AUTOMATED NEGOTIATIONS IN O.C.E.A.N

By

CHAITANYA CHOKKAREDDY

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

CHAITANYA CHOKKAREDDY
To Mom, Dad and Satish K. Chittamuru
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 Jih-Kwon Peir, for their meticulous review of this thesis.

I would also like to express my gratitude to all members, past and present, of the O.C.E.A.N 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 my friend, Y. Sai Krishna, for lending me his computer to work on my thesis, and my girlfriend, Y. Nagasupraja, whose unending support and encouragement were a constant source of inspiration throughout this experience.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGMENTS</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
</tbody>
</table>

## CHAPERS

### 1 INTRODUCTION

1.1 Motivation                                                                 | 1  |
1.2 Approach                                                                    | 2  |
1.3 Organization                                                                | 4  |

### 2 O.C.E.A.N ARCHITECTURE

2.1 An Overview                                                                 | 5  |
2.2 O.C.E.A.N API

  2.2.1 Trader API                                                              | 7  |
  2.2.2 Application Programmer Interface                                        | 8  |
  2.2.3 Node Configuration and Operation API                                    | 8  |
2.3 O.C.E.A.N Node Architecture

  2.3.1 Trader                                                                   | 8  |
  2.3.2 Auction                                                                  | 8  |
  2.3.3 Peer List Update Manager Component (PLUM)                               | 9  |
  2.3.4 Task Spawning and Migration Component                                   | 9  |
  2.3.5 Negotiation Component                                                    | 10 |
  2.3.6 Security Component                                                       | 10 |
  2.3.7 Communication Component                                                  | 10 |
  2.3.8 Naming Component                                                         | 12 |
  2.3.9 Local Accounting                                                         | 12 |
  2.3.10 Centralized Accounting Server (CAS)                                    | 13 |
2.4 O.C.E.A.N Grid – Neutral Architecture                                       | 13 |
2.5 Current Status                                                              | 14 |
  2.5.1 First Iteration                                                         | 15 |
  2.5.2 Second Iteration                                                        | 15 |
  2.5.3 Third Iteration                                                          | 15 |
3 CONTRACT SCHEMAS ..............................................................................................................16

3.1 Contracts In e-Business ........................................................................................................16
3.2 XML And SOAP For Contracts ..........................................................................................17
3.3 Contract Schema ................................................................................................................20
   3.3.1 Essential Elements of a Contract: ...........................................................................20
   3.3.2 Contract Schema ......................................................................................................23

4 NEGOTIATION IN O.C.E.A.N ............................................................................................30

4.1 Negotiation States ................................................................................................................30
4.2 YesNo Protocol And Non-SOAP Based Negotiation In O.C.E.A.N ...............................35
   4.2.1 “YesNo” Protocol ....................................................................................................36
   4.2.2 Non-SOAP Based Negotiation Component ............................................................37

5 SOAP BASED NEGOTIATION COMPONENT IN O.C.E.A.N ...........................................41

5.1 “Static Bargain” Protocol ..................................................................................................43
5.2 Rules In Negotiation .........................................................................................................45
5.3 SOAP Based Negotiation Component .............................................................................46
   5.3.1 Seller .........................................................................................................................47
   5.3.2 Buyer .........................................................................................................................47

6 NEGOTIATION API AND EXAMPLE IMPLEMENTATION .................................................49

6.1 Information Gathering .......................................................................................................49
6.2 Buyer Implementation .......................................................................................................55
   6.2.1 Contract Building And Negotiation ........................................................................55
   6.2.2 Contract Sending .....................................................................................................57
6.3 Seller Implementation ........................................................................................................57

7 CONCLUSIONS AND FUTURE WORK ..............................................................................60

7.1 Conclusions .......................................................................................................................60
7.2 Future Work .......................................................................................................................61

APPENDIX

SCHEMAS ..............................................................................................................................63

   A1 Contract Schema ...........................................................................................................63
   A2 Rule Schema ................................................................................................................64

LIST OF REFERENCES ..........................................................................................................66

BIOGRAPHICAL SKETCH .......................................................................................................69
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig 2.1: The O.C.E.A.N System</td>
<td>6</td>
</tr>
<tr>
<td>Fig 2.2: O.C.E.A.N Node Architecture</td>
<td>11</td>
</tr>
<tr>
<td>Fig 2.3: Naming System</td>
<td>12</td>
</tr>
<tr>
<td>Fig 2.4: O.C.E.A.N Grid-neutral architecture</td>
<td>14</td>
</tr>
<tr>
<td>Fig 3.1: XML Schema legend diagram</td>
<td>23</td>
</tr>
<tr>
<td>Fig 3.2: Envelope element structure</td>
<td>23</td>
</tr>
<tr>
<td>Fig 3.3: Header element structure</td>
<td>24</td>
</tr>
<tr>
<td>Fig 3.4 Body element structure</td>
<td>26</td>
</tr>
<tr>
<td>Fig 3.5: NegotiationItems element structure</td>
<td>27</td>
</tr>
<tr>
<td>Fig 3.6: PaymentInformation element structure</td>
<td>28</td>
</tr>
<tr>
<td>Fig 4.1: Negotiation Activity state diagram</td>
<td>32</td>
</tr>
<tr>
<td>Fig 4.2: Buyer State Diagram of YesNo protocol</td>
<td>38</td>
</tr>
<tr>
<td>Fig 4.3: Seller State Diagram of YesNo protocol</td>
<td>38</td>
</tr>
<tr>
<td>Fig 4.4: Initial Configuration Screen</td>
<td>39</td>
</tr>
<tr>
<td>Fig 5.1: Buyer State Diagram-StaticBargain</td>
<td>44</td>
</tr>
<tr>
<td>Fig 5.2: Seller State Diagram-StaticBargain</td>
<td>45</td>
</tr>
<tr>
<td>Fig 6.1: Initial Negotiation Configuration screen</td>
<td>50</td>
</tr>
<tr>
<td>Fig 6.2: NegotiationFrame Class Diagram</td>
<td>51</td>
</tr>
<tr>
<td>Fig 6.3: StaticBargain Configuration screen 1</td>
<td>52</td>
</tr>
<tr>
<td>Fig 6.4: Negotiation Configuration screen 2</td>
<td>53</td>
</tr>
</tbody>
</table>
Fig 6.5: Negotiation Configuration screen 3 .................................................................53
Fig 6.6: Negotiation Configuration screen 4 ..............................................................54
Fig 6.7: RuleBuilder class diagram ........................................................................54
Fig 6.8: Contract class diagram ................................................................................55
Fig 6.9: ContractMessenger class diagram ..............................................................57
Fig 6.10: Buyer Class Structure ...............................................................................58
Fig 6.11: SellerContractUtils class diagram ............................................................59
The automation of negotiations is a real challenge in the world of e-business. The problem stems from the fact that negotiations generally involve personalities and egos, which are very difficult to simulate in an artificial environment. Although there is a large body of literature on the topic of automated negotiations, most of the work concentrates on the method of negotiation and there is no standard specification for defining contracts, which are an integral part of any trade.

This thesis focuses on developing standards to aid the process of automated negotiations. We formally define a language to specify contracts using which contracts can be defined for any trade. We also define two protocols for negotiation, the "Yes No" protocol and the "Static Bargain" protocol, which make use of the contracts generated according to the specified contract language. The SOAP messaging framework is used to communicate messages between the trading partners.
Integrating this negotiation framework in any larger system in general and O.C.E.A.N in particular is explored. A complete description of the negotiation component in O.C.E.A.N is given and the implementation details are explained.
CHAPTER 1
INTRODUCTION

The automation of negotiations is a very difficult goal to achieve. There are many obstacles in the way of complete automation of negotiations, the most important being the personalities of human beings. People are different; the value of objects is different for different people. So, people will negotiate in different ways. In fact, depending on the circumstances, the same man will negotiate for the same object in different ways. In the real world people use all kinds of techniques like coaxing, pleading, deception and sometimes-even use threats during negotiations to achieve the required results.

In order for computers to negotiate on behalf of the users we might have to incorporate personalities into computers and though a lot of research is being done in the area of Artificial Intelligence we still have a lot of way to go to achieve acceptable results.

1.1 Motivation

The automation of negotiations is not a new concept. Maes et al. [1] of MIT media lab put negotiation at the center of the consumer buying behavior (CBB) model for e-commerce. According to Morgan Stanley Dean Witter [2], the online purchases will grow to $1.4 trillion by 2002. Given the large amount of online business going on, automating negotiations will make it easier for consumers as well as suppliers and in general help e-business.

