the project, and download a small program, similar to a screensaver. The program also downloads a portion of radio signal data from space, and attempts to match any pattern which may be a sign of the existence of extraterrestrial intelligence. Because these projects do not have an economic benefit to users, it is difficult to increase the deploying servers, because users may be lost to competitors with market mechanisms.

United Devices [10], Entropia [11], Parabon [12], and Porivo [13] are examples of venture companies which provide a marketplace for buyers and sellers. In this type of system, users buy and sell computing resources. However, there are several problems in an open, computational market. First, closed markets are

completely controlled by these companies, unlike the OCEAN project which is controlled by the users themselves. Secondly, it is difficult for application developers to create applications, because free open standard APIs are often not provided. Thirdly, market systems are also much less flexible than the OCEAN project, because the computational resources in a typical market system should be contracted in advance and cannot be purchased on-demand when computational resources are most needed. Finally, market systems are less scalable than OCEAN, because they have a centralized resource management system, unlike the OCEAN project which has a Peer-to-Peer distributed auction system.

Architecture of OCEAN

The OCEAN system is a distributed computer environment. Its goal is to buy and sell computational resources over the Internet. A node may be placed in the OCEAN system either directly or indirectly through a firewall. The system is either connected to the Internet through a direct IP connectivity to other nodes, or it is behind a firewall in a private IP address space in an intranet. In the latter case, a special communications proxy installed on the firewall provides OCEAN connectivity to and from the intranet, if it is permitted by the site's administrators. An OCEAN node may perform more than one task at a time. For example, a node may participate in the distributed auctioning process while simultaneously performing a computation for one or more multi-threaded client jobs. The auction only requires a small fraction of the computational resources of the OCEAN node. The auction service is free as part of the normal overhead cost of participating in OCEAN. However, in the future, OCEAN upgrades will provide a mechanism that may compensate auction nodes for the trades that they successfully facilitate.

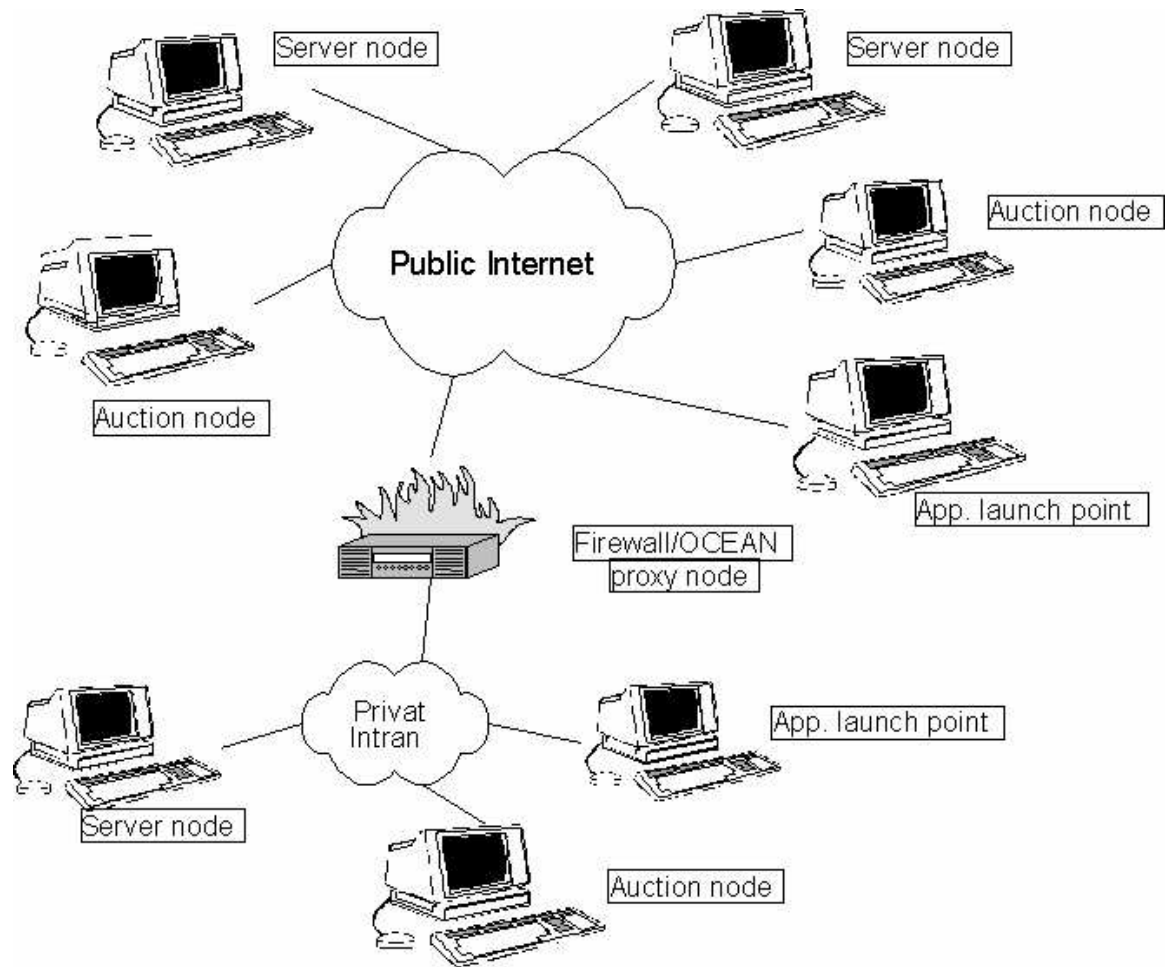


Figure 1 Architecture of OCEAN System

Scenario

This is a scenario which shows how the OCEAN system works for buyers and sellers. A buyer first needs computational resources and discovers that the OCEAN system can provide reliable resources. The buyer must first register with the Central Accounting System (CAS), and then obtain an account for exchange using cyber or real money. The buyer downloads the OCEAN node program, which contains all the components for auctioning and negotiation. The buyer makes a trade proposal, which contains information about what computational resources are needed, and for what length of time. There is also information provided about resource deadlines, and any other pertinent information the buyer thinks the seller should know about the target

program. The auction component of the local OCEAN node also sends copies of the trade proposal to the peers the local node knows.

