



Alexandria University
Faculty of Engineering
Computer Science and Automatic Control Department
2004-2005

An Agent-Based Bidding System

Implementing an Electric Power Market

Supervised by

Dr. Khaled Nagi

Presented by

Ahmed Ragab El-Habahsy

Shahira Abou Samra

Mohamed Abd El-Aziz

Osama Yehia

Ahmed Samy Hamed

Acknowledgement

First, we would like to thank Allah for helping us to complete this project successfully.

We would like to thank **Dr Khaled Nagi** for guiding us during the perpetration of this project. And guiding us in improving the performance of the system.

Last but not least; we acknowledge our families' support and back up.

Contents

1	Introduction	2
1.1	Overview	2
1.2	Key features of our system	2
1.3	Report Organization	4
2	Background	6
2.1	Online Auctions	6
2.2	Auctions for Electric Power	7
2.2.1	The Electric Power Industry	7
2.2.2	What's Different about Electric Power Auctions	7
2.3	Auction Types	8
2.3.1	Auction Classification according to User Roles	8
2.3.2	Auction Formats	8
2.4	Intelligent Software Agents	10
2.4.1	Agent life Cycle	10
2.4.2	Agents Communication	12
2.4.3	Agents Benefits	13
2.4.4	Agents Applications	16
2.4.5	Agent-Oriented versus Object-Oriented Models	18
3	Agency for Marketing Electric Power	19
3.1	Electric Power Market Operations	19
3.2	Agents in Our System	22
3.3	Agents Description	23
3.3.1	Auctioning Agent Description	23
3.3.2	Bidding Agent Description	26
3.3.3	TSO Agent Description	28
3.3.4	Monitor Agent Description	31
3.4	Agents Interaction Protocol	33
3.4.1	Agents Communication Language (ACL)	33
3.4.2	Agents Interaction Protocol Design	34

4	Agency Implementation	36
4.1	JADE Platform	36
4.2	Agents Platform for Our System	38
4.3	Agents Implementation	38
4.3.1	Seller Auctioning Agent Implementation	38
4.3.2	Buyer Auctioning Agent Implementation	44
4.3.3	Buyer Bidding Agent Implementation	46
4.3.4	Seller Bidding Agent Implementation	54
4.3.5	TSO Agent Implementation	56
4.3.6	Monitor agent implementation	63
4.4	Data Files	68
4.4.1	XML files	68
4.4.2	Log files	70
5	Running the Electric Power Market Agency	72
5.1	Prerequisites	72
5.2	Application scenario	72
5.2.1	Starting the agents	76
6	Conclusion and Future Work	95
6.1	Conclusion	95
6.2	Future Work	96
A	Java Agent DEvelopment Framework (JADE)	97
A.1	JADE Overview	97
A.2	JADE Programming	98
A.3	The Yellow Pages Service:	102
A.4	Why JADE	103
B	FIPA ACL Message Structure Specification	105
B.1	FIPA ACL Message Structure	105
B.2	Type of Communicative Act	107
B.2.1	Performative	107
B.3	Participants in Communication	107
B.3.1	Sender	107
B.3.2	Receiver	108
B.3.3	Reply To	110
B.4	Content of Message	110
B.4.1	Content	110
B.5	Description of Content	110
B.5.1	Language	110

B.5.2	Encoding	110
B.5.3	Ontology	111
B.6	Control of Conversation	111
B.6.1	Protocol	111
B.6.2	Conversation Identifier	112
C	The Simplex Method	113
C.1	Introduction	113
C.2	LP solution space in equation form	113
C.3	Converting Inequality into equations	113
C.4	Computational Details of the Simplex Algorithm	114
	Bibliography	120

List of Figures

1.1	Energy Market Multi-Agent System	4
2.1	Agent Thread path of execution	11
3.1	Auctioning Agent Flow Chart	25
3.2	Bidding Agent Flow Chart	27
3.3	TSO Agent Flow Chart	29
3.4	TSO Agent Flow Chart (cont.)	30
3.5	Monitor Agent Flow Chart	32
3.6	Agents Interaction Protocol	35
4.1	Containers and Platforms	37
4.2	UML class diagram for seller auctioning agent	39
4.3	UML class diagram for buyer auctioning agent	45
4.4	UML class diagram for buyer bidding agent	47
4.5	UML class diagram for seller bidding agent	55
4.6	UML class diagram for TSO agent	57
4.7	UML class diagram for monitor agent	64
5.1	Companies Graph	73
A.1	Role of the middleware: "vertical" approach (left) vs. "horizontal" approach (right)	98
A.2	The JADE Architecture	99
A.3	Agent Thread path of execution	101
A.4	The Yellow Pages service	102

List of Tables

2.1	Classification of KQML Performatives	13
2.2	Taxonomy of FIPA Performatives	14
2.3	Equivalent Operations in FIPA-ACL and KQML	14
B.1	FIPA ACL Message Parameters	106
B.2	FIPA ACL Message Frame and Ontology	107
B.3	FIPA ACL Message Performative	107
B.4	FIPA ACL Message Sender	108
B.5	FIPA ACL Message Receiver	108
B.6	FIPA ACL Message Reply-To	109
B.7	FIPA ACL Message content	109
B.8	FIPA ACL Message Language	110
B.9	FIPA ACL Message Encoding	110
B.10	FIPA ACL Message Ontology	111
B.11	FIPA ACL Message Protocol	111
B.12	FIPA ACL Message Conversation Identifier	112

Chapter 1

Introduction

1.1 Overview

In this project, we deal with a real world problem; that of buying and selling of electric power among distributed customers, resulting in a maximum reasonable profit with the least possible effort. This is achieved by building a Multi Agent System (MAS), that implements Dutch auction format, such that each auction is composed of a number of auctioning and bidding agents along with a single transmission system operator agent that is used by the whole system.

We implement the MAS using the open source library JADE (Java Agent DEvelopment Framework) through which we build agents that live on distributed platforms. The agents in the system communicate using FIPA-ACL (Foundation for Intelligent Physical Agents-Agent Communication Language) which is the most widely used and supported ACL. XML (eXtensible Markup Language) files are used to maintain customers' accounts and various logging files.

In the rest of this report, we discuss all aspects of our system in detail, showing results and conclusion at the end.