Though a lot of research has been done on automated negotiations most of it deals with the actual process of negotiation, that is, the strategy used for negotiations, and not
much work has been done on developing standards for automated negotiations. For example Rosenschein et al. [3] gives some very good examples of negotiation strategies.

The one important thing that computers lack is instinct. Computers cannot make decisions instinctively. They need a set standard of rules to follow in all cases and in cases where there are no rules the computer program just crashes. So, it is our contention that in order to automate negotiations, the first thing that needs to be done is standardize a basic set of rules and documents which any trading party participating in a negotiation on the Internet can understand.

Teaching computers to negotiate like humans is a very difficult task. We might employ machine-learning strategies or use other artificial intelligence techniques to teach a computer. But in the end the computer would eventually lose out to a human competitor, except maybe in the case for speed of negotiation is critical. The knowledge base of the humans is just too great and the negotiating skill of humans has been honed through generations.

With these observations, this thesis tries to provide a radical new approach to the problem of automating negotiations. Instead of trying to teach computers to negotiate we provide humans with tools to negotiate using computers, but in an automated way.

1.2 Approach

FIPA [4] has standardized some rules for conducting automated negotiations. But the problem with FIPA is that it has a steep learning curve and people are not interested in learning new technologies for each new job. They would much rather use an existing technology, which is easy to learn. This is where XML [5] comes in. XML is a markup language, which is revolutionizing the way business is being done on the Internet. In this thesis we will try to establish FIPA like standards for automated negotiations using XML.
documents. People can dynamically setup rules for negotiations depending on their circumstances. This somewhat captures the personalities of people during negotiations as different people set up rules differently.

Contracts are an integral part of any trade. But there are no standards to write contracts for e-businesses. This thesis promotes a specification for writing contracts for e-businesses. The contracts will be written in XML. The SOAP [6] messaging framework is used to communicate the contracts between the trading partners.

The O.C.E.A.N (Open Computation Exchange and Auctioning Network) system provides a test bed to verify the validity of the proposed standards. O.C.E.A.N is a distributed application, which buys and sells computing resources on behalf of the users. One of the major components of the O.C.E.A.N system is the negotiation component, which provides automated negotiation services for the users. An example implementation of the negotiation component for O.C.E.A.N using the proposed standards and specifications is provided in this thesis.

Su et al. [7] provides a negotiation server, which can be used for the negotiation component of O.C.E.A.N. But Su et al. [7] was designed to be used by large corporations with multiple simultaneous negotiations whereas O.C.E.A.N is targeted to be used by large corporations as well as single users. As such, it was thought that people would find it difficult to learn a large system like Su et al. [7] and so the decision was made to come up with a new negotiation component, which is much easier to understand, and implement and which at the same time is extensible. But, this in no way means that we cannot use Su et al. [7] as the negotiation component. We can easily substitute the negotiation component provided by O.C.E.A.N with Su et al. [7] if the need asks for it.
1.3 Organization

The thesis is organized as follows:

Chapter 2 provides an overview of the O.C.E.A.N architecture and explains the position of the negotiation component in the overall O.C.E.A.N architecture.

Chapter 3 gives the contract schema specification. Essential elements in a contract are explored and the use of XML for contract writing and SOAP for contract communication is discussed.

Chapter 4 explains how the process of non-SOAP based negotiation is implemented in the O.C.E.A.N system. The “Yes No” protocol for automated negotiations is explained.

Chapter 5 explains the process of SOAP based negotiation in O.C.E.A.N. “Static Bargain” protocol for automated negotiations is given. The implementation of a rule based negotiation system using JESS in O.C.E.A.N is explored.

Chapter 6 provides the API details for the negotiation component in O.C.E.A.N and shows an example implementation.
CHAPTER 2
O.C.E.A.N ARCHITECTURE

2.1 An Overview

O.C.E.A.N stands for Open Computation Exchange and Arbitration (or Auctioning) Network. The purpose of the O.C.E.A.N is to provide a worldwide automated commodities market for the on-demand purchase and remote usage of computing resources (such as CPU cycles, memory, disk space, network bandwidth, access to specialized remote resources, etc.) by distributed applications and mobile agents.

The main goal of O.C.E.A.N is to provide users with a way to buy and sell computing resources seamlessly over the Internet in the easiest way possible with as little human interaction as possible. It also aims at providing application programmers with APIs to develop O.C.E.A.N applications easily.

The two main issues addressed by O.C.E.A.N are scalability and portability. Scalability is achieved by implementing O.C.E.A.N as a distributed, peer-to-peer based architecture. At present, portability is achieved by using O.C.E.A.N on the Java programming language. In the present implementation O.C.E.A.N is packaged as a single unit, and can be installed on any computer with a Java Virtual Machine (JVM), thereby causing the machine to become a node in the Network. Any node can perform more than one task at a time, e.g. hold an auction, execute a buyer's computation, serve as an application development environment, or as an application launch pad.

In the present version of O.C.E.A.N buyers are active and sellers are passive in the sense that buyers initiate the deal and sellers will be waiting for incoming offers. So, in
O.C.E.A.N if a user wants to deploy a job that requires extra resources, a trade proposal of the user requirements is formed and submitted to an auction. The auctioning system may contact other auctioning servers to find a best possible match of the required resources. Once a match is found then the negotiation will start between the seller and the buyer of the resources. If an agreement between the two parties is reached during negotiation, both the parties sign the contract and the code is migrated to be executed at the seller’s machine. The seller runs the code and sends back the results of the computation to the buyer. If the parties are not able to reach an agreement during negotiation, then we move onto another seller and the process is continued. For the first version of O.C.E.A.N parallel negotiations will not be supported. The support of parallel simultaneous negotiations is planned for future releases.

Fig 2.1: The O.C.E.A.N System
The high level interactions among nodes in the O.C.E.A.N system are depicted in Figure 2.1. As can be seen from the figure, a node can be connected to the O.C.E.A.N system either directly through the Internet or indirectly through a node that operates as a gateway to other nodes in an intranet behind a firewall.

A node can play different roles depending on the situation. For example, a node that is acting as an auction server at some point in time can act as a server performing a computation at another time, or both simultaneously. In the present implementation the only exception to the completely distributed nature of the O.C.E.A.N system is the presence of the centralized authentication server, which also keeps account information of all clients in a network. But in the future implementations this bottleneck will also be removed by employing several centralized authentication servers, one for each region. The users register with the centralized authentication server of the corresponding region and the centralized authentication servers of all regions must in turn be registered with a master centralized authentication server. This hierarchy can be extended to many levels depending on the number of users.

2.2 O.C.E.A.N API

Like any other major software system, O.C.E.A.N also provides a basic set of APIs using which developers can extend the system. The O.C.E.A.N API basically comprises of three external APIs: the Trade Proposal API, the API for the programmer and the Node Configuration and Operation API. Each API serves a specific purpose as explained below.

2.2.1 Trader API

The main purpose of this API is to allow the user to specify the trade proposals according to the Resource Specification Language, specify the Contracts for negotiations,
submit bids in an auction and do any other things necessary to carry out trading of computing resources in O.C.E.A.N.

2.2.2 Application Programmer Interface

The API for the programmer abstracts the O.C.E.A.N system away from the programmer. It allows the programmer to develop distributed applications for O.C.E.A.N without worrying about the internal architecture of O.C.E.A.N.

2.2.3 Node Configuration and Operation API

This API allows a node operator to set up his parameters and change them whenever necessary. Some of the parameters that a user sets up when he logs onto the system are resource costs, security parameters, trading mechanism, negotiation protocol etc.

2.3 O.C.E.A.N Node Architecture

O.C.E.A.N system has the following main components:

2.3.1 Trader

The Trader component is the interface between the user and the O.C.E.A.N system. The Trader component manages the Negotiation, Auctioning and basically oversees the trading of resources.

2.3.2 Auction

The auction component [8] in O.C.E.A.N acts as a matching service, which matches traders with the best possible trading partners. Any node in the O.C.E.A.N system can hold auctions. Buyers and Sellers who wish to buy or sell resources enter auctions. The matching algorithm of the Auction component tries to match compatible buyers and sellers. This means that for the Buyers, the Auction component tries to find sellers who are providing the resources that closely match the buyer’s requirements at the best prices and for the Sellers it tries to find Buyers who are asking for resources within the Sellers’
limits at the best prices. The trade proposals in a computational market can be very complex with strict requirements regarding hardware and software resources that must be satisfied. Every computation has its own requirements and every computing machine has its own limitations. All these have to be specified in the trade proposals. In order to address these issues, resource specification languages were developed in the Extensible Markup Language, or XML. [5] deals with this issue in more detail.