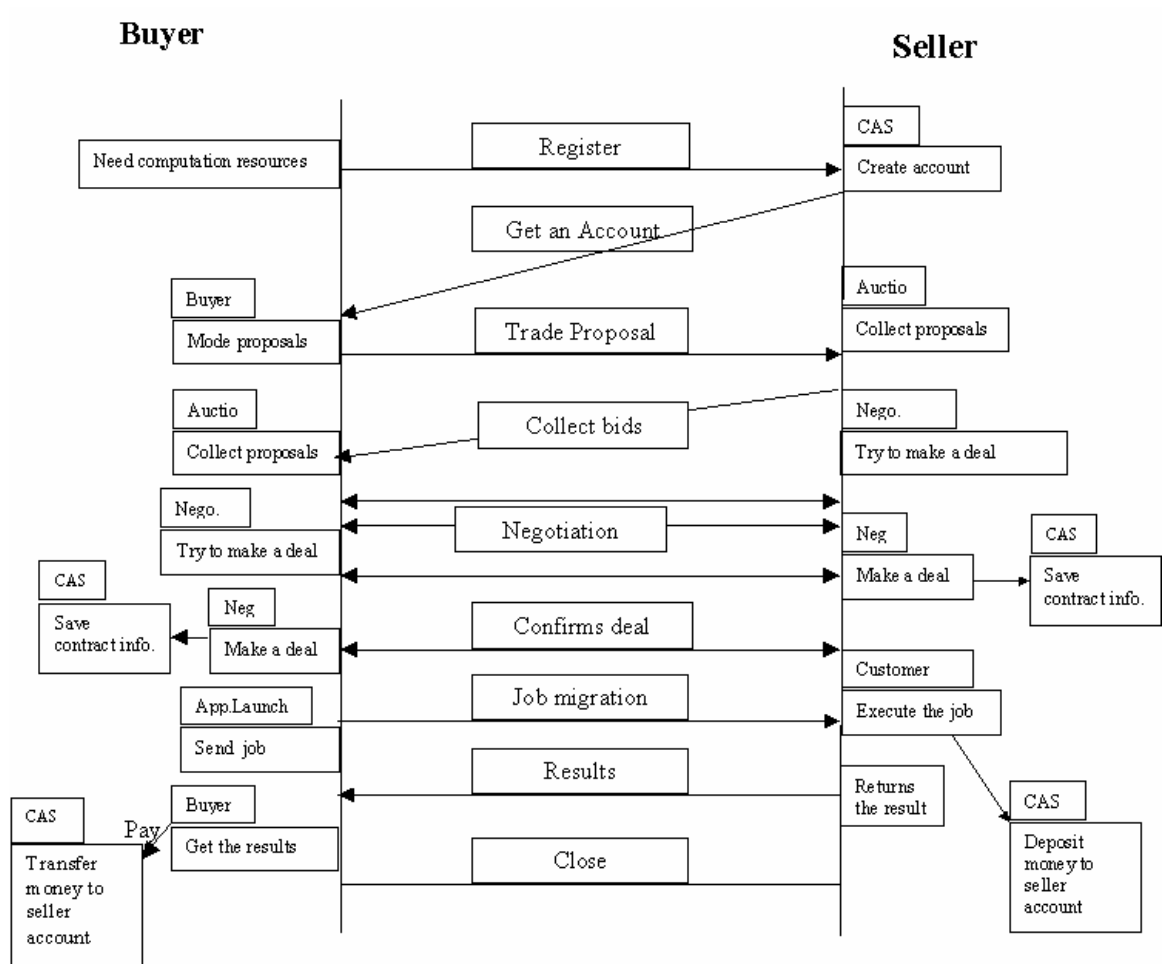


Figure 2 A Scenario for Trading and Negotiation

A seller who wants to sell computational resources collects all the proposals from the other OCEAN nodes. These are the buyers. The seller finds the best deals and starts negotiation. When the buyer and seller agree on the right price, they create a contract and send it to CAS. After CAS approves the contract, the buyer sends the executable program to the seller. The seller executes the program and returns the results of the execution. There is always a possibility that the seller may not finish the program due to program errors or other urgent reasons. If this is the case, the seller

informs the buyer that the seller was not able to finish the job, and that the buyer needs to find another seller. The seller who failed to finish the job is allowed to have his trust points reduced. If the seller successfully finishes the job, CAS will transfer money based on the contract the buyer and the seller entered into, and both the buyer and the seller receive trust points based on the algorithm for trust points.

Components of OCEAN

The high-level system or network organization is illustrated in Figure 3. The components are described as follows.

OCEAN Node

An OCEAN node is an instance of a freely available server software package by an organization that wants to buy or sell computational resources. An OCEAN node normally contains all the core components although some components may be omitted or unused in nodes of specialized types such as gateway nodes and application launch points. Nodes can be deployed all over the Internet and also behind firewalls.

Central Accounting Server (CAS)

The CAS is the centralized account system for OCEAN. Any organization or individual should first register to the Central Accounting Server in order to participate in OCEAN. The CAS provides services to log contract agreements and micropayment transactions for individual trades. It also maintains account balances for all the traders and periodically talks to the real-world financial networks to debit/credit the traders' outside accounts, according to accumulated funds earned and spent.

Firewall/Gateway Node

This node acts a communications proxy for OCEAN nodes living within a private IP space in an organization or a home LAN. It requires communications

forwarding to operate transparently through the firewall. Of course, the administrator of the firewall must agree to run the proxy. The OCEAN Naming system allows even private nodes to be referenced through appropriate pathnames.

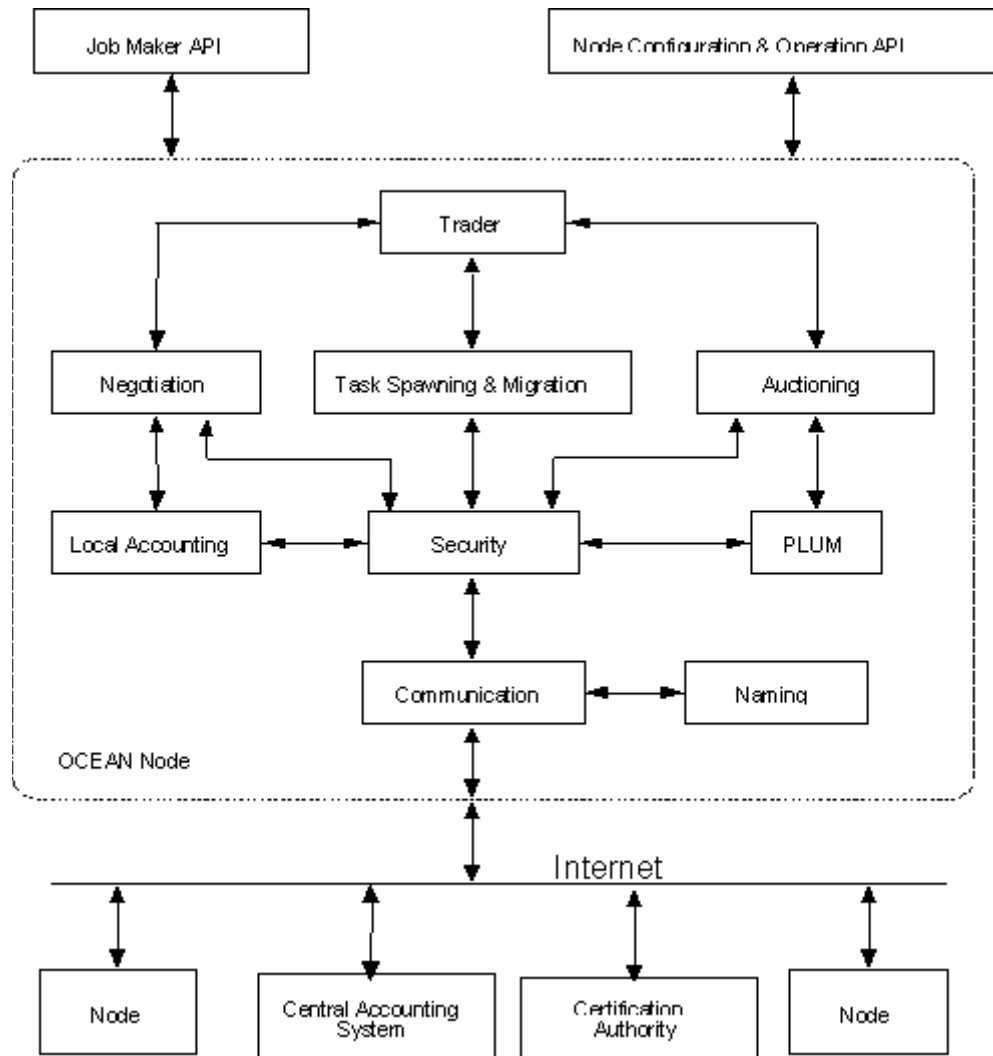


Figure 3 Components of OCEAN

Trader

The Trader component acts as an interface among the human participants and the rest of the node components. Node-operator interaction is accomplished through the external APIs. In effect the Trader subsystem acts on behalf of the customers to

allow them to communicate with the Negotiation, Auctioning, Task Spawning and Migration subsystems.

Auctioning

The Auctioning subsystem provides the marketplace where traders exchange economic benefits with computational resource. The Trader subsystem must provide information to the auctioning subsystem so that trade proposals are entered in auctions. When the auctioning system needs to communicate with the auctioning systems of other nodes, it does so with assistance from the Peer List Update Manager (PLUM) and Security components.

Peer List update Manager (PLUM)

The main purpose of PLUM is to ameliorate the problem of directory lookup operations mentioned by Parameswaran et al. [14]. More specifically, it is impractical for nodes to communicate with one another in a broadcast fashion. Instead they must communicate with selected peers. It is the responsibility of PLUM component to determine provide the promising peer list with which a node will communicate.

Negotiation

The Negotiation component allows traders to automate the process of negotiation. The Auctioning system determines who are potential trading partners. However the Negotiation system provides the means to allow traders to agree on the terms of a contract. This includes resolving conflicts that arise when a trader has many potential trading partners.

Task Spawning and Migration

The Task Spawning and Migration (TSM) component is responsible for the proper dissemination and execution of computing tasks in the network. When a contract for the execution of a computation is successfully negotiated the TSM subsystem will migrate computing tasks to remote servers through the Security and

Communication layers at each node. When the server receives the task(s) to be executed, they must be started and run to completion. Also, that task may spawn new tasks and so on.