1.2 Key features of our system

Our system, Agent-based Electric Power Auction, provides the following features:

1. Bidders can participate in one or more auctions to fulfill their needs at the best possible price.
2. Auctioning parties negotiate with the bidders to maximize their benefits.
3. It monitors supply and demand.
4. The system provides means of checking the status of generation and transmission equipment for transmission feasibility and cost.
5. It gets the minimum-cost path to any other agent's company location.
6. It keeps a grid map of all nodes capable of propagating electric power, associated with the cost of the lines between nodes.
7. The presentation layer is separated from the logic such that agents' behaviors can be easily extended to provide the intelligence requested by the user.
8. Different bidding strategies are provided in an attempt to meet various business needs.
9. It ensures that a user trying to start an auction or make a bid is legitimate; through a known user name and password for that user.
10. Checks user's deposit before starting in auctions.
11. It provides financial penalties for agents that reject a transaction after making a proposal.
12. Provides simple graphical user interfaces that enable customers to use the system smoothly and easily.

The multi-agent system is characterized as shown in figure 1.1. Buyer agents and Seller agents are engaged in two-sided contracts for energy supply. Each buyer or seller is either a bidding agent or an auctioning agent. TSO (Transmission System Operator) monitors the physical transmission assets and gets the best transmission route.

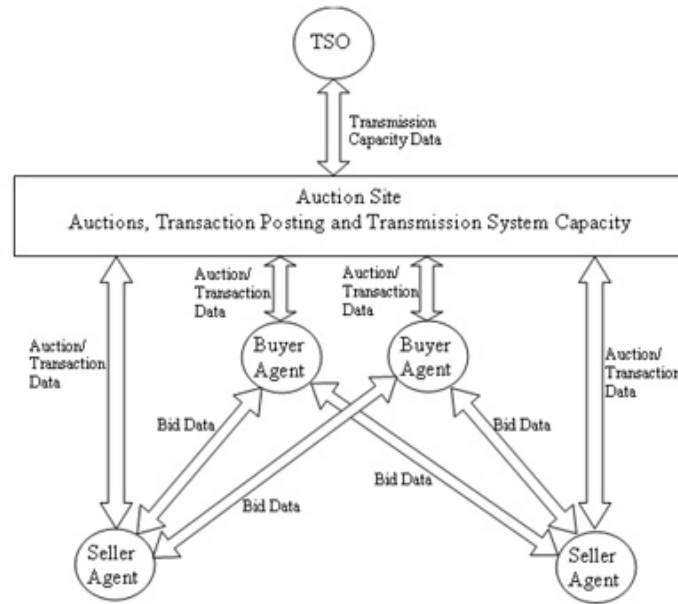


Figure 1.1: Energy Market Multi-Agent System

1.3 Report Organization

Chapter 1: Introduction - This chapter gives an overview of the project and tools used, and shows the key features of our system.

Chapter 2: Background - This chapter provides a background on on-line auctions, the electric power industry, auction types and classifications and finally a background on intelligent software agents.

Chapter 3: Agency for Marketing Electric Power - This chapter is about the system design. It starts off by discussing the electric power market operations. It then shows the types of agents needed in the system and describes how they should operate. Finally the choice of the ACL language is discussed and the design of the agents' interaction protocol is shown.

Chapter 4: Agency Implementation - introduces the JADE platform where agents can live. The chapter discusses the client/server model. It also investigates agents' behaviors and the interaction between the various agents through their life cycle. Finally, the chapter discusses the structure of the log files where auction and bidding data are maintained.

Chapter 5: Running the Electric Power Market Agency - gives the details of system deployment and discusses a running example showing the system interface the agent's communications and actions.

Chapter 6: Conclusion and Future Work - summarizes the conclusion of our work, and discusses how the system can be expanded and improved in the future.

Chapter 2

Background

2.1 Online Auctions

One of the best ways to allocate goods and resources is to sell them using free market techniques. An auction is an excellent method of distributing goods to those who value them most highly.

Several attempts have been made to develop generalized online auctions. This has been achieved to different degrees of success. Kumar and Feldman developed a system at IBM that covered both English and Dutch Auctions. Their conclusions were that online auctions are a very viable means to transacting the sale of items, however significant infrastructure is needed to properly administer the auctions.

Currently several commercial offerings are available, claiming ease of installation and use. Among those products are: Auctioneer, Beyond Solutions, C-U-S Business Solutions, Emaze Auction, Merkatum, IbidU.com, Auction-USA, and Auction Broke Software. All of these products are limited in the types of auction that can be run, particularly when it comes to specifying the deployment system and changing aspects of the auction rules. Out of the box solutions tend not to have full configuration support and secure administration without significant infrastructure added.

2.2 Auctions for Electric Power

2.2.1 The Electric Power Industry

There exist many participants that are cooperating in the electric power industry. Those participants are mainly: the producers, the consumers, and the transmission companies. The producers are the suppliers that feed the market with the electric power. The consumers are the customers of the electric power. The transmission companies are the responsible of transmitting the electric power between the producers and the consumers.

The electric power industry had faced a great evolution in the last few decades. The legacy systems are based on a centralized model where all information flows into a central site where decisions are made. Currently, the electric power industry is becoming a large interconnected network of distributed resources.

The electric power industry is a very active industry that is involving many sensitive operations. These operations could be summarized as follows: Buying and selling energy, monitoring supply and demand, system monitoring and maintenance, etc.

The electric power industry is a critical industry that is facing many difficulties, buying and selling in this environment is made difficult because of the difficulty of obtaining information in reasonable time and making decisions using this information.

2.2.2 What's Different about Electric Power Auctions

Electric power auctions are different and more complex than other auctions. Typically, auctioning goods requires only determining the amount of available goods and the time of delivery. Auctioning electric power requires determining the amount of electric power and the duration of delivery. Several consumers would request electric power for different durations and all within the generator's period of delivery.

For example, a generator offers 50 watt/hr for 12 months; from January 2005 till December 2005, and there are three consumers. Consumer A requests 40 watt/hr for 4 months; from January 2005 to April 2005. Consumer B requests 20 watt/hr for 6 months; from January 2005 to June 2005.