2.3.3 Peer List Update Manager Component (PLUM)

In an ideal O.C.E.A.N system a node would be able to communicate with all the peer nodes in the system simultaneously in a broadcast fashion in a reasonable amount of time and make decisions based on the outcome of the communication with all the nodes. But this is not a feasible situation in the real world. The other alternative is to choose a subset of peer nodes through some heuristic and communicate with only the nodes in the subset. This is where PLUM comes in. It is the responsibility of the PLUM component to determine the list of peer nodes with which a given node will communicate.

The PLUM at each node maintains a list of addresses of other O.C.E.A.N nodes (peers) that it knows about, and associated status information about that node (e.g. present & average accessibility, availability, intercommunication bandwidth & latency, etc.). PLUM dynamically updates its list based on the feedback it receives from other components.

2.3.4 Task Spawning and Migration Component

The Task Spawning and Migration (TSM) component as its name indicates is responsible for spawning and migrating the jobs in the Network. After the contract is signed, the TSM component will spawn computing tasks on remote servers through the Security and Communication layers at each node and execute them.
2.3.5 Negotiation Component

This thesis deals with the negotiation component. The auction component matches potential trading partners based on price and provides a list of the matches to the negotiation component. The negotiation component then chooses a trader from the list using some predefined criteria and negotiates with that trader on several items of the deal other than the price. The negotiation process is completely automated. The negotiation component uses a rule-based system to resolve conflicts during negotiation. Though in the initial release of O.C.E.A.N both auction and negotiation components are included as part of the whole system along with the other components, it is planned that in the future the matching and negotiation services will be provided as web services which existing Grid architectures like Globus can make use of. The negotiation component reports the result of the negotiation to the trader component, which will make the necessary decisions to either start migrating the code if the negotiation is successful or indicate to PLUM about the failure of the negotiation.

2.3.6 Security Component

Security is a very important aspect in any distributed system, more so in O.C.E.A.N where remote executions can take place. The problem of providing security is handled by the security component. [9] explores in detail the issues involved in security.

2.3.7 Communication Component

The Communication Component is responsible for communication between the O.C.E.A.N nodes. The component is primarily responsible for data delivery to the other O.C.E.A.N node(s). All information received by the communication component is sent to the security component for verification, before being passed to any other component.
OCEAN Node Architecture

Fig 2.2: O.C.E.A.N Node Architecture
2.3.8 Naming Component

The Naming component is responsible for name resolution of a node in the Network. IP addresses have many issues like dynamic IP address allocation, firewalls, private IP address spaces, single hosts having multiple IP addresses etc. In O.C.E.A.N, each system should be referenced by a single name. The naming component takes care of that. Any resource or process can be located with a O.C.E.A.N name. This is O.C.E.A.N naming convention is shown in figure 2.4:

```
O.C.E.A.N://<public host name or IP address>:<port number>
/<private host name or IP address>:<port number>
/.(any additional hosts on private sub-subnets).
/@<O.C.E.A.N module name>/<jobid>
```

Fig 2.3: Naming System

2.3.9 Local Accounting

The Local Accounting component of each node communicates with the centralized accounting server. When contracts are successfully negotiated, the Local Accounting component will log the payment details locally. When payment is due, the transaction is logged to the Central Accounting server.

2.3.10 Centralized Accounting Server (CAS)

The CAS registers the users and keeps track of the client’s certificate and their private information for verification. In addition to this, it also logs transactions of the business that occurred in the node, both buying and selling. All O.C.E.A.N users register with
CAS before they can join the O.C.E.A.N financial network. Once CAS validates the users, the users are ready to buy or sell resources in the O.C.E.A.N network. The users can configure the CAS through the node configuration API. They can set up CAS to accrue the money in their accounts for small transactions until the sum reaches some threshold and then transfer the money to the external financial institutions like banks.

2.4 O.C.E.A.N Grid – Neutral Architecture

At present, O.C.E.A.N is being implemented as a standalone system with its own communication, security and code mobility system. But on further research it has been found that other Grid systems such as Globus [10] already provide these functions in their systems and it will be years before O.C.E.A.N can provide all the same services. But, it was observed that these systems lacked a market-based architecture, which the higher layers of O.C.E.A.N can provide. So, in order to be competitive, a Grid-neutral architecture of O.C.E.A.N is planned to be developed simultaneously along with the Node architecture explained in the previous subsection. The Grid-Neutral architecture of O.C.E.A.N is indicated in the figure below. The main aim of the Grid-Neutral architecture of O.C.E.A.N is to develop O.C.E.A.N in such a way that the existing Grid architectures can make use of the market-based services of O.C.E.A.N with very little effort. We can see from the figure that the higher level components of O.C.E.A.N provide the trading services and the lower level components of O.C.E.A.N access these services along with other Grid based systems. The advantage of this system is that existing systems can access the services of O.C.E.A.N without any major extra effort. Also, the higher-level components like Auction and Negotiation will be implemented as web services. This decision was made to make use of the extensive web based protocols available and ensure that we don’t reinvent the wheel.
Currently O.C.E.A.N is being implemented as a standalone system, using the “extreme programming” approach. The initial reference implementation is being developed in Java platform. The team members are also exploring the possibility of porting the source code to the .NET platform in the near future to ensure compatibility with larger number of applications.

2.5 Current Status

The iterations of the extreme programming approach were organized as follows.
2.5.1 First Iteration

Develop a minimal working version of the communication component and the Task Spawning and Migration component. Also develop an internal messaging system to be used by the components of the O.C.E.A.N system internally. Finally, integrate all the developed components and test the integrated system.

Status:

All the above tasks have been completed and we now have a working lower level O.C.E.A.N system.

2.5.2 Second Iteration

Develop the Auction component, the Negotiation component and the Trader component. Integrate the components developed in the second iteration with the components developed in the first iteration and test the system as a whole.

Status:

The Auction and the Negotiation components have been developed. Work is still being done on the Trader component and integration.

2.5.3 Third Iteration

Develop the Local Accounting system and the Central Accounting system and hook up the O.C.E.A.N system with external financial networks.

Status:

The interface to one external financial network, eGold, has been completed, and the accounting system is under development. We intend to complete the prototype implementation and release a beta version of the software for public testing by the end of the third quarter of 2002.
3.1 Contracts In e-Business

A contract is an agreement between two or more competent parties in which an offer is made and accepted, and each party benefits. The agreement can be formal, informal, written, oral or just plain understood. A contract can be defined as an agreement of two or more persons, upon a sufficient consideration or cause, to do, or to abstain from doing, some act; an agreement in which a party undertakes to do, or not to do, a particular thing.

Our whole economy is based on the freedom of individuals to contract and a system of laws that enforces contracts freely entered into. Contracts are an integral part of business negotiation, they will spell out the business deals’ details, reduce the likelihood of any misunderstandings between the buyer and the seller, and, in the unfortunate case where contractual obligations are not met, can be used in the court of law as evidence against the offending party.

There are many types of contracts, and many ways in which one can enter into a contract in the human world. Some contracts are required to be in writing to be enforced while some contracts are entered into with simple words or actions like the nodding of the head. But once we enter the electronic world the contract domain becomes much smaller. For businesses that take place over the Internet the contracts have to be concrete, in the form of some documents that are in computer readable format. That way both parties can read and analyze the contracts and come to a decision, all electronically.
According to Morgan Stanley Dean Witter [2], the dollar amount of online business-to-business (B2B) commerce in 2000 was $200 billion and it would reach $720 billion by 2001 and $1.4 trillion by 2002. Given the huge amount of online business going on it is a wonder that there are not many efforts to come up with standards for something as important as contracts in the e-business world till now. EbXML [11] provides a way to define contracts, but it means contracts in a different sense of the word. It means contracts as something that the trading partners have to agree upon before starting the main negotiation phase.