Security

The Security component plays an integral part in all communication performed among nodes. When one node communicates with another, the Security system must verify that the communication is legal. Furthermore, the Security component must protect sensitive user data when it gets transferred over the network.

Naming

The Naming component is responsible for name resolution for the OCEAN. The Communication system is the main service target. In particular it is undesirable for components to refer to OCEAN nodes strictly by their IP addresses. IP addresses can be dynamic and obscured by firewalls. In addition it is desirable to identify computers on network and other entities such as computing tasks, data, and resources. The Naming and Communication subsystems provide this functionality to locate any OCEAN entity on the Network.

Communication

The Communication component has the responsibility of handling all low-level communication between nodes. It interacts with the Security subsystem to accomplish communication among components on different nodes. For example when a trade proposal is propagated from one auction to another, the proposal must pass through the Security and Communication layers of each node en route to the receiving auction.

Local Accounting

Each node uses the Local Accounting component to communicate with the centralized accounting system. When contracts are successfully negotiated and

subsequently honored (or not) the Local Accounting subsystem will ensure that the transaction is logged to the Central Accounting system.

CHAPTER 3 PLUM

PLUM maintains a list of the addresses of the other OCEAN nodes, and the status information about these nodes, such as the present and average accessibility, availability, and intercommunication bandwidth and latency. Based on the node administrator's preferences, PLUM periodically updates its peer list. When the customer registers on the OCEAN database system, the database system will give a set of initial peers. It will measure communication latency and bandwidth by sending test messages. In addition, PLUM expands the peer list in several ways. First, it asks the peers in the local peer list to send their peer information. Secondly, the promotion requests by remote nodes. PLUM is able to send the promotion messages to the other nodes in XML document format. Whenever PLUM attempts to add new peer information, it checks the redundancy and trust points which determine the local ranking. If there is no space to add, or it already has the peer in the peer list, then it simply ignores the request. Every several minutes, it will cut out the old peers which have been unreachable or unavailable for several hours. These time limits may be set by the node operator, or may be automatically set to default values.

Typical Implementation Approaches

In order to implement a PLUM-like application to maintain peer information, there are three typical approaches: centralized, distributed, and Peer-to-Peer. In this section, we compare these three typical approaches with the way PLUM is implemented. In the first approach, a centralized server may be used such as MP3.com [15]. A centralized server would store all the information about peers, and

clients would be able to access music files through it. The advantage to this approach is that it is easy to maintain the integrity of the information, because only one server would store all the information and files. If a user wished to upload a new music file, or change the music file information, it would be possible to update it on the centralized server. However, there are some disadvantages to this approach. It would cause a communication bottleneck on the server-side, and there would be high maintenance costs for the main server. There is also a possibility that the server would crash.

The second approach is pure Peer-to-Peer style, such as Gnutella [16]. Since there are no servers, but only peers which have equal functionalities, a peer could act as a client or server, or both. Peers have music files and peer information. Peers directly exchange music files with other peers. There are many advantages to this approach. There are no maintenance costs for servers. In addition, there is easy deployment. However, there are two main disadvantages. First, it is an inefficient communication algorithm, because there is no central directory system. A peer must broadcast a request and wait until the response arrives. Secondly, versions of the peer-to-peer software are difficult to control. The final approach is similar to Napster [17], which has several distributed-file servers. The servers have only enough information to determine the location of the music file. However, there are no actual music files on the server. Peers exchange music files through peers, not servers. The advantages and disadvantages are combined from the first and second approaches. An advantage to this approach is a well distributed communication load. Since servers do not have to transfer comparably large music files, they store relatively little information about peers. The server has complete control of the file exchange, since peers have to travel through the servers first to obtain information about music files.

PLUM Approach of Implementation

The PLUM implementation is based on the pure Peer-to-Peer approach, with CAS and the peer-selection algorithm. CAS plays the role of a trustworthy certificate authority. CAS is entrusted with all the financial responsibilities. This is crucial for the auctioning mechanism. It saves all the contracts the OCEAN customers have entered into, and awards trust points based on how much was paid.

Each peer has a peer list containing peer information. PLUM does not pick up all the peers in the list, like Gnutella. It selects peers by the peer-selection algorithm based on dynamic and static information. The dynamic information may be availability, network bandwidth, and latency. PLUM checks the dynamic information by sending queries, and by pinging to the peers. PLUM obtains the static information by querying the main OCEAN database and CAS about transaction history and trust points. The peer-selection algorithm is explained in detail in Chapter 7.

Philosophies of the PLUM Implementation

PLUM has three main implementation philosophies.

First, PLUM should require minimal user involvement. It should automatically run when the OCEAN node runs, and update the peer list periodically—perhaps every 7 minutes. The node operator may set all the configuration parameters, indicating how often PLUM should update, clean, expand, and promote peers.

Secondly, PLUM should be a reliable peer-list provider. This is crucial for auctioning. If PLUM provides a bad list of peers, the possibility of effectuating a good deal will be very low. To be a reliable peer-list provider, PLUM has several functionalities. First, PLUM has a persistent object storage for saving the peer information permanently because even after rebooting the OCEAN node, PLUM does

not lose any peer information. In addition, PLUM has an effective peer selection algorithm. It monitors the peer status like latency, bandwidth, and existence.

And lastly, PLUM should provide APIs with the ability to communicate with other components, including remote OCEAN nodes. PLUM is implemented in Java and XML. This makes it easy to understand. It also makes it easier to expand the PLUM functionalities in the future.

CHAPTER 4 PLUM ARCHITECTURE

In the architecture of OCEAN, PLUM cooperates with communication, security and auction components. It uses the communications and security components to get a secure socket to exchange messages with its peers with authenticated route. It will be used primarily by the auction component. It provides the auction component with a list of promising peers to exchange bids with. For configuration PLUM should have interfaces for configuration when the node operator needs to customize PLUM. PLUM also interacts with the rating-service component to get the trust points of the peers which shows how much it could trust the peers. The more trust points it has, the more reliable and preferable it is. PLUM may need some sort of lightweight database or persistence manager to maintain its state so that it does not need to re-establish its peer list from scratch every time the node gets rebooted (especially static information such as the transaction history). However in this stage, we just use the local file system to store the peer list and current status information. As you see in Figure 5 below, PLUM architecture has three-tier layers. The first layer is the peer list which contains the peer information and methods to manipulate the peer information. The second layer is the core PLUM class and query manager. Core PLUM class has a collection of methods to handle requests from the other local components and remote PLUM components through the query manager. Originally, the query manager should belong to the communication layer but for purposes of PLUM demonstration we assume that it is in PLUM. The query manager handles all the requests from remote nodes and PLUM.

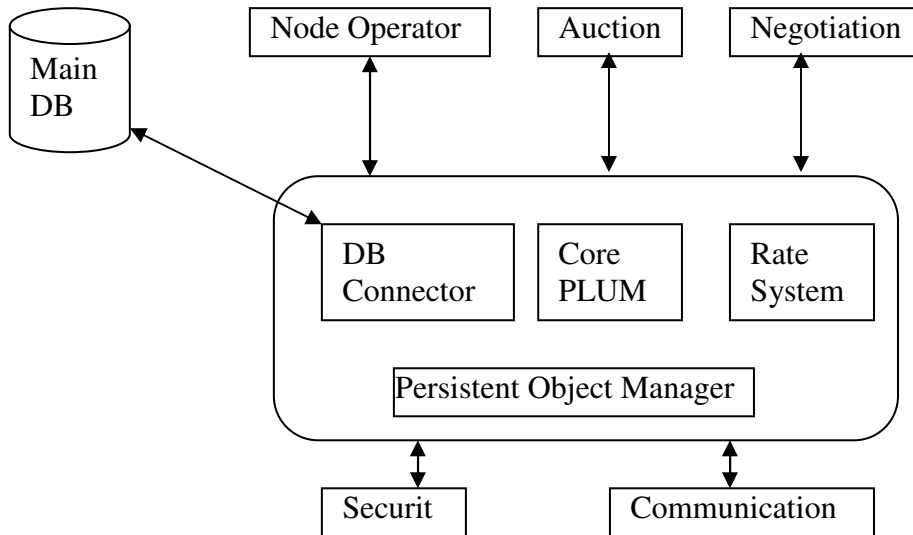


Figure 4 Three-Tier Architecture of Core PLUM

It uses an XML utility from a third party to parse and extract the information needed. The third layer is all the nodes and components outside of PLUM that need to communicate with PLUM.