Consumer C requests 30 watt/hr for 3 months; from September 2005 to November 2005. As we see in the previous example, different consumers request different electric power and for different periods. The duration for consumer A and consumer B are overlapping and the sum of the requested electric power by both of them is greater than the generator can provide resulting in a collision. Consumer A requests higher power than consumer B, but consumer B's request is for a larger period. It is the generators responsibility to resolve this collision and to determine the best combination of consumers that would result in the highest profit thus adding more complexity to the auctioning problem.

2.3 Auction Types

2.3.1 Auction Classification according to User Roles

Auctions can be grouped by the roles users play:

- Consumer to Consumer, C2C, describes the model where end users place items for auction on a large site where other consumers can register and make bids. These systems are usually set up in custom environments where a large range of items can be accommodated. eBay and Yahoo have been very successful establishing a business around this model.
- Business to Consumer, B2C, describes an auction market where companies setup auctions to sell off excess inventory and product.
- Business to Business, B2B, where companies use the auction to source supplies, parts and raw materials. The generation of an RFP is generally part of the auction process. In the case of multiple suppliers a Reverse Auction may be used that is an auction where consumers open the auctions for the suppliers to bid in.

2.3.2 Auction Formats

Auctions come in many forms. The most popular of which are the English Auction and the Dutch Auction formats.

English auction format

In the English auction, the auctioneer seeks to find the market price of a good by initially proposing a price below that of the supposed market value and then gradually raising the price.

Each time the price is announced, the auctioneer waits to see if any buyers will indicate their readiness to pay the proposed price. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with an incremented price.

The auction continues until no buyers are prepared to pay the proposed price, at which point the auction ends and the highest bidder wins the auction.

Dutch auction format

The term "Dutch Auction" derives from the flower markets in Holland, where this is the dominant means of determining the market value of quantities of cut flowers.

In the Dutch auction format, the auctioneer attempts to find the market price for a good by starting the at a price much higher than the expected market value, then gradually reducing the price until one of the buyers accepts the price.

The rate of reduction of the price is up to the auctioneer. There is usually a reserved price below which not to go. If the auctioneer reduces the price to the reserved price with no bidders, then the auction terminates.

While modeling the actual Dutch flower auction (and definitely in other markets), some additional complexities occur:

- First, the good may be split: for example the auctioneer may be selling five boxes of tulips at price X , and a buyer may purchase only three of the boxes. The auction then continues, with a price at the next increment below X , until the rest of the good is sold or the reserve price met. Such partial sales of goods are only present in some markets; in others the purchaser must bid to buy the entire good.
- Secondly, the flower market mechanism is set up to ensure that there is no contention amongst buyers by preventing any other bids once a

single bid has been made for a good. Offers and bids are binding, so there is no protocol for accepting or rejecting a bid. In the agent case, it is not possible to assume, and too restrictive to require, that such conditions apply. Thus it is quite possible that two or more bids are received by the auctioneer for the same good.

2.4 Intelligent Software Agents

Intelligent software agents are a new class of software that act on behalf of the user to find and filter information, negotiate for services, easily automate complex tasks, or collaborate with other software agents to solve complex problems.

2.4.1 Agent life Cycle

Each software agent has a life cycle through which it executes the task it is intended for. The agent life cycle is shown in figure2.1.

The agent life cycle can be divided into three main processes:

- Agent Setup, this is performed as soon as an agent starts and is intended to include agent initializations. Typically, but not necessarily, service registration (publication) is done also within the agent setup.
- Executing the behaviors that represent the tasks the agent is intended for. This can be summarized by the following steps:
 - Check whether the agent has been killed and have to perform the clean-up operations, or the agent is still alive and have to get the next behavior from the pool of active behaviors,
 - Perform the agent action that actually defines the operations to be performed when the behavior is in execution, and
 - Specify whether or not a behavior has completed and have to be removed from the pool of behaviors an agent is carrying out.
- Agent Take Down, this is performed just before an agent terminates and is intended to include the agent clean-up operations, also deregistering from the yellow pages is done here.

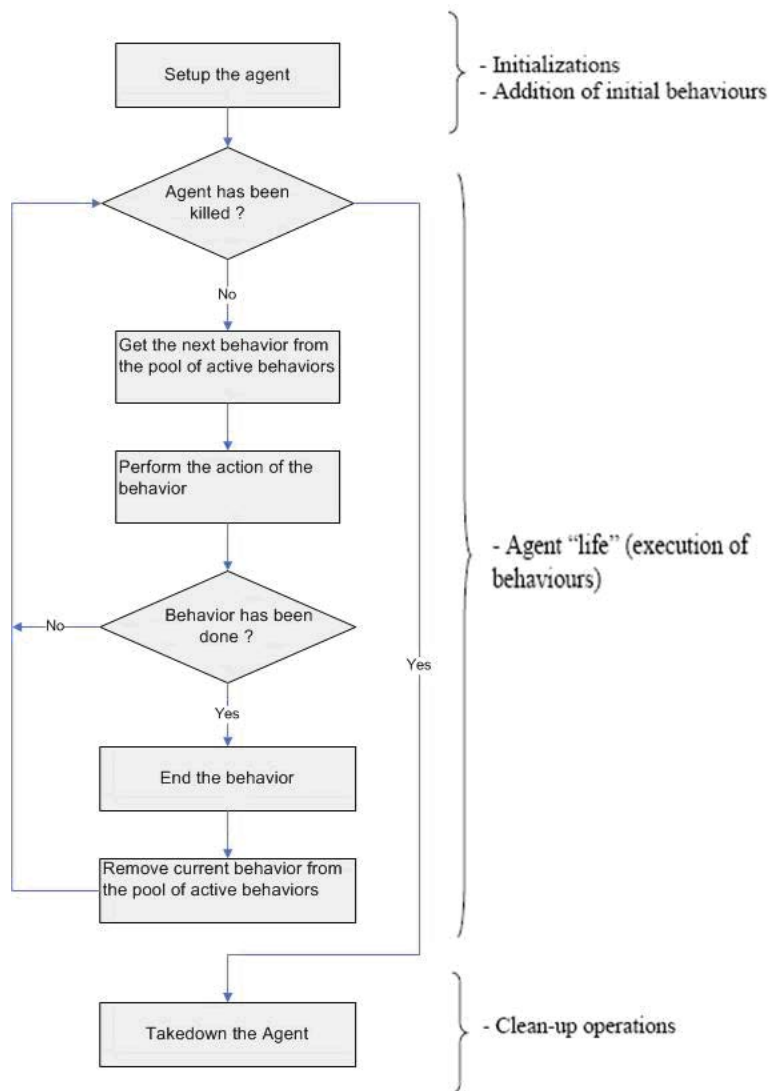


Figure 2.1: Agent Thread path of execution

2.4.2 Agents Communication

One of the most important advantageous features of agents is their ability to communicate which allows them to cooperate to solve more complex problems in much simpler ways. All the agents that implement that a common language will be mutually intelligible.