As of today, most of the online transactions are handled through credit cards and the credit card company handles any fraud. There is no way a party can hold the opposite party accountable for a sale made over the Internet. Even large online businesses that hold auctions, like e-bay, can’t give a guarantee on any sale. Most of the business online is based on the trust model. All the parties trust each other and hope nothing goes wrong, and in the unfortunate event of something going wrong the blame is placed on the credit card companies. But with contracts, the trading parties have concrete evidence of the sale and can approach the law for justice in case anything goes wrong. It is with this thought in mind that this thesis puts forward a specification for writing contracts for online businesses. The specification is in the form of an XML schema and it is called the Contract Specification Language or CSL. It is hoped that in the future the CSL is standardized and trading parties will make use of the specification for writing contracts and maybe even use the CSL documents as evidence in the court of

3.2 XML And SOAP For Contracts

Now that we have established the need for contracts in the e-business world we need to explore ways how and in which format these contracts can be constructed and which
messaging framework can be used to communicate the contracts between the trading agents.

The Extensible Markup Language, XML [5] was used to construct the contracts for the following reasons:

1. XML provides a way to convey metadata. So when a contract is written using XML all the business data as well as the information describing the business data is included in one document.

2. XML is entirely text based and so it is completely platform independent. This is important because applications running on different platforms may want to do business with each other.

3. XML is designed to run over http connections on the Internet, which fits in nicely with the idea of e-business.

4. XML is a very readable format. The final XML document is just like a normal document with just some extra tags. The importance of readability is much more important than it seems. In most business deals the final say belongs to someone who is not a computer expert and they prefer to have an easily readable contract.

XML schemas [12] were used to define the Contract Specification Language. The XML schema specification defines the format taken by an XML schema. XML schemas define the legal structure of a set of XML documents. An XML document that conforms to an XML schema is said to be valid with respect to the schema. Therefore, by using schemas we can define the structure of a contract and specify what elements it must contain and all the applications that conform to the contract schema have to follow the specification of the schema and all contracts will have the same structure. This is a clean way to enforce standardization. Also, by using schemas it becomes much easier to generate or parse contracts because we have well established set of rules specified in the schemas.
Another advantage of using schemas is that the schemas are extensible. We can specify a base set of elements that have to be in a contract document and other applications can build on top of that to provide their own set of elements depending on the business needs of that particular application. That way there will be a global standard for contracts that all applications have to conform to and a smaller set of local standards built on top of the global standard that each application domain can enforce. This will become clearer as I will show some examples in the next subsection. At the time of this writing the O.C.E.A.N group is administering the contract schemas. But, it is desirable that in the near future some non-profit standards body like the World Wide Web Consortium can embrace the contract schema and promote it as the standard for contracts in e-business negotiations.

SOAP for Messaging:

SOAP version 1.2 [6] is a lightweight protocol for exchange of information in a decentralized, distributed environment. According to the W3 specifications:

SOAP version 1.2 provides the definition of an XML document which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. It is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying transport protocol and/or application-specific information.

The contract schema in this thesis conforms to the SOAP specifications. The recent support for SOAP by large organizations like Microsoft, the adoption of SOAP specifications by some other major e-business models like ebXML [11] and the ease with which we can conform to the SOAP specification all lead to the decision to support the
SOAP specifications in the contract schema. In its present form the contract existing SOAP processors can process schema, which makes it available to a larger audience.

3.3 Contract Schema

Before we go ahead and define the contract schema we have to make sure of what are the necessary elements in a contract. The contract should capture the details of the deal clearly and exhaustively so that all the trading partners are clear about the details and nothing is left for assumption.

3.3.1 Essential Elements of a Contract

The following are considered as essential elements in a contract:

1. **Contract ID:** Every contract should have a unique id using which we can identify the contract. The contract id can also be used as a key to search a contract in a database. The format of the contract ID hasn’t been agreed upon as of the present implementation.

2. **Parties:** The full names of all the trading parties have to be included in the contract. Parties may be individuals, corporations, governmental entities, or other business entities. There should be no ambiguity about the names. This is very important because identifying the other party using an inaccurate or incomplete name may result in the contract being unenforceable against the party with whom you thought you were doing business. In an e-business world, the name of a party need not be the legal name of the party involved as long as all the parties are clear about who they are trading with and if the name is unique in an application domain,
for example: a user name in O.C.E.A.N. It is not acceptable for the
parties' names to appear differently in the first paragraph of the
contract and on the signature line.

3. **Effective Date:** The contract should have a date stated as the
contract date or effective date. This date is not necessarily the date
when the contract was signed but rather the date from which
contractual rights and obligations begin, and from which any term
of time usually begins.

4. **Items:** The contract should clearly state what items are being
negotiated. The items negotiated can be concrete (ex: a painting) or
abstract (ex: an idea). In an e-business world most of the times we
will require a resource specification language to specify the items
being negotiated. But everything is fine as long as the parties
involved in the negotiation understand the resource specification
language. Trading parties can use the language of their preference
to specify the resources though we advice the users to use XML
schemas for resource specification.

5. **Payment Information:** The payment information should include
how much is being paid and how it is being paid. Ex: Is the
payment type a subscription model or is it a one-time payment?

6. **Contract Break Action:** This item specifies what action to take in
case a contract is not fulfilled by any of the trading parties.
7. **Signatures:** This is a very important part of the contract. The signature indicates the consent of the parties to the deal. The signatures should be unique and should not be susceptible to fraud. That is, no one should be able to forge a signature. It is proposed that XML signatures [13] be used for this purpose.

Some other elements that are useful but not essential to have in an e-business contract are:

1. **Application name:** Each kind of application of the contract language will have its own business needs. So if we include the application name in the contract, the application will know how to handle the contract depending on the application. The application name also sets the context for interpreting other fields.

2. **Techniques:** When applicable, the technique as to how materials will be applied should be detailed. For example: If a deal is being negotiated to run a job on a remote computer, then the owner of the job may specify what resources the job will use.

Now that we know what we need in a contract we can build the Contract Schema. The beauty of XML is that each of the above elements can be included directly as elements of an XML document as is shown below. As an example the contract schema for O.C.E.A.N is developed which includes all the necessary elements and adds some elements that are
specific to the O.C.E.A.N domain. In a similar manner contract schemas for any application can be developed just as easily.

**Note on Graphical Notation:**

The following sections contain figures that were generated with the XML Spy Integrated Development Environment [14]. Figure below shows a legend for some of the diagramming components used.

Fig 3.1: XML Schema legend diagram

3.3.2 Contract Schema

The complete schema, an example XML document and the schema documentation generated by XML Spy software are printed in the appendix.

1. **SOAP Envelope:** This is the mandatory SOAP Envelope that all SOAP compatible documents must have. The schema for the SOAP envelope is provided in [5]. This is the root element of the document. It has got Header and Body child elements.

Fig 3.2: Envelope element structure
a. **SOAP Header:** This is the optional header element of a SOAP message. But in the contract schema this element is mandatory. The header contains important information like the names of the parties, the contract date, the application name etc.

![Header element structure diagram]

Fig 3.3: Header element structure

i. **Application:** This element is a child of the SOAP Header. It contains the name of the application this contract belongs to. Any XML parser would first parse this node to find out the name of the application the contract belongs to and then handle the contract according to the needs of the particular application. For example: The value of this node for the O.C.E.A.N application would be “O.C.E.A.N”. Once the processor finds the name O.C.E.A.N in the application element, it knows to handle the contract as a contract belonging to the
ii. **Contract ID**: This is the unique contract id generated for each contract. Since the contract ID is a very important part of the contract it was thought that it would be best to postpone the design of the contract ID format until we were completely sure of the O.C.E.A.N architecture.

iii. **Buyer**: The name of the buyer. In O.C.E.A.N this will be an O.C.E.A.N client id.

iv. **Seller**: The name of the seller. In O.C.E.A.N this will be an O.C.E.A.N client id.

v. **Negotiation Protocol**: This element is specific to O.C.E.A.N. Only O.C.E.A.N contracts will have this element in the header. In O.C.E.A.N the contract is also used as a document to negotiate deals between parties. Therefore we need an element that specifies the type of the Negotiation Protocol agreed upon by the parties. More details on how the negotiation
is done will be given in the next chapter. In the present implementation, the only value allowed for the NegotiationProtocol element is “YesNo”. In future implementations we might add more negotiation protocols like a “Bargain” protocol etc.

vi. **Status:** This is also an O.C.E.A.N specific element. This element is used to indicate the status of a negotiation. For now, the only values allowed are “Accepted”, “Rejected”, and “Negotiate”.

vii. **Date:** This element contains the Contract date.

b. **SOAP Body:** This is the mandatory body element of the SOAP message. This element contains information about the items being negotiated, the price etc. For the present implementation the elements given below are included and it is expected that more elements will be included in the future, as the need arises.