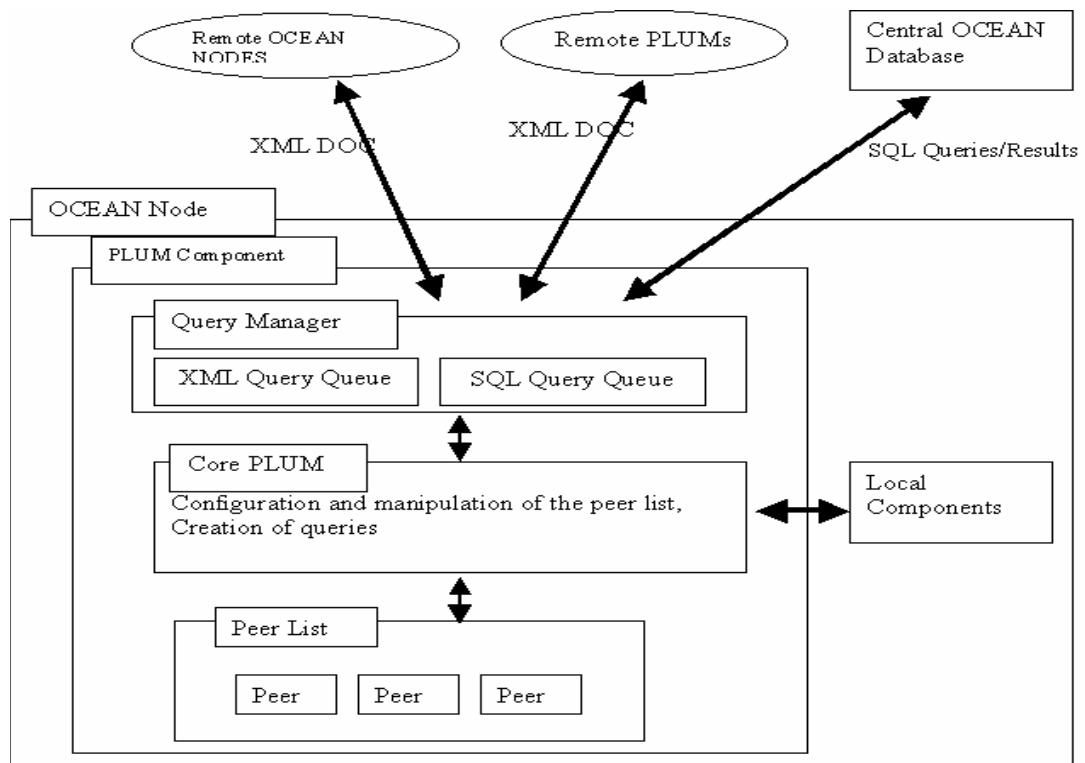


Figure 5 Architecture of PLUM

CHAPTER 5 PLUM FUNCTIONAL SPECIFICATION

Role of Components in OCEAN System Architecture

This section briefly describes the context of the component and how it relates to the system architecture of the OCEAN system as a whole.

PLUM component is present in each OCEAN node. It is used primarily by the Auction component, although it might also receive helpful information from the Negotiation component. Like most node components, it uses the security and communications layer for communications to other nodes on the network as shown Figure 6.

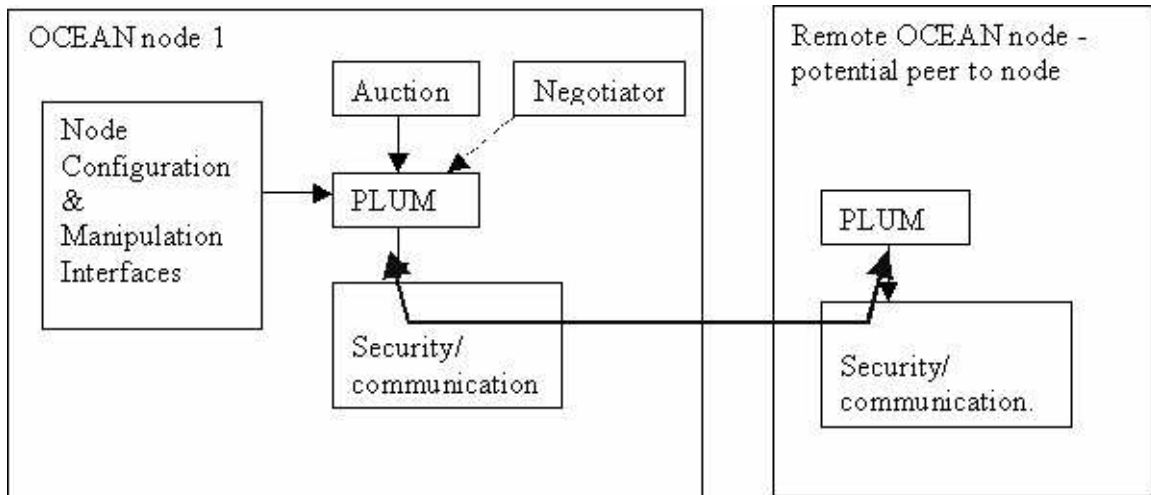


Figure 6 Architecture View of PLUM's Role

Like all node components, PLUM supports configuration and maintenance interface. The OCEAN network communications are primarily with PLUM components of other nodes. PLUM's role, as its name implies, is to manage and

maintain a node's list of peer nodes to support the peer-to-peer vision of the distributed auction system.

Functional Needs

Customers. The main PLUM customer is the auction component at its local node. Indirectly, it also serves the node operator. It also has PLUM components of other nodes as customers - it provides certain services to them, by default free of charge. It also exists to help the infrastructure promoter OCEAN itself to ensure that the infrastructure as a whole grows and flourishes.

Needs of the Local Auction Service

1. PLUM local auction needs to be able to quickly but selectively find other OCEAN nodes to which it can/will forward bids.
2. The auction needs to be able to find as many peers as its needs to satisfy its trade proposal redistribution policies.
3. The communication with the peers needs to have reliability, high bandwidth, and low latency to ensure timely resolution of trade proposals.
4. The peers themselves need to be of high quality. A peer should have a good record of leading to match proposals that easily lead to deal closures with dispute-free payments.
5. The local auction component needs to be able to advertise itself to peers, so that it might receive prospective trade proposals from them.

Needs of the Node Operator

1. The operator needs PLUM not to consume an unreasonably large portion of the local resources controlled by the operator such as CPU, memory, disk, network etc.

2. The operator needs to be able to add specific peers that he or she would like to partner with the peer list.
3. The operator needs to be able to manually influence the ordering of peers, in case he or she wants to promote certain partners.
4. The operator needs to be able to "blacklist" specific nodes from appearing on the peer list at all, if that node is mistrusted or a competitor.
5. PLUM should maintain its knowledge persistently, so that if the node is brought down (e.g., from a power failure or for maintenance), its state will remain when brought back up. In this stage, it uses the local file system to save the peer list.
6. For ease of setup, PLUM should not require the node operator to do anything special for it to start working. It should just work in the out-of-box installation.

Needs of PLUMs at Other Nodes

1. Other PLUMs need to be able to test the round-trip communications characteristics, bandwidth, and latency between them and our local node. They need a responder capable of letting them do this.
2. Other PLUMs need to be able to find out about good peers with which they can try communicating.
3. Other PLUMs need to be able to ask PLUM to help them rate third-party peers.
4. Other PLUMs need to be able to ask us to add them to our own peer list.

Needs of the Infrastructure Promoter

To promote rapid growth and widespread adoption of the peer-to-peer OCEAN infrastructure, PLUM reference implementation should provide its peer-support services to other nodes free of charge, while not consuming too much of the node operator's resources which would discourage adoption.

Needs from Security and Communication Layer

To communicate with other OCEAN nodes, it needs secure sockets from security and communication layer.

Functional Requirements

This section takes the broad needs of the customers and translates them into more detailed and specific requirements on the component.

Below are the requirements of PLUM. After each one we show which need or other requirement that specific requirement addresses.