Such a common language needs an unambiguous syntax, so that agents can all parse sentences the same way. It should have a well-defined semantics or meaning, so all agents can understand sentences the same way. It should be well known, so that different designers can implement it and agents would have a chance of encountering other agents that know the same language. It should have the expressive power to communicate the kinds of things that agents may need to say to one another.

An ACL (Agent Communication Language) provides language primitives that implement the agent communication model. In agent representation models, there are two terms: ontology and content language. Ontology, defines the domain model / vocabulary etc. of a particular domain of discourse; it is a particular conceptualization of a set of objects, concepts, and other entities about which knowledge is expressed, and of the relationships among them. Content language represents the agent's mental model of the world (e.g. beliefs, desires, intentions); it expresses facts about the domain. ACLs are commonly thought of as wrapper languages in that they implement a knowledge-level communication protocol that is unaware of the choice of content language and ontology specification mechanism.

The most popular ACL languages are: Knowledge Query and Manipulation Language (KQML) and the Foundation for Intelligent and Physical Agents (FIPA).

Knowledge Query and Manipulation Language (KQML)

KQML is a message format as well as a message handling protocol. KQML consists of three layers; the content layer, the message layer, and the communication layer. The content layer consists of the content of the message. The communication layer handles the lower level communication parameters. The message layer consists of a performative, which conveys information about the content of the message, as well as optional information to describe the content language.

Function Class	Member Performatives	Level
Query and Response	ask-if,ask-all,ask-about, ask-one, tell,untell,deny, sorry	agent-pair
Cursor Manipulation and Result Formatting	ready, next, discard, rest stream-all, stream-about, eos	agent-pair
advertise or commit to a capability	advertise, unadvertise	agent community
kb editing	insert,uninsert,delete-one,delete-all,undelete	agent-pair
enactment	achieve, unachieve	agent-pair
error handling	error	agent-pair
communication primitives other than pure asynchronous messages	broadcast, forward, standby, subscribe and monitor (like a kb alerter), pipe, break (make and dismantle a pipe), generator	either
trading	broker-one, broker-all, recommend-one, recommend-all, recruit-one, recruit-all	agent-community
name service	register, unregister, transport-address	agent-community

Table 2.1: Classification of KQML Performatives

Foundation for Intelligent and Physical Agents (FIPA)

FIPA-ACL is another ACL with a format similar to that of KQML. ACL messages use a set of parameters, the first representing the communicative act (equivalent to performative in KQML) of the message. The additional parameters specify items: to aid in message delivery, to assist in the interpretation of the content, and to allow the receiver to respond co-operatively.

2.4.3 Agents Benefits

The key characteristics of multi-agent technology provide several benefits over traditional large, hierarchical programs. While some systems may share some of these benefits they are all native to the nature of multi-agent technology.

Function Class	Member Performatives
query and response	<ul style="list-style-type: none"> • query-if, query-ref • request, request-when, request-whensoever • inform, inform-if, inform-ref • confirm, disconfirm
communication primitives other than pure asynchronous messages	cfp, propose, reject-proposal, accept-proposal, subscribe (like a kb alerter)
error handling	failure, not-understood
?	agree, cancel

Table 2.2: Taxonomy of FIPA Performatives

KQML Performative	Equivalent FIPA Performative
ask-if	query-if
tell, untell	inform
deny	inform or disconfirm
insert, uninsert	inform, disconfirm
subscribe	subscribe
error	not-understood
sorry	refuse or failure

Table 2.3: Equivalent Operations in FIPA-ACL and KQML

Autonomy

Each agent within the system is designed to operate autonomously. As such, multi-agent systems do not wait for instructions and are capable of reacting to any change in a situation without being prompted. While this makes multi-agent software essentially flexible, this is within the parameters set by the operator with the agents' domain-specific knowledge refreshed from the ontology without any need for the system to be shut down. This means that multi-agent systems are capable of working 24/7 with knock-on benefits in terms of saving time, money and effort.

Intelligence

Particularly within the supply chain, resource limitations make finding an optimal solution difficult if not seemingly impossible. Due to the fact that multi-agent technology consists of a network of small agents, the way in which the system approaches a resource problem or conflict is both unique and dynamic. The intelligent software agents can negotiate with other agents representing resources and demands. In addition, the agents compete and cooperate to make a decision that is best for all participants of the network, under the current dynamic circumstances. This means that should a conflict arise, the agents will attempt to work around it and provide a best-fit solution that works. This level of responsiveness is not possible within other sequential systems which, when faced with conflict, would be unable to provide a solution or more critically, could crash.

Pro-Activity

Software agents interact with the system users proactively, representing available resources and demands. Whenever parts of the schedule become infeasible, the users can make certain decisions without interrupting the system.

High Performance

High performance and scalability is an extremely important requirement. Multi-agent software meets this need by using hundreds of thousands of software agents running concurrently on a single server, exchanging up to 10,000 messages in a second. A typical system would be able to provide for web applications with one to two seconds of reaction time for ten to twelve users

in parallel. This allows the system to work and react in real-time.

Low Cost Maintenance

A large proportion of software maintenance costs are associated with the software being hard coded within the system. This means that any changes require expert technical knowledge in the required programming language and are extremely time-consuming. Within a specific multi-agent technology, the domain-specific knowledge of the system, stored in the ontology, is separated from the system code. This makes the system easy to modify by users without any programming skills, and means that the ontology can be changed on the fly, without any interruption to the system. Furthermore, the software provides the added benefit of enabling collaboration across business process and integration across enterprise systems.