![Fig 3.4 Body element structure](image-url)
i. **Negotiation Items:** This contains the list of items being negotiated. Each item is represented by an `<Item>` tag and each `<Item>` element contains a reference to an XML document which contains the names and prices of the items being negotiated, or contains resources as XML elements inline to the contract. This design was decided upon to provide different applications the independence to specify the resources being negotiated using their own resource specification languages. In O.C.E.A.N the language for negotiation items is specified in the context of the auction component. For more information on how the auction system works see [8].

Example:

```xml
<NegotiationItems>
  <Item>
  </Item>
</NegotiationItems>
```

Fig 3.5: NegotiationItems element structure
ii. **Payment Information:** This element contains information about the payment. It contains information on how the payments will be made, what type of payment method will be adopted etc.

For example:

```
<PaymentInformation>
  <Mode>Subscription</Mode>
  <Type>Paypal</Type>
</PaymentInformation>
```

Fig 3.6: PaymentInformation element structure

For now the only types allowed for `<Mode>` are “subscription” and “oneshot” and the only types allowed for `<Type>` are “paypal” and “eGold”. In future versions of O.C.E.A.N more options can be provided. It should be noted that this element needs more options and variations ultimately.

iii. **Contract Break Action:** This element specifies what action to take in the event a party breaks the contract. In the present implementation the only value allowed for this element is “noAction” and it is planned to provide another option, “kickOut”
which will kick the offending party out of the O.C.E.A.N system by removing the party for central authorization service, in the near future.

iv. **Signature:** This is the final element of the contract and contains the signature of the parties. Presently, the name of the party itself is used as the signature. In future implementations the XML signature technology will be used to develop unique signatures for all the parties involved.

In no way is the above list of elements for O.C.E.A.N exhaustive. The elements listed are sufficient for the present implementation of O.C.E.A.N and as O.C.E.A.N evolves, so will the list of elements to be negotiated. As more necessary elements are identified they can be added to the O.C.E.A.N contract schema. This is where the extensibility and flexibility of XML comes in handy. The next chapter explains how the Contract schema can be employed in the Negotiation process of O.C.E.A.N.
4.1 Negotiation States

Negotiation is more of an art than a science. According to Pruitt [15], negotiation is defined as the process by which a joint decision is made by two or more parties. The parties first verbalize contradictory demands and then move towards agreement by a process of concession making or search for new alternatives.

There are no set rules as to how to conduct a negotiation. In the real world, negotiation takes many forms. Some negotiations are by word of mouth while some are through full-fledged contracts with lawyers presiding over them. Many times negotiations depend on the personalities of the negotiators. Some negotiators are hard bargainers, while some are soft bargainers. This does not bode well for automating the process of negotiation in computers because the process of negotiation does not involve a specific set of rules that a computer system can follow. Instead many negotiations are done on the spur of the moment depending on the circumstances. Thus in order to construct a negotiation system which replicates the way humans do negotiations we might have to incorporate personalities in the computer system.

In a typical negotiation two parties come together. One party expresses its wish to buy some goods that the other party has. Then the two parties negotiate and agree upon a price for which the goods will be sold. We can add more detail to the negotiation process, like negotiating for the delivery date and quality of the products along with the price and so on. A negotiation ends when both parties are satisfied that they have got a good deal.
Some negotiations are not feasible for automation, like the negotiations for peace agreements. Such types of negotiations are best left to people. The use of information technology is not necessary for such type of negotiations. However, some other types of negotiations, like the e-business negotiations that occur over the Internet are much easier to automate.

The use of automated negotiations is not a new concept. Jelassi et al. [16] provides some good reports on design issues and existing software for Negotiation support systems. Oliver [17] gives us a machine learning approach for automated negotiations. There are automated negotiators like the Michigan Internet Auction Bot [18], The University Of Florida’s Negotiation Server [7], FM96.5 [19] etc. But, these are heavy-duty negotiation servers that have been designed to address the negotiation requests of varied kinds. It is our contention that since negotiation is a very specialized process with different needs for different kinds of products, it is best if the Negotiation components are developed depending on our needs. For example, if we are negotiating on behalf of a Fortune 500 company to settle a deal of millions of dollars then the overhead generated by the above mentioned negotiation servers might be justified. But on the other hand if we are negotiating on behalf of consumers who are buying $5 worth of computation on the Internet, then the overhead generated by the negotiation servers may not be justified. Keeping this in mind a simple negotiation component for O.C.E.A.N is proposed.

The following activities are identified as being necessary for a normal negotiation:

1. Finding trading partners.
2. Negotiating terms and conditions between the trading partners.
3. Signing the contracts if the negotiation is successful.
4. Completing the sale.

In the course of a negotiation process, the participating traders will be in one of the four states at any given time. Let us now see what these states are in more detail, how these states fit in the O.C.E.A.N architecture and how the process of negotiation is carried on in O.C.E.A.N.

The state diagram for the four states is given below.

![Negotiation Activity state diagram](image)

1. **Find Trading partners:** The distributed PLUMs do the job of finding servers, and the auction component conducts auctions among them to find a subset of servers with which we can negotiate. It is up to the negotiation component to negotiate with the different servers.

2. **Negotiate terms and conditions between the trading partners:** This state is the heart and soul of the negotiation process. This job
is carried out by the negotiation component in O.C.E.A.N. For a negotiation to be successful the trade proposals of the traders should have the 3Es. They are:

a. **Expressiveness:** The trade proposal should be able to clearly show what the traders are negotiating about and what each trader expects from the trade. The buyer should be able to clearly indicate what he is buying and the seller should be able to clearly express how much he is expecting for his product. There should be no ambiguity. For this we need an expressive language.

b. **Extensibility:** The language with which we form the trade proposals should be extensible so as to include new features in a negotiation that cannot be foreseen at this time and which are not part of general negotiations.

c. **Easy to understand:** The language used to form trade proposals should be easy to understand and learn so that even non-programmers can form the trade proposals with little difficulty. This is necessary because the people who form trade proposals are usually not programmers but businessmen. All the above qualities are present in the Contract language we specified in the previous chapter. Information on how negotiation is performed in O.C.E.A.N is explained later on in the section.

3. **Signing of Contracts:** This is a very important part of the trade. Once a negotiation is successfully completed both parties have to
sign a contract showing their commitment to make the trade. The contract can be used as a proof of purchase and can be called upon whenever there is any dispute on the sale. Since in O.C.E.A.N we are using XML documents for negotiation it is proposed that we use XML signatures [13] for signing a contract, which is also an XML document. The general structure of an XML signature is given below:

```xml
<Signature>
  <SignedInfo>
    <CanonicalizationMethod>
      <SignatureMethod>
        <Reference (URI=)? >
          (Transforms)?
          <DigestMethod>
            <DigestValue>
              </Reference>
            </DigestMethod>
          </Reference>
        </SignatureMethod>
      </CanonicalizationMethod>
    </SignedInfo>
  <SignatureValue>
    <KeyInfo>?
      (Object)*
    </SignatureValue>
  </Signature>
```

An example XML digital signature is given below:

```xml
<Signature Id="MyFirstSignature"
  xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference
      URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126="/>
    <Transforms>
      <Transform
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315="/>
    </Transforms>
```

<DigestMethod
    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

<DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=/</DigestValue>
    </Reference>
</SignedInfo>

<SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
    <KeyInfo>
    <KeyValue>
    <DSAKeyValue>
    <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
    </DSAKeyValue>
    </KeyValue>
</KeyInfo>
</Signature>

More information of XML digital signatures can be found at [13]
Versign has an API to generate XML signatures and though others are also in the
process of developing API for XML signatures no standard API has yet evolved. So for
the first version of O.C.E.A.N the O.C.E.A.N user name will be used as a signature. The
XML signatures are planned to be implemented in future releases.

4. **Complete the sale:** Once the contract is signed the code is
migrated to the user. The Task Spawning and Migration
component in O.C.E.A.N perform this task.

Now that all the protocols and standards are in place the next section explains in
detail the negotiation process in O.C.E.A.N.

4.2 YesNo Protocol And Non-SOAP Based Negotiation In O.C.E.A.N

In the first version of O.C.E.A.N it is assumed that the Auction component will take
care of matching the potential buyers and sellers based on price. The Negotiation
component negotiates for the other elements of a deal like the Payment Type, the
Contract Break Action etc. A basic non-SOAP based version and a SOAP based version
of the Negotiation component have been implemented. The non-SOAP version implements the “YesNo” protocol and the SOAP version implements the “YesNo” protocol as well as the “StaticBargain” protocol, which are also explained below. The rest of the section is structured as follows:

First a brief description of “YesNo” protocol is given in section 4.2.1. Then, the implementation of the protocol in the non-SOAP based version of the Negotiation component is explained in 4.2.2. In the next chapter, a brief description of “StaticBargain” protocol is given and the implementation of the SOAP based version of the Negotiation component is dealt with.