By Auction Component

1. PLUM should provide its local auction component with an enumeration of peer nodes. (Needs 1a, 1b, and 1c). This enumeration should be ranked in order from most-preferred to least-preferred peers.
2. The peers should be ranked according to preference for trade-proposal redistribution purposes. (Need 1c).
3. It should include any given peer at most once. (Need 1a).
4. It should continue generating as many unique peers as the auction requires, until no more can be found. (Need 1b).
5. Enumeration should be as fast as possible, at least for as many peers as can be locally remembered by PLUM within resource constraints. (Need 1a).

For Performance

PLUM should internally maintain a list of peers for fast retrieval.

(Requirement 1d).

Maintenance goal

PLUM should manage the peer list with least amount of system resources.

1. In the factory settings of the reference implementation, some default peers (public nodes to be operated by the OCEAN consortium) should be included in the peer list on PLUM startup. (Need 2f).
2. The peer list should respect a maximum size with a reasonable default and be configurable through the maintenance interface. (Need 2a).
3. The peer list should maintain uniqueness of peers internally. (Requirement 1b, and 1d).
4. The peer list should maintain a preference order of peers internally. (Requirement 1a, 1c, and 1d).

Background tasks

Because the maintenance should use the least resources possible, these tasks should periodically be performed in background tasks

1. PLUM should continuously work to expand its peer list by querying its existing peers for their peer lists and merging the results into its own peer list. (Requirement 1c, and Need 1a).
2. If the peer list grows too large, the lowest-preference peers should be expunged. (Requirement 2b).

Ranking system

The OCEAN ranking system is important to all the OCEAN components to make safe and reliable trades. PLUM manages the peer ranking system based on the history of transactions in which the peer was involved and the dynamic status information such as availability, network bandwidth, and latency. However in this stage the rate system is emulated. In the future it should be separated from PLUM and implemented with detailed job progress information like the trust model in Sun's JAXT.

1. PLUM improves the accuracy of its rankings by testing communications characteristics such as availability, bandwidth and latency to the peers on its list and factoring the results into its rankings. (Requirement 2d, and Need 2a).
2. PLUM queries peers about their rankings/ratings of other peers and factor the results into the local rankings in the background process. (Requirement 2d, and Need 2a).
3. PLUM should accept information from the auction component regarding which peers it forwarded trade proposals to and which forwards eventually led to a match, trade, and payment. This information should be used to influence the local rankings. (Requirement 1d, and Need 2d).
4. As a distributed system the peer rating system should be stable: The mere exchange of ratings among peers should not by itself drift ever closer to their minimum or maximum values except when actual data on trade success or responsiveness justifies this drift.

Node operator

The node operator should be able to work with PLUM for configuration and set certain characteristics of a peer.

1. The node operator could ask PLUM for a certain peer to be included and never automatically expunged.
2. The peer list should be able to be told by the node operator that a certain peer's rating should be boosted or diminished by a certain delta.
3. The peer list should be able to be told by the node operator that a certain peer should be moved to a "blacklist" of nodes that the given node will never peer with or recommend to other nodes. (Need 2d).

Persistent object storage

PLUM should be capable of saving the current peer list to persistent object storage so that when the OCEAN node reboots, it does not lose all the peer information which could be one of the main assets. In this stage we do not have a database-like persistent object storage but just a local file system. It should be the top priority for the future works.

1. The peer list should be backed up using files, a database, or a persistent object system. (Need 2e).

Communication with Other Components

PLUM makes XML queries for all the requests and responses. Here are the detailed needs to communicate with other OCEAN nodes. Communication should be independent from certain types of software. It should also be simple and easy to understand and implement for the entire level of machines from personal computer to super computer.

Other OCEAN Nodes

The reference PLUM implementation should respond to certain queries from other nodes, free of charge (Need 3 and 4a).

1. Other PLUMs not on our blacklist should be able to stream us test messages of varying lengths, which we just stream back to them. This will allow them to measure communications latency and bandwidth to our node. (Need 3a).
2. The node operator should be able to tune the maximum length of test streams we accept. (Need 2a).
3. Other PLUMs not on our blacklist should be able to ask us for our own internal peer list. (Need 3b).

4. The number of peers we return to nodes who request our peer list should be able to be tuned (to amounts less than the maximum) by the node operator. (Need 2a).
5. Other PLUMs not on our blacklist should be able to ask us for our rating of a specific other peer. (Need 3c).
6. The operator should be able to limit the number of such requests to which we respond. (Need 2a).
7. Other PLUMs not on our blacklist should be able to ask us to add them (or perhaps a whole list of peers) as peers. (Need 3c). By default we accept these requests.

Auction Node

The local auction node should be able to ask us to start or stop advertising our auction node to our peers. (Need 1e).

1. Advertising should be turned on by default when a node starts up. (Need 2f).
2. While advertising, as a background task, we iterate through our local list of peers, and "push" our own node address out to all our peers (requesting them to add us).

Secure Communication

The secure sockets are essential to communicate with other PLUMs or OCEAN nodes.

1. Getting a secure socket from communication layer.

CHAPTER 6 PEER SELECTION ALGORITHM

PLUM must select the reliable peers for the auction service to exchange the bids. It is impractical for nodes to communicate with one another in a broadcast fashion. Instead they must communicate with selected peers. It is the responsibility of the PLUM component to provide the promising peer list with which a node will communicate.

When the auction component asks PLUM to send a peer list, PLUM selects promising peers based on trust points. The more trust points a peer has, the more reliable the peer is. The trust points of a peer is calculated on the dynamic and static information. The dynamic information can be retrieved by the ping utility which includes the peer's availability and latency. PLUM also gets the static information from CAS which includes the history of all the transactions the peer was participated in. The history of transactions is classified by the final status of the transactions such as successful, suspended, and failed. Each peer has the three lists of transactions. A transaction status should be determined on the contract after the transaction has been finished. When the transaction is successfully finished, CAS saves it with the final status of successful. The difference of suspended and failed transactions should be on the contract the seller and buyer agreed on. For example the seller and buyer could agree that if the seller returns a certain intermediate result and could not finish the job, the buyer pay 50% of the total transaction cost. If that happens, CAS saves the transaction as a suspended job with 50% of transaction cost.

Here is the equation for trust points.

$$TrustPoints = \left[\left(\sum_{i=0}^{\#ST} T_Cost(i) \right) * 2 + \sum_{j=0}^{\#SST} T_Cost(j) - \sum_{k=0}^{\#FT} T_Cost(k) \right] * Availability * 1/Latency$$

Eq.1 Trust Points

#ST: Total number of the successful transactions

#SST: Total number of the suspended transactions

#FT: Total number of the failed transactions

T_Cost: Paid cost of the transaction (\$)

i j k : the index of the transaction history list

Availability: The percentage of successful pings

Latency: Time delay for exchanging a data packet (mille second)

First, if the final status of the transaction is successful, the peer gets the trust points by the double of the transaction cost. Second, if the status is suspended on the contract, the peer gets the trust points of the paid transaction cost. Finally, if the status is failed, the peer loses the trust points by the transaction cost. However, even a peer has high trust points, it does not mean that the peer is always available and has a good latency. So that availability and latency should be main factors for selecting reliable peers. If the availability is increased, the trust points should be increased too. However, the latency is increased, then the performance might be dropped by communication delay. So the latency and the trust points should be reversed.

CHAPTER 7 XML SCHEMA FOR PLUM

PLUM communicates with other remote components like PLUM in remote OCEAN nodes, XML technology and Simple Object Access Protocol (SOAP) help to communicate with any hardware and software dependencies. Below are XML schemas to define message formats and the kind of functionalities needed. Appendix has the schema for all the XML documents.

XML Schema for Peer List

This shows how the peer list is organized and what elements are required by peer. The actual XML schema is in the Appendix.