High Level of Customization

For an organization to remain competitive it must keep its individuality. Therefore it is essential that any software can be customized to meet adapted requirements. Multi-agent technology does this by using an ontology and software agent structure which allows individual goals, strategies, preferences, restrictions and other parameters to be uniquely defined. It also allows the systems to take into account every user, resource or any other element of the system, which makes it possible to build a system fully customized for the client's business network.

2.4.4 Agents Applications

Agents can be used to implement systems without centralized control, systems decomposable into independent tasks and systems composed of independent autonomous entities. Some of the applications where multi agent systems can be utilized are discussed below:

A Bidding System:

There exist specific agents to achieve:

- The role of the auctioneer.
- The role of the bidder.

A Network Management System:

There exist specific agents to achieve:

- Dynamic bandwidth management.
- Fault restoration.
- Spare capacity planning.

A Teleteaching Project:

There exist specific agents to achieve:

- Companion agent: assists a student by providing a graphical user interface that will help interact with other agents and mostly with the tutor agent.
- Tutor agent: provide assignments.
- Verifier agent: analyze the validity and coherence of a proof given by a student.

A Hotel Service Broker System:

An agent accessing the system services can:

- Search for hotels satisfying some given conditions.
- Retrieve information about these hotels.
- Finally book these hotels.

A Travel Assistant:

A multi-agents system that delivers journey timetables.

e-Banking Service provider:

An electronic banking transactions MAS.

2.4.5 Agent-Oriented versus Object-Oriented Models

- Agent-Oriented models are much more suitable for distributed systems than Object-Oriented models.
- Behaviors in an agent are not pre-emptive (as for Java threads) but cooperative. This means that when a behavior is scheduled for execution its `action ()` method is called and runs until it returns.
- Therefore it is the programmer who defines when an agent switches from the execution of a behavior to the execution of the next one. Though requiring a small additional effort to programmers, this approach has several advantages:
 - Allows having a single Java thread per agent (that is quite important especially in environments with limited resources such as cell phones).
 - Provides better performances since behavior switch is extremely faster than Java thread switch.
 - Eliminates all synchronization issues between concurrent behaviors accessing the same resources (this speed-up performances too) since all behaviors are executed by the same Java thread.
 - When a behavior switch occurs the status of an agent does not include any stack information and is therefore possible to take a "snapshot" of it. This makes it possible to implement important advanced features e.g. saving the status of an agent on a persistent storage for later resumption (agent persistency) or transferring it to another container for remote execution (agent mobility).

Chapter 3

Agency for Marketing Electric Power

3.1 Electric Power Market Operations

The energy market will consist of buyers and suppliers interacting directly to establish two-sided contracts for the sale and delivery of energy. Information is provided to all participants via a public forum. The market will operate continuously under a process having three basic stages:

- Auction
- Transaction Definition
- Transaction Execution

Stage 1: Auction

Buyers and suppliers are matched using a modified Dutch auction format. In this format both buyers and suppliers may conduct an auction session by posting buy/sell information in a public forum.

Buyers and sellers conduct individual Dutch auction sessions without the assistance of an auctioneer. The process begins when the auctioning party, buyer or seller, registers himself by posting the auction information: company name, auction capacity, price per watt, minimum-contract amount, start and end delivery time. The auctioning party is not responsible for calculating transmission cost since it depends on the physical location of the bidders, which is not known to the auctioning party at the time of posting.

For simplicity, we assume that all auctions will start at the same time and auction round time will be an hour from start time.

In accordance with the Dutch auction format, the price of the next round time is modified using a known price modifier. However, the auctioning agent may choose to discontinue the auction session if the next round price falls below or rises above the reserved price established by, and known only to, the buyer or seller auctioning party respectively. The difference between buyer and seller auctioning parties is that the price modifier is negative for seller and positive for buyer.

Under this auction format the auctioning party is compelled to accept bids and complete transactions before the next round or risk getting a lower price. The bidder, on the other hand, is under pressure to submit a bid before the capacity is purchased, or supplied, by another bidder or before the session is discontinued and the capacity is possibly withdrawn from the market altogether. Waiting for the next posting can result in a "better deal" for the bidder but at the risk of losing the transaction altogether, if the auction session is discontinued before a bid can be submitted.

The Dutch auction format has been chosen to support bidding strategies that result in equitable product valuations. Bidding agents have less fear of winner's curse, over-paying, since the bidding process does not involve a public competitive bidding process such as in the English Outcry, or Open Cry, auction. No information is available on bids from other prospective bidders prior to bidding so bids tend to approach the true valuation of the product, e.g., bids are not raised by huge or misinformed bidding.

Stage 2: Transaction Definition

Auctions and bidders perform registration. Each auctioning or bidding party is either an electric power generator or an electric power consumer (i.e. is either a buyer or a seller). A generator (seller) may satisfy the needs of different consumers each for his specified duration but within the generators announced minimum and maximum delivery dates, while for a consumer (buyer), different generators may satisfy parts of consumers needed capacity but each for the whole duration announced by the consumer.

Authorized bidders can access the public information, evaluate information on the various auction sessions relative to their needs and check transmis-

sion feasibility and cost. If a decision is made a bid then bids are submitted directly to the auctioning party.

The auctioning party waits for a known amount of time for bidders to submit their bids. Each bid is validated, recorded and time stamped as it is received. Invalid bids (e.g. bids that are below minimum-contract amount or are for the wrong auction) are rejected and their bidders are notified. When the time of bidding has expired, the auctioning party checks if the total of the bids exceeds the posted load/capacity for any time slot. If so, then a bid collision condition exists that the auctioning party must eliminate prior to proceeding. Otherwise, the auctioning party send accept proposal to the bidders and reevaluate its needs.

In a traditional Dutch auction, when a collision occurs the auctioneer announces that a collision has occurred and the bids are withdrawn. The auctioneer then increases the price and allows bidding to resume. This process is repeated until the collision condition no longer exists or until an impasse is reached (repeated collisions) and the lot is withdrawn. In the absence of an independent auctioneer, this response to bidder collisions is not practical. Without oversight, auctioning parties could falsely announce a collision as a means of forcing higher bids. In addition, repeated collisions can consume too much time and resources.