4.2.1 “YesNo” Protocol

1. Buyer initiates the deal.

2. The Buyer sends his offer in the Contract using the Contract Specification Language.

3. The Seller verifies the Contract and checks to see if all the elements in the Contract match his specifications.

4. If everything is in order the seller updates the status of the Contract to “Accepted” and sends back the Contract to the buyer. If something is wrong, that is, if some elements in the Contract don’t match, then the seller updates the status of the Contract to “Rejected” and includes the rejected items in the Contract along with a reason why they were rejected and sends back the updated Contract to the Buyer.

5. The Buyer receives the updated Contract and checks the status of the Contract. If the Contract is “Accepted” then the Buyer signs the Contract and sends the signed Contract back to the Seller. If the Contract is
“Rejected” the Buyer either goes onto the next Seller in the list or tries to take care of the rejected items and sends back an updated offer back to the same Seller. The Seller treats this updated offer as a new offer.

6. If the Seller receives a signed Contract, it makes all the necessary checks again and if everything is in order countersigns the Contract and sends it back to the Buyer. If anything is wrong the Contract status is updated to “Rejected” and sent back to the Buyer.

7. When the Buyer receives the countersigned Contract it takes the necessary steps to start the deal.

As you can see the YesNo protocol has two phases. The trading partners come to agreement on the contract in the first phase. The signing of the contracts constitutes the second phase. This design was decided upon to provide the buyer with more leverage. A buyer can simultaneously be negotiating in the first phase with several sellers at the same time. Once he gets back several offers the buyer can then choose a seller according to his liking and seal the deal with that seller by signing the contract.

The state diagrams of the Buyer and the Seller in the YesNo protocol are given below. The state diagrams are self-explanatory. As can be clearly seen, the buyer is active and the seller is passive in the sense that the buyer initiates the deal and the seller will be waiting for someone to initiate the deal.

4.2.2 Non-SOAP Based Negotiation Component

The Auctioning system determines who the potential trading partners are. The Negotiation system then provides the means to allow traders to agree on the terms of a contract (a service agreement). This includes resolving conflicts that arise when a trader
has many potential trading partners. The negotiation component tries to automate the process of negotiation as much as it can. In any event, it also provides methods so that humans can intervene and take over the job of negotiation, if necessary, for a high-value transaction.

The Negotiation component primarily makes use of the Auction component, through the Trader component. Like most node components, it uses the security and
communications component for communications to other nodes on the network. Like all node components, it supports a configuration & maintenance interface for use by the node operator. Its network communications are primarily with the Negotiation components of other nodes. The Negotiation component's role, as its name implies, is to negotiate the terms of the deal with its peer nodes before Task Migration takes place.

When the user starts up the system he will be shown an initial configuration screen as shown below.

The user sets up his information depending on whether he is acting as a Buyer or a Seller. The information entered by the user is stored in a file, for example, BuyerNegotiationConfiguration.xml if the user is acting as a Buyer and in SellerNegotiationConfiguration.xml if the user is acting as a Seller in the configuration directory.

Fig 4.4: Initial Configuration Screen
Once the negotiation process is started the negotiation component of the Buyer gets a list of potential trading partners from the auction component and chooses a particular Seller to negotiate with. The negotiation component then gets the relevant information from the BuyerNegotiationConfiguration.xml file and forms the contract according to the Contract Specification Language. The negotiation component then calls upon the communication component to deliver the Contract to the Seller. In the first version of O.C.E.A.N it is assumed that all trading parties follow the “YesNo” protocol. If necessary an application can define its own negotiation protocol depending on its application specific needs. So, both the Buyer and Seller follow the “YesNo” protocol explained above, to negotiate. The Seller will receive the Contract and compare it to his SellerNegotiationConfiguration.xml file and make a decision based on the result of the comparison. Finally, an agreement is reached between the Buyer and the Seller or the Buyer moves onto the next Seller in its list.

The advantage of the YesNo protocol is its ease of use. It is also very easy to understand and the results are fast. The Contracts are matched with the initial configuration files and we get a Yes or No answer fast.

Its disadvantage is that it’s not flexible and the probability of getting a match is not high, as the comparison will fail even if one element in the contract does not meet the requirements. That is where the “StaticBargain” protocol comes in. The StaticBargain protocol is flexible and users can specify their own rules. The StaticBargain protocol is explained in the next chapter. An example implementation of the YesNo protocol is shown in the last chapter.
CHAPTER 5
SOAP BASED NEGOTIATION COMPONENT IN O.C.E.A.N

Though the initial version of O.C.E.A.N is being developed as a stand alone system with its own communications system, security system, code migration system etc, the ultimate goal of O.C.E.A.N is to develop it as a web based service which existing Grid Computing architectures like Globus [10], Parabon [20] etc can use. Existing systems like Globus have their own communication, security and code migration systems. What they lack is a market-based architecture that O.C.E.A.N provides as a web service. The existing systems can use O.C.E.A.N to find potential trading partners and then use their own internal systems for communications and code migration.

To achieve this end the major technologies planned on being used by O.C.E.A.N are

1. SOAP [5]
2. WSDL [21]
3. UDDI [22]

SOAP:

According to W3C

SOAP version 1.2 provides the definition of an XML document which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment.

Though SOAP is fundamentally a stateless, one-way message exchange paradigm, applications can create complex interaction patterns like Request/Response, Request/Multiple Responses by combining such one-way exchanges with features provided by an underlying transport protocol and/or application-specific information.
SOAP does not specify any application specific semantics, how SOAP messages have to be routed, how to transfer data reliably or how to traverse firewalls. What SOAP provides is a framework by which application-specific information may be conveyed in an extensible and platform independent manner.

**WSDL:**

Suppose a seller in O.C.E.A.N wants to give web-based access to a set of services and depending on the request the seller might return a SOAP response, an XML document or an image. The seller should have some way of letting the buyers know the type of services he is offering. To do that, the seller and the buyer either have to write their own discovery mechanism or use WSDL and a UDDI registry. WSDL stands for Web Services Description Language. In plain terms, WSDL is an XML format for describing network-based services.

**UDDI:**

It is believed that the ability to discover and use web services will make the integration of disparate connected systems easier. To do this a standard must exist and the location of the information has to be well known. In order to facilitate this UDDI was proposed. UDDI stands for Universal Description, Discovery and Integration. UDDI specifies what the API for a web based registry looks like. Anyone can implement one of these and any client can access these services. A site that implements the UDDI registry is called an operator site. It is generally assumed that, in order to encourage the use of these services the basic lookup services are provided for free to clients while the more advanced services like the advertisement of a service on the site are paid for.
Since the YesNo protocol was explained in the previous chapter in section 4.2.2, we will now look at the StaticBargain protocol and then see how it is used in the SOAP based Negotiation component.

5.1 “Static Bargain” Protocol

1. Buyer initiates the deal.

2. The Buyer sends his offer in the Contract using the Contract Specification Language.

3. The Seller verifies the Contract and checks to see if all the elements in the Contract match his specifications in his Rules document. Every trader in O.C.E.A.N who wishes to use the StaticBargain protocol must specify some rules in a rules document, which is also an XML document. The rules in the rules document are used to decide what to bargain for and how. More information on rules is given later on in the chapter.

4. If everything is in order the seller updates the status of the Contract to “Accepted” and sends back the Contract to the buyer. If something is wrong, that is, if some elements in the Contract don’t match, then the seller updates the status of the Contract to “Negotiate” and includes the rejected items in the Contract along with a reason why they were rejected and sends back the updated Contract to the Buyer.

5. The Buyer receives the updated Contract and checks the status of the Contract. If the Contract is “Accepted” then the Buyer signs the Contract and sends the signed Contract back to the Seller. If the Contract is “Negotiate” the Buyer looks at the rejected items and it’s rules document and decides to either go onto the next Seller in the list or try to take care of
the rejected items and send back an updated offer back to the same Seller. The Seller treats this updated offer as a new offer.

6. If the Seller receives a signed Contract, it makes all the necessary checks again and if everything is in order countersigns the Contract and sends it back to the Buyer. If anything is wrong the Contract status is updated to “Rejected” and sent back to the Buyer.