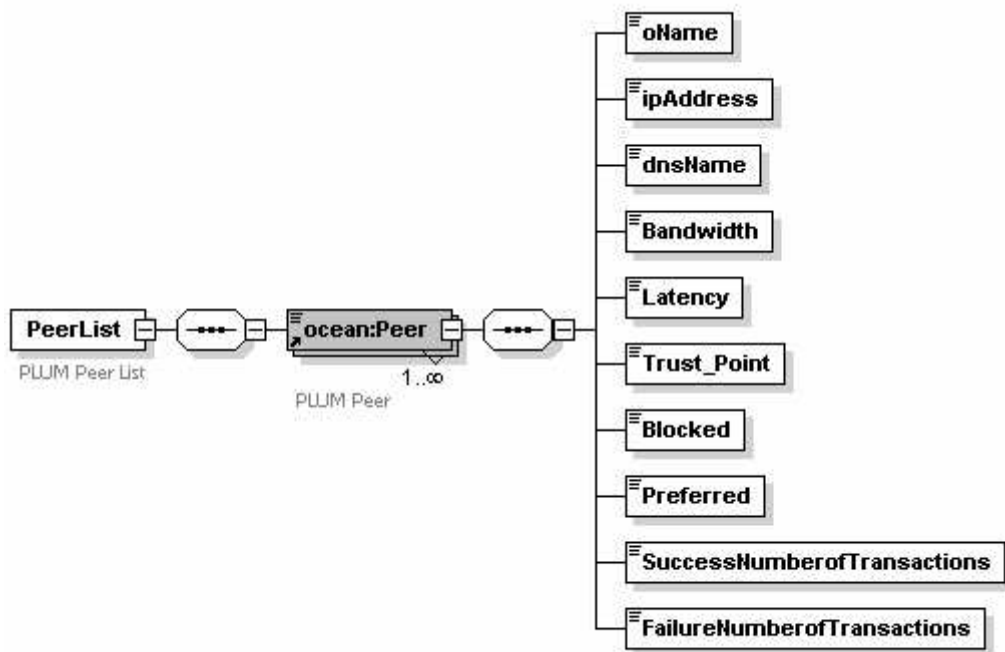


Figure 7. XML Schema for Peer and Peer List

A peer list is consisted of peers and peers contain the necessary information such as ocean name, IP address, DNS name, bandwidth, latency, trust points, flags for blocking, and preferred, the transaction information.

XML Schema for Peer Requests

PLUM has four types of requests. The table below shows a brief explanation for each request.

Table 1. Types of requests

Type	Definition
Expand	PLUM continuously works to expand its peer list by querying its existing peers for their peer lists and merging the results into its own peer list. (Requirement 1c, and Need 1a).
Trust_Point	PLUM queries peers about their trust points of other peers and factor the results into the local rankings in the background process. (Requirement 2d, and Need 2a).
AddtoPL	The local auction component needs to be able to advertise itself to peers so that it might receive prospective trade proposals from them.
Test	To test the network bandwidth, PLUM sends a large byte stream like 1.0mega-byte string stream and calculates how long it takes to send and receive 1.0 mega-byte message.

Below is the figure for XML schema of PLUM requests. The actual XML schema is in the Appendix. As the table 1 shows above, the XML schema for request messages has four different types of contents such as expand, trust points, adding to the local peer list, and testing for network bandwidth.

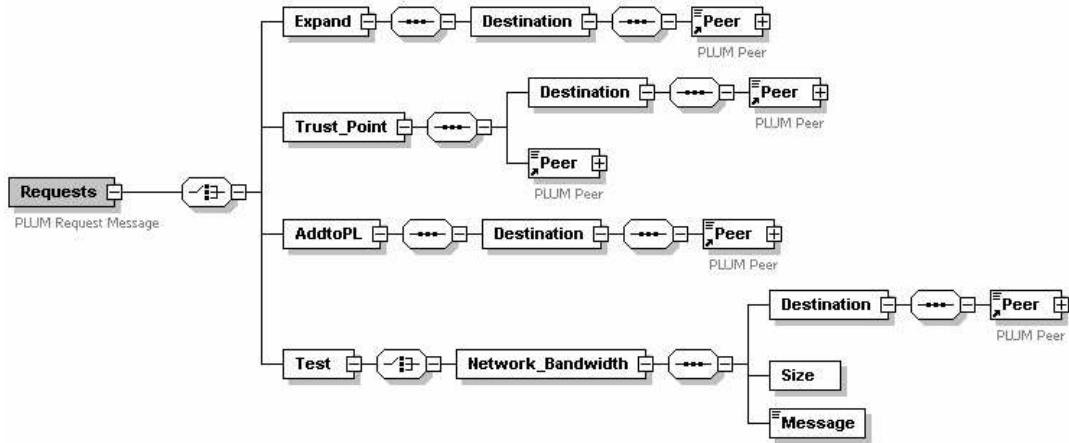


Figure 8. XML Schema for Request Message Format

XML Schema for Responses

PLUM has four different types of responses. Table 2 shows a brief explanation for each request.

Table 2. Types of Responses

Type	Definition
TransferPeerList	Response for the expand request. PLUM returns a list of the local peer list.
Trust_Point	The response for the trust point request. PLUM returns a list of peers with trust points
Test	It is the response for the test request. PLUM returns the packet the originator sent.
AddtoPL	It is the status telling whether or not the request has been completed.

The actual XML schema is in the Appendix. The actual XML schema is in the Appendix. As the table 2 shows above, the XML schema for responding messages has four different types of contents such as expand, trust points, adding to the local peer list, and testing for network bandwidth.

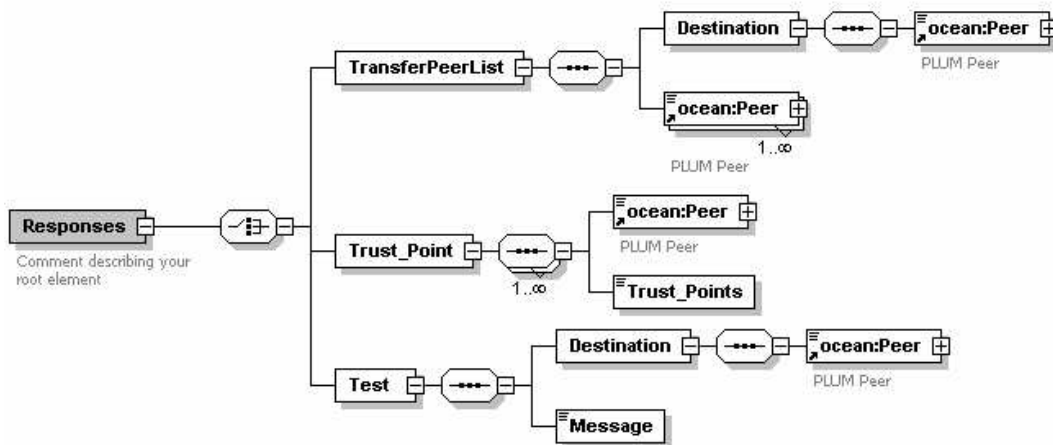


Figure 9. XML Schema for Response Message Format

A Scenario for Expanding Peer Information

Peer nodes need to expand their peer lists to obtain more promising peers. This is in order to make successful transactions. This section describes a scenario in which peers interact with other peers in order to exchange peer information. This scenario assumes that there are several remote OCEAN nodes necessary to expand their peer lists. The diagram below shows how to exchange the peer information between peer A and B.

<Request>

1. PLUM A checks the existence of the local peers by Ping utility.
2. PLUM A wakes up the XML-request thread.
3. PLUM A creates an XML document for the peer-expansion request.
4. PLUM A sends the XML document to all the peers in the local peer list.

<Request Handling>5

1. About every hour, the XML-response thread checks whether there is a request on the waiting queue.
2. If there is a request, the XML parser analyzes it.
3. PLUM B creates the XML document for the request.
4. PLUM B sends the response document to the original requester.

<Request Handling>

1. About every hour, the XML-response thread checks whether there is a request on the waiting queue.
2. If there is an XML document, the XML parser extracts the information from the document.
3. PLUM A adds the peer information to the local peer list.