For these reasons, an alternate solution is proposed that distribute the load/capacity at each time slot amongst the bidders. Because the price is constant for all bidders and the auctioning party wants to maximize its profit, the auctioning party can take the objective function as: For these reasons, an alternate solution is proposed that distribute the load/capacity at each time slot amongst the bidders. Because the price is constant for all bidders and the auctioning party wants to maximize its profit, the auctioning party can take the objective function as:

$$\text{Maximize } Z = \sum_n^1 B_i * N_i$$

Where:

n = number of bids received.

B_i = Bid for bidder i .

N_i = Time slots count required by bidder i .

Note:

For buyer auctioning party, N_i is constant for all bidders and equal to the auctioning party's duration.

For a seller auctioning party, N_i differs from one bidder to the other.

And the constraints can be:

For each time slot: $\sum \text{Bids Received} \prec \text{auction capacity}$

The auctioning party attempts to maximize the objective function by modifying the bid amounts without violating any of the constraints can use the Simplex algorithm to solve these linear equations. Some bidders modified bids may fall below the minimum contract amount then the auctioning agent can exclude those bidders, send modified bids to the bidders and wait their replies. If the bidder replies with approval then, add that bidder to accepted bidders. Otherwise, the capacity returned by the rejected bidder is put back into the capacity/demand. (If the capacity/demand can satisfy one or more from the excluded bidders add those bidders to the accepted bidders.)

Stage 3: Transaction Execution

The auctioning party informs the monitor party with the winner bidders and forfeited bidders, the monitor party takes 10% from the winner bidders' company deposit and adds the taken amount to the auctioning party's company. If the company deposit falls below a minimum amount, the monitor party prevents this company from proceeding till it updates its deposit

3.2 Agents in Our System

We have four types of agents in our system; Auctioning agent, Bidding agent, Transmission System Operator agent (TSO) and Monitor agent.

Auctioning agent: This represents a company that is either a buyer or a seller. It initiates an auction to meet its needs.

Bidding agent: This also represents a company that is either a buyer or a seller. It tries to satisfy its needs by reviewing and subsequently bidding into the various auctions according to user constraints and bidding strategy.

TSO agent: It monitors the physical transmission assets and gets the best transmission route. This is achieved by using a grid map that contains

the locations of the participants and the transmission routes and costs between them.

Monitor agent: This is the agent at which all other agents must register in order to find each other. It simulates the actual buying or selling and forfeiting a portion of a bidder's deposit in case of it refuses to complete a transaction.

3.3 Agents Description

3.3.1 Auctioning Agent Description

At start, auctioning agent registers with the monitor agent by posting its company name, company password, auction capacity, price per watt, minimum-contract amount, start and end delivery time. The auctioning agent is not responsible for calculating transmission cost since it depends on the physical location of the bidders, which is not known to the auctioning agent at the time of posting. The auctioning agent waits registration confirmation from the monitor; the confirmation includes auction start and round time.

By start time, the auctioning agent waits for bidders to submit their bids. Each bid is validated, recorded and time stamped as it is received. Invalid bids (e.g. bids that are below minimum-contract amount or are for the wrong auction) are rejected and their bidders are notified. When the time of bidding has expired, the auctioning agent checks if the total of the bids exceeds the posted load/capacity for any time slot. If so, then a bid collision condition exists that the auctioning agent must eliminate prior to proceeding. Otherwise, the auctioning agent sends accept proposal to the bidders and reevaluate its needs.

The auctioning agent can attempt to eliminate the bid collision problem by:

- Select bidders that are the most advantages to the auctioning agent (i.e., a single large bid for a small time slots versus many small bids for a large time slots).
- Distribute the load/capacity at each time slot amongst the bidders.

Once the bid collisions are resolved the auctioning agent transmits the modified bids to the bidding agents for confirmation. Capacity returned by rejected bidders is put back into the capacity/demand and distributed among the excluded bidders.

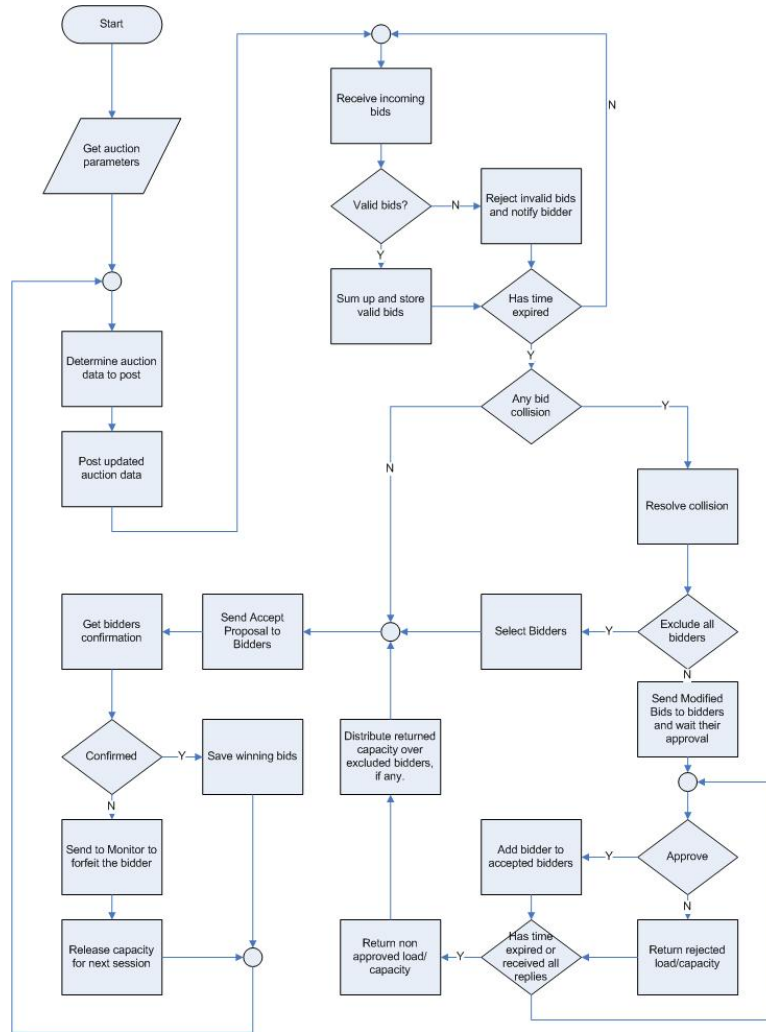


Figure 3.1: Auctioning Agent Flow Chart

3.3.2 Bidding Agent Description

The bidding agent represents a company that is willing to satisfy its needs (buy/sell electric power) by subsequently bidding in existing auctions. The bidding agent reviews the open auctions (represented by auctioning agents) and if any satisfy its defined constraints, it checks the transmission feasibility and transmission cost. If those still satisfy the agent's constraints, then it can bid in that auction so the auction is added to the agent's list of available auctions. From that list the bidding agent select the auctions to bid in and executes the bidding.