7. When the seller receives the countersigned Contract it takes the necessary steps to start the deal.

The state diagrams of the seller and buyer for the StaticBargain protocol are given below:

![Buyer State Diagram - StaticBargain](image)
5.2 Rules In Negotiation

In the StaticBargain protocol a rules document, which is an XML document, is used to make dynamic decisions during negotiation. When the user selects StaticBargain protocol as his negotiation protocol in the initial configuration screen, the user will be presented with a sequence of screens where he sets up his negotiation rules. Once the rules have been set up the user makes his negotiation decisions based on the rules document. The Java Expert System Shell, JESS [23], is planned on being used as the interpreter for the rules. Once O.C.E.A.N is started up the JESS rule engine also starts up. The rule engine then looks at the rules document and builds facts from the rules document. When the negotiation is going on the rule engine will make decisions and take actions based on the
facts. The advantage of using an expert system shell is that we can make the system to learn from previous negotiations. An example rules document is given below:

```xml
<Rules>
  <If>
    <Element>PaymentType</Element>
    <Operator>=</Operator>
    <Value>egold</Value>
    <Then>
      <ChangeElement>PaymentType</ChangeElement>
      <ChangeValue>paypal</ChangeValue>
    </Then>
  </If>
</Rules>
```

The head element of a rules document is `<Rules>`. The rules in the rules document are stored under the element `<If>`. The elements in the file should be self-explainatory.

Basically, what the above file says in plain English is:

“If the value of PaymentType in the received contract is egold, then add PaymentType as a rejected element and set the expected value of PaymentType to PayPal”.

### 5.3 SOAP Based Negotiation Component

The buyer searches the UDDI registries for sellers offering their services. As soon as the buyer finds some compatible sellers, the buyer invokes his negotiation component that will negotiate with the sellers using SOAP messages.

When the user starts up his system he will be shown an initial configuration screen where he sets up his initial parameters as explained in the previous chapter. In addition to the previous parameters, the user also sets up his rules for negotiation in a file called “Rules.xml”. Let us look at how the seller and buyer work in the SOAP based system.
5.3.1 Seller

The seller describes his services in a Web Services Description Language file and advertises them to a Universal Discovery, Description and Integration registry. Since the seller service is a web service the seller will be running as a servlet inside a web server, like Apache Tomcat 4.0 [24], which will be installed simultaneously on the users system when the user installs O.C.E.A.N. After the seller advertises its services, the seller will wait listening for any incoming offers.

5.3.2 Buyer

When a user wants to buy some services, O.C.E.A.N will look in several UDDI registries to find compatible sellers. Once a set of compatible sellers is found, they are ordered based on some predefined criteria and the ordered set is given to the Negotiation component. In the present version of O.C.E.A.N it is the Auction components job to order the set of compatible sellers based on price and the Negotiation Components job is to negotiate for other items of a deal like payment method, payment type etc.

Once the negotiation component gets an ordered set of sellers, it will start negotiating with the sellers one by one in the preferred order and the path of the negotiation is determined by the negotiation protocol chosen by the trading parties. The buyer and the seller will use their rule documents to make negotiation decisions dynamically and try to work towards reaching an agreement. If they are not able to reach an agreement the buyer moves onto the next seller in it’s list and the seller goes back to listen for more incoming offers.

By using the rule document in the StaticBargain protocol we are able to capture the individuality of each user, as each user will set up his rules differently based on his needs. Thus, flexibility is attained using the StaticBargain protocol.
In the present implementation of O.C.E.A.N the PLUM, and the Auction components handle the discovery of traders and the actual negotiation is carried out as explained above, using SOAP messages. Using WSDL and UDDI for discovery is planned on being possibly implemented in future releases.
CHAPTER 6
NEGOTIATION API AND EXAMPLE IMPLEMENTATION

For any software system to be successful it should provide a rich API to allow programmers to work with the system and it should provide an easy user interface for users to use the system. Developers who will work on O.C.E.A.N will need an extensive API to extend the capabilities of the system and application programmers will need the API to write applications that can be run on O.C.E.A.N. The Negotiation API was developed with this thought in mind. The Negotiation component provides APIs to generate Contracts, send and receive Contracts as SOAP messages, develop rules for negotiation and handle XML documents. The Negotiation component also provides an easy to use GUI interface for users to set up their initial configuration files and use the system. This chapter explains the API provided by the Negotiation component and will show through an example implementation how easy it is to develop Contracts and negotiation rules using the API and GUI provided by the Negotiation component.

6.1 Information Gathering

When the users first start up the system they will be shown an initial configuration screen as shown below.

When the users click on Submit Buyer/Seller info button, the data entered by the user is stored in a corresponding configuration file. The basic class that does the job of collecting the user information and storing it into an XML document is the NegotiationFrame class. The class diagram of NegotiationFrame class is given below. All
the class diagrams are according to the Universal Modeling Language (UML) [25] specifications.

The NegotiationFrame class uses the XMLUtils class, which provides basic utilities to build XML Document objects from an XML file and write the Document objects back to stable storage. We use the JDOM [26] API to manipulate XML documents. The JDOM API is still in the beta phase, but it provides very useful wrapper classes to easily manipulate XML documents in java and is one of the most used APIs to parse XML documents.

![Initial Negotiation Configuration screen](image)

Fig 6.1: Initial Negotiation Configuration screen
For example, if the user selects the information as shown in the picture above and clicks the Submit Buyer Info button the following XML document will be generated and stored in a file called BuyerNegotiationConfiguration.xml, in the configuration directory of the user.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<NegotiationConfiguration>
  <UserName>chetan</UserName>
  <NegotiationProtocol>YesNo</NegotiationProtocol>
  <PaymentInformation>
    <Mode>subscription</Mode>
    <Type>Paypal</Type>
  </PaymentInformation>
  <ContractBreakAction>Nothing</ContractBreakAction>
  <NegotiationAttempts>3</NegotiationAttempts>
</NegotiationConfiguration>
```
If the user selects StaticBargain as the negotiation protocol in the initial configuration screen, he will be presented with another screen as shown below where the user sets up his negotiation rules.

![Image of negotiation screen](image)

Fig 6.3: StaticBargain Configuration screen 1.

The GUI will guide the user to set up the rules by showing him one element at a time. As soon as the user gives the required input the next element is shown. This interaction is depicted in the sequence of pictures shown below.

A more intuitive GUI will be developed for later versions. The rule system basically works on existing elements of the contract providing rules on how to act depending on the values of the contract elements. Also, more work should be done on error checking, that is, checking to see if the user enters contradictory rules etc.
Fig 6.4: Negotiation Configuration screen 2

Fig 6.5: Negotiation Configuration screen 3
The main class responsible for gathering the user input is the StaticBargainFrame. It makes use of the helper class RuleBuilder provided by the NegotiationComponent. The class diagram of RuleBuilder is given below.

![RuleBuilder class diagram](image)

It should be noted that though StaticBargainFrame uses one method to gather user input, developers are free to develop their own method of gathering user input and use the utility methods provided by the RuleBuilder to build rules according to the O.C.E.A.N specification. The document formed by entering the input as shown above is given below.
<?xml version="1.0" encoding="UTF-8"?>
<Rules>
  <If>
    <Element>PaymentType</Element>
    <Operator>==</Operator>
    <Value>paypal</Value>
  </If>
  <Then>
    <ChangeElement>PaymentType</ChangeElement>
    <ChangeValue>egold</ChangeValue>
  </Then>
</Rules>

This document is stored in a file called Rules.xml in the configuration directory of the user.

Let us now see how the information gathered is used to build Contracts and used in negotiations in the Buyer.

6.2 Buyer Implementation

6.2.1 Contract Building And Negotiation

The main class used to build the contracts is the Contract class. The main purpose of the contract class is to abstract the handling of SOAP messages completely away from the programmer so that he can concentrate on building complex negotiation strategies.