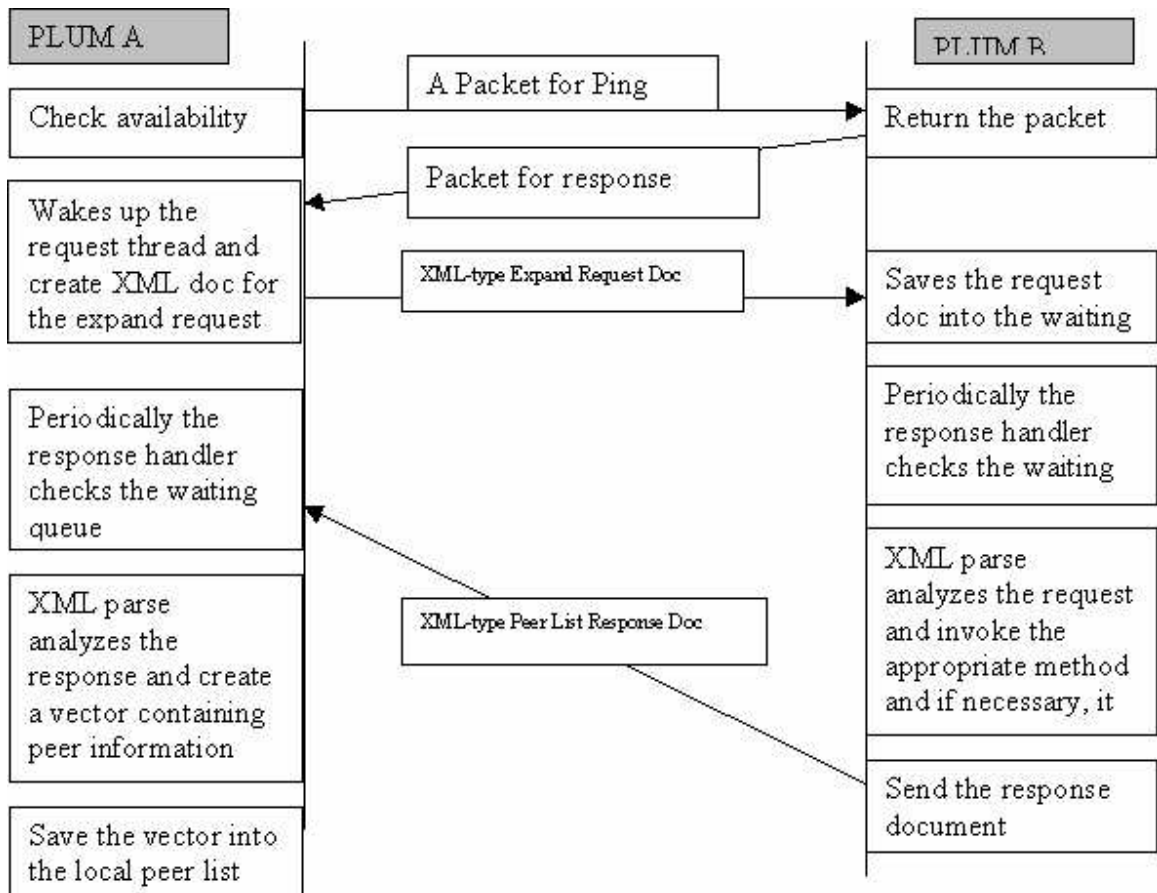


Figure 10. Scenario for Exchanging Peer Information

CHAPTER 8 EXPERIMENTAL RESULTS

Thread Number in Ping Utility

The Ping utility checks the existence of peers. The simulation below shows how the number of threads for the ping tests effects the general performance of the ping utility. It is not efficient to have only one thread sending the same packet to all the peers in the local peer list. This ping test could measure the optimal number of threads for the ping, and for other utilities, such as promotion query threads

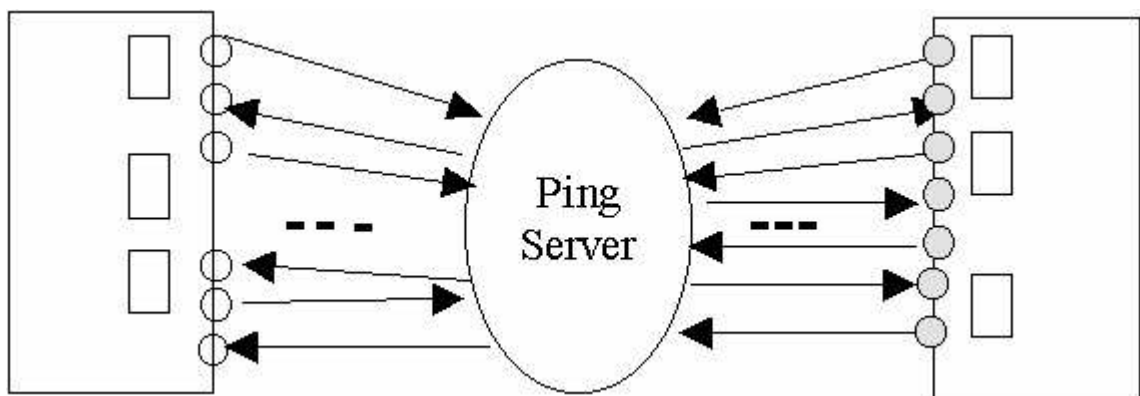


Figure 11. Ping Simulation

The ping test is simulated as Figure 11 shows. Two OCEAN nodes and the ping server are created. A square in Figure 11 represents a thread for sending a packet. The PLUM node generates a certain number of threads, and sends a ping query to the ping server for the simulations. The total execution time is calculated as the average of the total execution time of five simulations. Figure 12 contains the statistical result. It shows how the number of threads effects the total execution time. In this case, the

optimal point is 32, which is the best performance in this simulation. The appearance of more than 32 threads may make the total execution time slower due to thread synchronization.

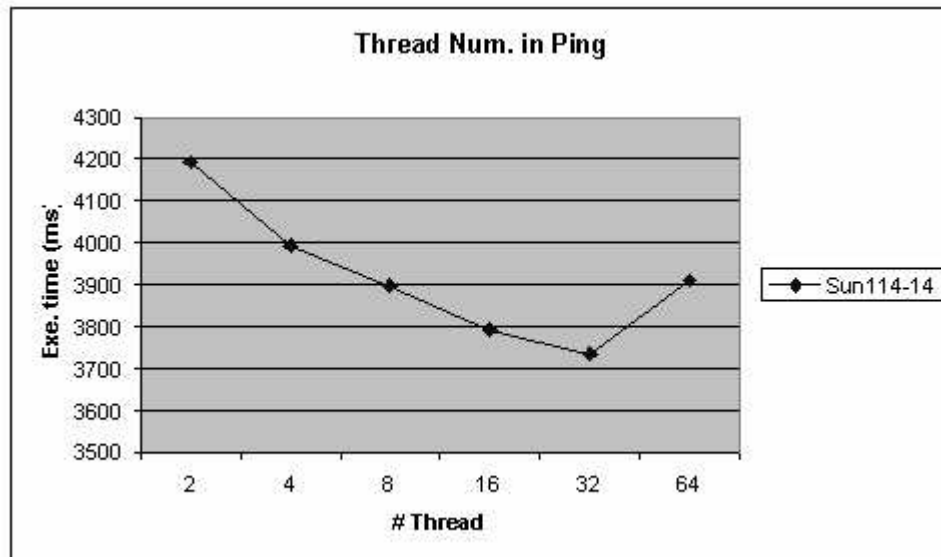


Figure 12. Performance of Ping Test

Maximum Number of Peer Transfer in Expand Query

This simulation shows how fast the peer list could be expanded by limiting the transference of peer information. The simulation is tested using three OCEAN nodes. The nodes contain 50 peers, and the transference of peer information is limited. A OCEAN node is able to send only the maximum peer information. Each node sends an XML-type *expand* request to the other two nodes. When a node receives a request, it sends the maximum peer information to the requester. The original requester eliminates the duplications, and adds new peer information to the local peer list. Figure 13 is the result. It shows how long it takes for it to reach to a certain number of peers. When the maximum number of transferring peers increases, the total execution time decreases exponentially. We can assume that the maximum number of peers

transferring information is one of most important factors in determining the expansion of the peer list.