The bidding execution proceeds as follows: The bidding agent requests reservation of a transmission line from its company to the auctioning agent's company with the specified electric power capacity. If the reservation is granted, the bidding agent sends a proposal to the auctioning agent and awaits acceptance or rejection. The auctioning agent may also respond with a proposal of a reduced quantity (electric power capacity) upon which the bidding agent replies according to its strategy (e.g. according to it's defined minimum contract-amount).

For a given auction session, after executing all bids and transactions the bidding agent reevaluates its needs, adjusts its constraints and then blocks till the next session begin and the whole process is repeated until the bidding agent satisfies all its needs.

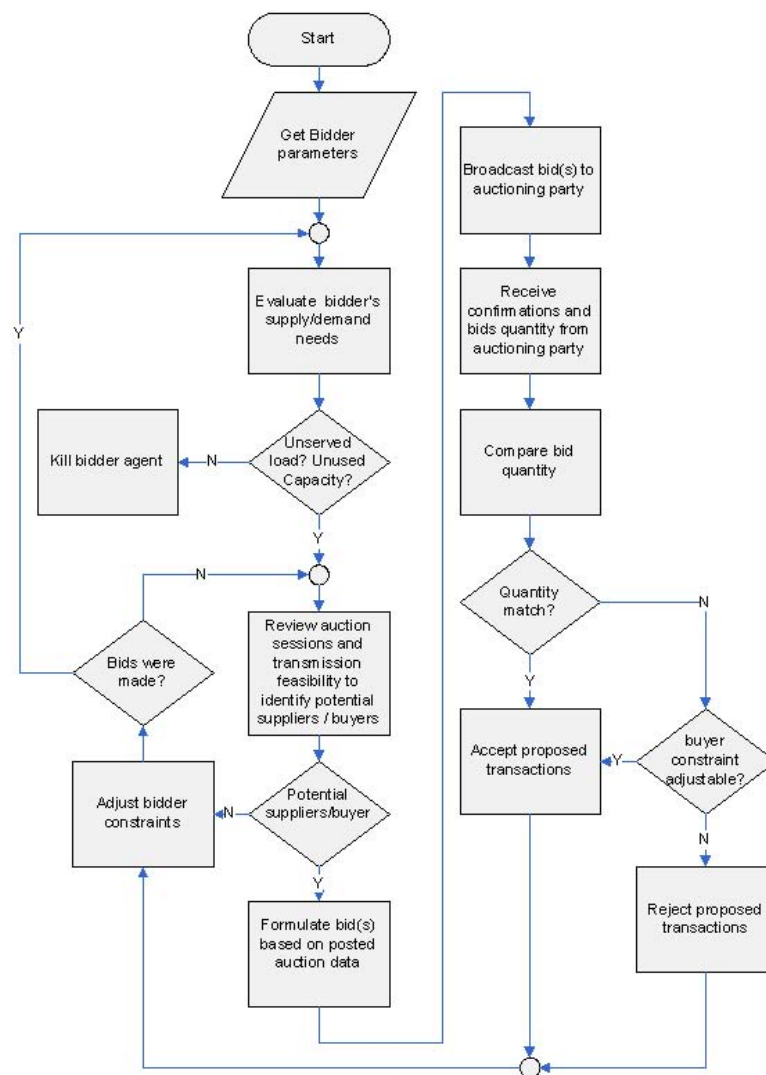


Figure 3.2: Bidding Agent Flow Chart

3.3.3 TSO Agent Description

The TSO agent monitors the physical transmission assets and gets the best transmission route. This is achieved by using a grid map that contains the locations of the participants, the transmission routes and the costs (per unit flow) between them.

The grid map owned by the TSO is a directed graph of the participants companies that is represented as an adjacency list, in which each company has a list of its adjacency companies. Each adjacent company is saving the data of the line between it and the source company.

The data for transmission line between a company and one of its adjacent companies includes: cost (per unit flow), capacity and reserved flow in various time slots. Each time slot is one month in period.

The TSO agent receives information from the monitor agent including adding or dropping a company or line. Using this information the TSO agent updates the grid map accordingly.

The TSO agent receives a request from the bidding agent asking for all the feasible paths between its company and other auctioning agents' companies during a specific time interval. The TSO agent returns the non-intersecting paths and for those paths that are intersecting, it returns the one with the highest ratio: path availability/path cost. That is using a heuristic that agrees with the bidding agents willingness to find a low cost path with a high available electric power flow, which increases the bidding agent's probability of finding the path still available when requesting reservation.

The TSO agent receives a request from the bidding agent asking for reserving the best path(s) between its company and a specific auctioning agent's company given a flow, a specific time interval, minimum acceptable flow and a maximum acceptable transmission cost (per unit flow). The TSO agent replies with refusal if there is no set of paths that can satisfy either the minimum acceptable flow or the maximum transmission cost (per unit flow), otherwise it reserves a path with such specifications.

The TSO agent receives a confirmation from the auctioning agent in order to confirm the flow reservation between its company and a specific bidding agent's company by sending the difference between the previously reserved flow and the actual consumed one during a specific time interval. The TSO agent frees the amounts given in the auctioning agents message.