```java
Contract
- ocean : String = "http://www.cise.ufl.edu/mseam/ocean"
- negotiationAttempts : int
+Contract() : Contract
+generateContact(buyerName : String, sellerName : String, application : String, c : Document)
+addContractHeaderElement()
+addContractBodyElement()
+getContractHeader()
+getContractBody()
+getContract()
+getLocalName()
+getProtocol() : String
```

Fig 6.8: Contract class diagram

The class diagram for the Contract class is given above.

The single method generateContract(String buyerName, String sellerName, String applicationName, Document configDocument) will generate the contract for the user in
the form of a SOAP message. The class also has utility methods to access the individual elements of a Contract once it is formed. If the programmer wants to, then he can generate the Contract using JAXM API [27] provided by java. But it is recommended to generate the Contract by calling the generateContract() method because by calling this method we can be sure of generating a contract which conforms to the O.C.E.A.N contract specifications. For example, when we call

generateContract("chetan", "frank", "O.C.E.A.N", config), where “config” is the XML document generated from “BuyerNegotiationConfiguration.xml”, the following Contract is generated:

<soap-env:Envelope
xmlns:soap-
 env=http://schemas.xmlsoap.org/soap/envelope/
xmlns:O.C.E.A.N="http://www.cise.ufl.edu/research/O.C.
E.A.N"
>
  <soap-env:Header>
    <O.C.E.A.N:ContractID/>
    <O.C.E.A.N:Application>O.C.E.A.N</O.C.E.A.N:Applications>
    <O.C.E.A.N:Buyer>chetan</O.C.E.A.N:Buyer>
    <O.C.E.A.N:Seller>frank</O.C.E.A.N:Seller>
    <O.C.E.A.N:NegotiationProtocol>YesNo</O.C.E.A.N:
 NegotiationProtocol>
    <O.C.E.A.N:Date>Fri Apr 05 16:00:57 EST
  2002</O.C.E.A.N:Date>
  </soap-env:Header>
  <soap-env:Body>
    <O.C.E.A.N:NegotiationItems>
      <O.C.E.A.N:Item>O.C.E.A.NTask</O.C.E.A.N:Item>
    </O.C.E.A.N:NegotiationItems>
    <O.C.E.A.N:PaymentInformation>
      <O.C.E.A.N:Mode>subscription</O.C.E.A.N:Mode>
      <O.C.E.A.N:Type>Paypal</O.C.E.A.N:Type>
    </O.C.E.A.N:PaymentInformation>
    <O.C.E.A.N:ContractBreakAction>Nothing</O.C.
  E.A.N:ContractBreakAction>
  </soap-env:Body>
The logic for negotiation is handled by the methods in the BuyerNegotiationLogic class. In the present implementation there are just two methods called YesNo() and bargain() which handle the logic for the YesNo protocol and the StaticBargain protocol respectively. Though the YesNo protocol has been fully implemented the StaticBargain protocol is still in the development phase. It should be noted that developers could easily extend this class by providing their own methods to handle different negotiation logic according to their business needs.

6.2.2 Contract Sending

The methods in the BuyerNegotiationLogic class use the ContractMessenger class to send and receive SOAP messages to the seller. The class diagram of the ContractMessenger class is given below.

<table>
<thead>
<tr>
<th>ContractMessenger</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ContractMessage(): ContractMessage</td>
</tr>
<tr>
<td>+send(String, con)</td>
</tr>
<tr>
<td>+getResponse()</td>
</tr>
</tbody>
</table>
| +getResponseHeaderElements() : List<
| +getResponseStatus() : String |

Fig 6.9: ContractMessenger class diagram

The method send() sends a given Contract to the specified seller and the method getResponse() returns the response received from the seller.

The entire class structure of the Buyer as generated by Poseidon for UML, is given below. Poseidon is a very handy tool that generates UML diagrams from java code.

6.3 Seller Implementation

The ContractReceive class is a servlet on the seller, which will be waiting for any incoming offers. As soon as the server receives a Contract the onmessage() method of the
Fig 6.10: Buyer Class Structure

The servlet is called. The servlet then makes use of the class SellerContractUtils to parse the Contract. SellerContractUtils also provides many utility methods to access individual elements of the Contract as can be seen from the class diagram given below.

Like the Buyer, the seller has a SellerNegotiationLogic class, which will take care of the negotiation logic for the Seller. The SellerNegotiationLogic class will apply the negotiation rules of the seller to the received Contract and send back the Contract to the buyer after updating it with the outcome of negotiation.
Fig 6.11: SellerContractUtils class diagram
CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

This thesis presented a set of standards and specifications using which one can easily develop an application specific negotiation component. An example implementation was also developed using the O.C.E.A.N system as the test bed.

Chapter 2 explained the O.C.E.A.N architecture and also showed how the negotiation component fit in with the whole O.C.E.A.N architecture. The O.C.E.A.N system is still in its initial stages of development, but we plan on releasing an alpha version of O.C.E.A.N by the third quarter of 2002.

In Chapter 3 e-business contract specifications were explored. Essential elements in a contract were identified and incorporated into the contract specifications. The contract specification language (CSL) is a very important part of e-business because it provides the traders the ability to define contracts, which in turn can act as evidence of a trade. Of course, the contract specification language also enjoys the advantages provided inherently by the XML schemas. That is, the CSL is extensible. Once, more essential elements of an e-business contract are identified the CSL can easily be extended to incorporate these additional elements.

Chapter 4 detailed the design of a non-SOAP based negotiation component of O.C.E.A.N. Understanding the O.C.E.A.N negotiation component is important if we have to extend the functionality of the negotiation component in future versions of O.C.E.A.N.
The “Yes No” protocol for automated negotiations was detailed and its advantages and disadvantages were discussed.

Chapter 5 dealt with the architecture of the SOAP based negotiation component in O.C.E.A.N. Understanding this architecture is very important if O.C.E.A.N is planned to be implemented as a web service which existing grid architectures can make use of. The “Static Bargain” protocol of automatic negotiations was detailed and its advantages and disadvantages were discussed. Chapter 5 also explored the possibility of adding rule-based systems to the negotiation component of O.C.E.A.N to help in its decision-making capabilities.

Chapter 6 provided the negotiation API that relieves much of the hassles of generating contracts and communicating with the contracts. The API also provides means for generating rule documents and parsing rule documents.

Though the standards and protocols proposed are sufficient for the initial implementation of O.C.E.A.N there is no doubt that as the O.C.E.A.N system develops, more needs will be identified and the protocols and standards will have to be modified. All the protocols and standards have been developed with this in mind and there should be no problem for later versions of O.C.E.A.N to build on top of the existing system.

7.2 Future Work

The amount of future work that can be spawned from this thesis is mind-boggling. In the future as changes happen, more essential elements of contracts can be identified. The negotiation component of O.C.E.A.N has to be migrated completely to a web based service. Latest technologies like JXTA, JNDI etc can be used for peer discovery. More complex rule based systems can be developed to capture the personalities of humans in a more effective manner. The rule language can be extended to include complex
combinations of Boolean expressions. Research can be done on developing more complex negotiation protocols like the “Yes No” and “Static Bargain” protocols.

The aim of this thesis was not to develop a comprehensive negotiation system that would handle all cases. Instead, it was to provide tools for systems to develop comprehensive negotiation systems and provide the tools in such a way that they can be easily extended depending on the needs of the application.

In this regard we consider the thesis to be a success.
APPENDIX A
SCHEMAS

A1 Contract Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs: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:xs="http://www.w3.org/2001/XMLSchema"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >
<xs:element name="Envelope">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Header">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ContractID"/>
            <xs:element name="Application" type="xs:string"/>
            <xs:element name="Buyer" type="xs:string"/>
            <xs:element name="Seller" type="xs:string"/>
            <xs:element name="NegotiationProtocol" type="xs:string"/>
            <xs:element name="Status" type="xs:string"/>
            <xs:element name="Date" type="xs:dateTime"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Body">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="NegotiationItems">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Item" maxOccurs="unbounded">
                    <xs:complexType/>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="PaymentInformation"/>
    </xs:complexType>
  </xs:sequence>
</xs:element>
</xs:complexType>
```
<xs:sequence>
  <xs:element name="Mode">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="subscription"/>
        <xs:enumeration value="oneshot"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="Type">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="paypal"/>
        <xs:enumeration value="eGold"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ContractBreakAction">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="noAction"/>
      <xs:enumeration value="kickOut"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="Signature"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

A2 Rule Schema

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.cise.ufl.edu/research/O.C.E.A.N"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="Rules">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="If">
          <xs:complexType>
            <xs:sequence>
              <!-- Elements go here -->
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
LIST OF REFERENCES


[27] Sun Microsystems, Java API for XML messaging (JAXM),
BIOGRAPHICAL SKETCH

Chaitanya Chokkareddy was born in Tirupati, Andhra Pradesh, India, on June 29th, 1978. Chaitanya attended Neduramalli Bala Krishna Reddy Institute of Science and Technology, which is affiliated to Sri Venkateswara University and located in Vidyanagar, Andhra Pradesh, India, where he received a Bachelor of Science degree in Computer Science and Engineering in 2000.

In 2000, Chaitanya 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 XML and SOAP.