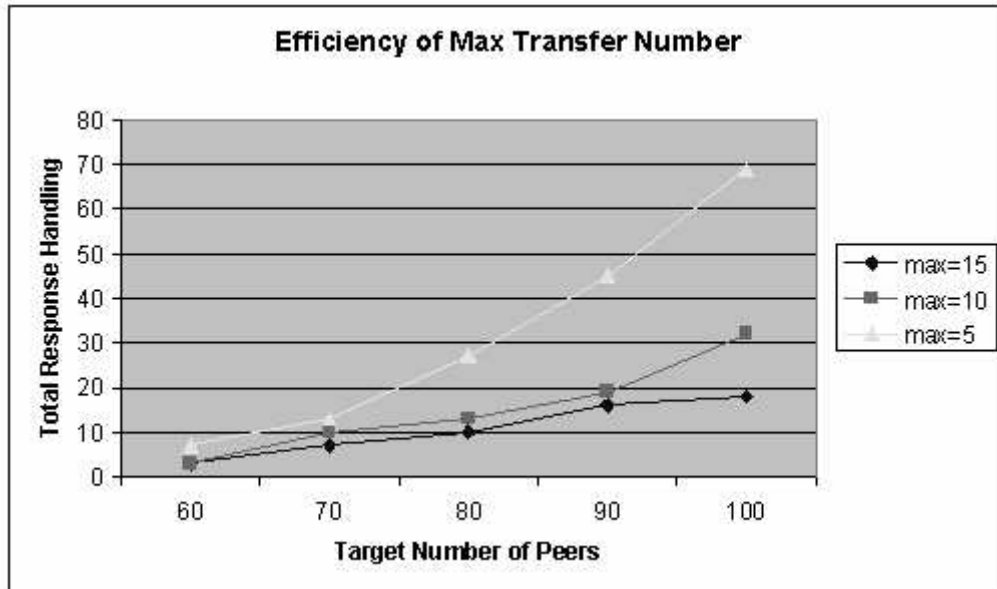


Figure 13. Efficiency of Peer Transfer Number

CHAPTER 9 PLUM LIFE CYCLE

This is a scenario showing how PLUM gets initialized, maintains, and terminates the peer lists. There are three stages registration, maintenance, and termination stages.

Registration Stage

This is the stage when a customer registers for the OCEAN services . The customer gives the information about the personal information, payment options, and local device information. After the customer registers, the customer downloads all the components needed for OCEAN services. PLUM has the ability to securely manage and update the information about the peer list. Once PLUM gets installed, it automatically connects to the main OCEAN database server and gets several peers' information depending on locality or customer's preference.

Maintenance Stage

Whenever a customer opens the OCEAN client or server, it automatically or manually runs Ping service to update the peers' information. The Ping service sends packets to the peers on the peer list and tries to get back all the packages. It analysis the current status of the peers and updates the related information

Termination Stage

When the customer does not want the OCEAN service any more, then PLUM safely deletes all the information of the peers.

CHAPTER 10 FUTURE WORKS

Naming and Directory Service

To support Java and other platforms such as Grid computing, Legion, and legacy systems, PLUM should be able to discover computing resources. Currently the OCEAN system supports only Java platform but it is planned to provide APIs or ways to communicate with other platforms.

SOAP (Simple Object Access Protocol)

SOAP is a lightweight protocol to exchange information in decentralized and distributed environments. To provide secure communication with XML documents it is more secure for the OCEAN system to support SOAP over secure sockets.

Lightweight Persistent Object Storage Service

The PLUM component currently uses the local file systems to save peer information for urgent situations such as the system crashing or rebooting. However it might be more secure if PLUM uses a database-like lightweight storage for saving peer information.

APPENDIX XML SCHEMAS FOR PLUM

XML Schema for Peer List

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by heeyong (University of Florida) -->
<!-- W3C Schema generated by XML Spy v4.0.1 U (http://www.xmlspy.com)-->
<xs:schema targetNamespace="http://www.cise.ufl.edu/research/ocean"
xmlns:ocean="http://www.cise.ufl.edu/research/ocean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="PeerList">
    <xs:annotation>
      <xs:documentation>PLUM Peer List</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ocean:Peer" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Peer">
    <xs:annotation>
      <xs:documentation>PLUM Peer</xs:documentation>
    </xs:annotation>
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="oName" type="xs:string"/>
        <xs:element name="ipAddress" type="xs:string"/>
        <xs:element name="dnsName" type="xs:string"/>
        <xs:element name="Bandwidth" type="xs:float"/>
        <xs:element name="Latency" type="xs:float"/>
        <xs:element name="Trust_Point" type="xs:float"/>
        <xs:element name="Blocked" type="xs:boolean"/>
        <xs:element name="Preferred" type="xs:boolean"/>
        <xs:element name="SuccessNumberofTransactions" type="xs:unsignedInt"/>
        <xs:element name="FailureNumberofTransactions" type="xs:unsignedInt"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema for Request

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by heeyong (University of Florida) -->
<xs:schema targetNamespace="http://www.cise.ufl.edu/research/ocean"
xmlns="http://www.cise.ufl.edu/research/ocean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation="C:\WINNT\Profiles\heeyong\Personal\ocean\PLUM\PeerList.xsd"/>
  <xs:element name="Requests">
    <xs:annotation>
      <xs:documentation>PLUM Request Message</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:element name="Expand">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Destination">
```

```

        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Peer"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Trust_Point">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Destination">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Peer"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element ref="Peer"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="AddtoPL">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Destination">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Peer"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Test">
    <xs:complexType>
        <xs:choice>
            <xs:element name="Network_Bandwidth">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Destination">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="Peer"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="Size"/>
                        <xs:element name="Message">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:length value="1000000"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

```

XML Schema for Response

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by heeyong (University of Florida) -->
<xs:schema targetNamespace="http://www.cise.ufl.edu/research/ocean"
xmlns:ocean="http://www.cise.ufl.edu/research/ocean" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xml:ocean="http://www.cise.ufl.edu/research/ocean">
  <xs:include schemaLocation="C:\WINNT\Profiles\heeyong\Personal\ocean\PLUM\PeerList.xsd"/>
  <xs:element name="Responses">
    <xs:annotation>
      <xs:documentation>Comment describing your root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:element name="TransferPeerList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Destination">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="ocean:Peer"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element ref="ocean:Peer" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Trust_Point">
          <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
              <xs:element ref="ocean:Peer"/>
              <xs:element name="Trust_Points" type="xs:unsignedInt"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

LIST OF REFERENCES

- [1] The Distributed.net homepage, *Distributed Project*, 2001, Distributed.net, accessed 4/25/2002.
- [2] Brian Hayes, "Collective Wisdom", *American Scientist*, vol. 86, no. 2, March-April 1998, pp 118-122.
- [3] OCEAN homepage, *OCEAN Project*. 2001-2002, <http://www.cise.ufl.edu/research/ocean> , 04/25/2002.
- [4] Mark Tobias, "Resource And Requirement Schemas Applied To Auctioning In A Computational Market," Masters Thesis, University of Florida, December 2001.
- [5] Michael Frank, "OCEAN, A Highly Liquid Market for Distributed Computing," Technical Article, University of Florida, Jun 21, 2001.
- [6] Ori Regev and Noam Nisan, "The POPCORN Market—An Online Market For Computational Resources," ICE 1998, Proceedings Of The First International Conference On Information And Computation Economies, October 1998 pp 25-28.
- [7] Eric Drexler and Mark S. Miller, "Incentive Engineering: For Computational Resource Management," *The Ecology of Computation*, Bernardo Huberman (ed.) Elsevier Science Publishers, North-Holland, Amsterdam, 1988.
- [8] SETI Homepage, *SETI@home Project*, 1996-2002, <http://setiathome.ssl.berkeley.edu/>, accessed 4/25/2002.
- [9] The LEGION homepage, *LEGION Project*, 1993-2002, <http://www.cs.virginia.edu/~legion>, 4/25/2002.
- [10] United Devices homepage, *Grid Computing*, <http://www.uniteddevices.com>, accessed 4/25/2002.
- [11] Entropia Inc., *Entropia Project*, 2001, <http://www.entropia.com>, accessed 4/25/2002.
- [12] Parabon Computation, Inc. *Internet Computing is Computing Outside the Box*, 1999 – 2002, <http://www.parabon.com>, accessed 4/25/2002.
- [13] Porivo homepage, *Porivo Project*, 1999–2001, <http://www.porivo.com>, accessed 11/04/2001.

- [14] Manoj Parameswaran, Anjana Susarla and Andrew Winston, "P2P Networking: An Information-Sharing Alternative," *IEEE Computer* vol. 34 no. 7, July 2001, pp. 31-38
- [15] MP3.com homepage, *MP3 Project*, 1998–2002, <http://www.mp3.com>, accessed 4/25/2002.
- [16] Gnutella homepage, *Gnutella Project*, 2000, <http://www.gnutella.com>, accessed 4/25/2002.
- [17] Napster homepage, *Napster Project*, 1999-2002, <http://www.napster.com>, accessed 4/25/2002.

BIOGRAPHICAL SKETCH

HeeYong Park was born in ChinJu, South Korea in 1972. He majored in computer science for undergraduate study 1997 and is expected to obtain a master's degree in computer science in 2002.