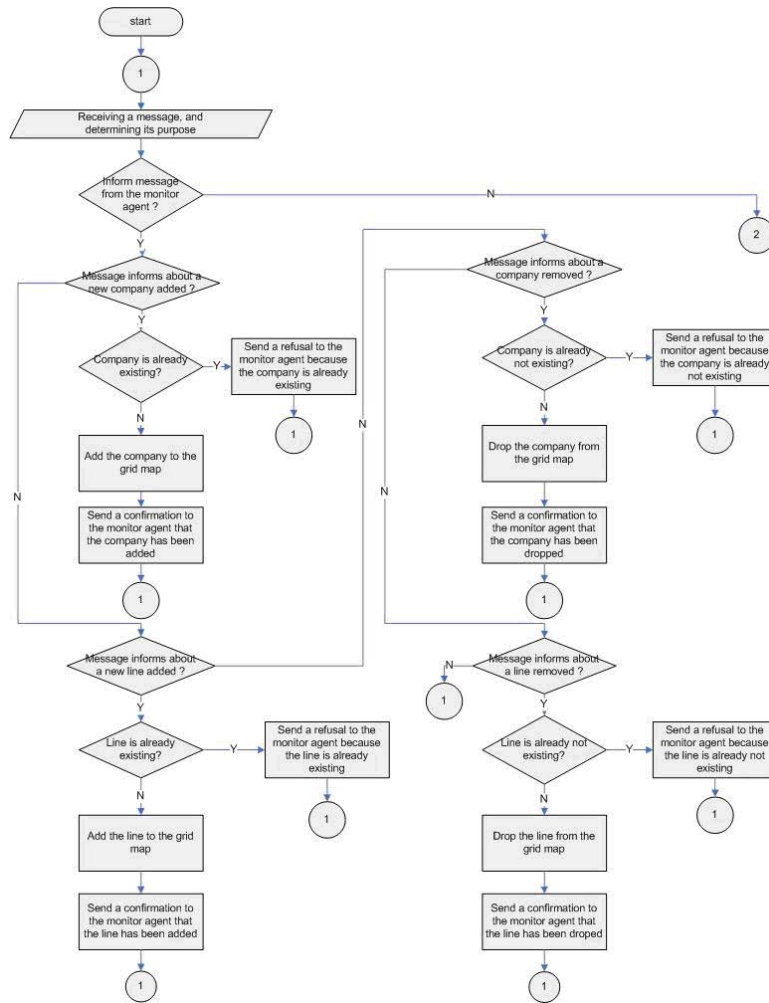


Figure 3.3: TSO Agent Flow Chart

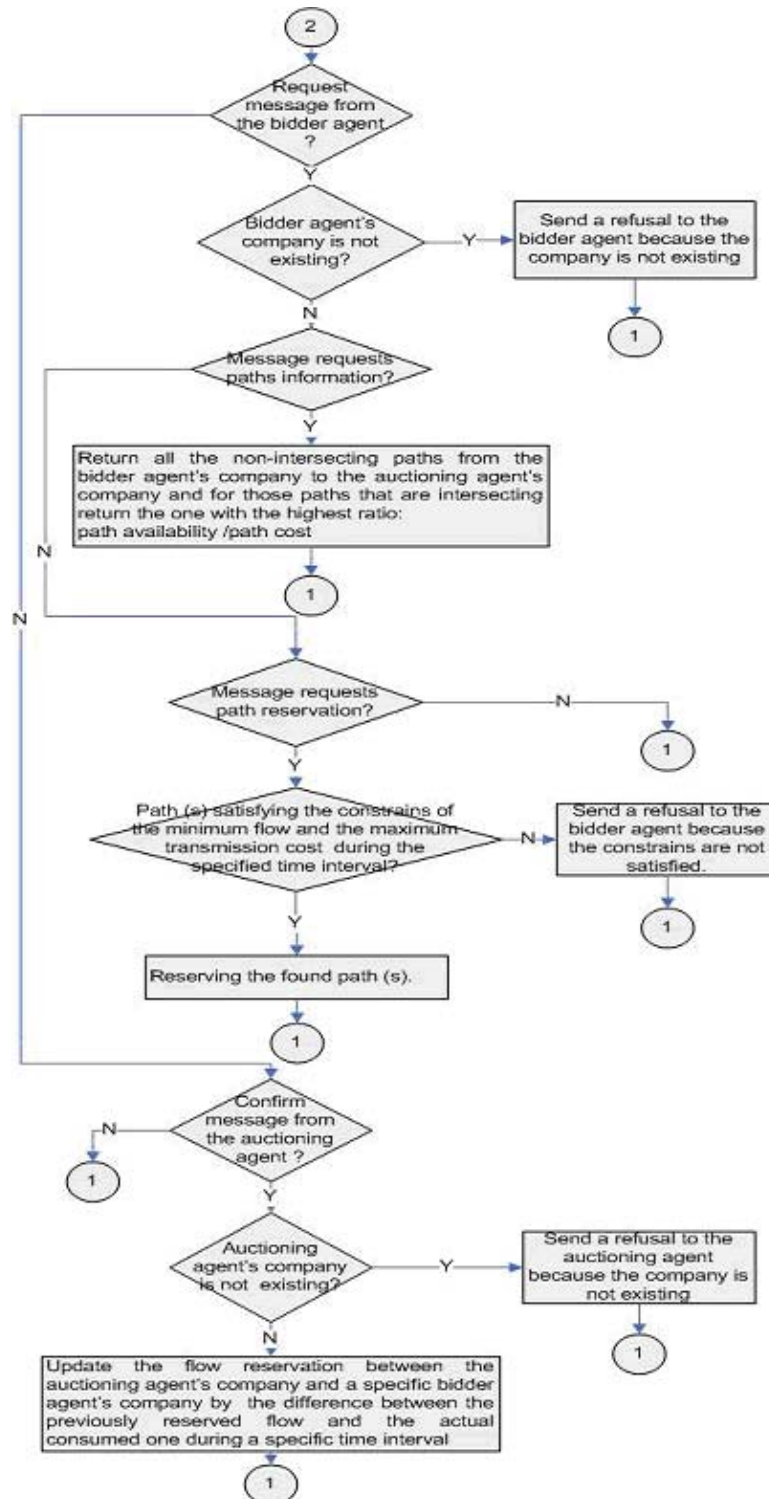


Figure 3.4: TSO Agent Flow Chart (cont.)

3.3.4 Monitor Agent Description

Monitor agent is the agent at which all other agents must register in order to find each other. It is responsible to inform the TSO agent about changes in the electric power market such as, adding and dropping transmission lines and if a new company was established or a company was dropped. The monitor agent also simulate the actual buying, selling and forfeiting a portion of the bidder's deposit in the case of it refuses to complete the transactions.

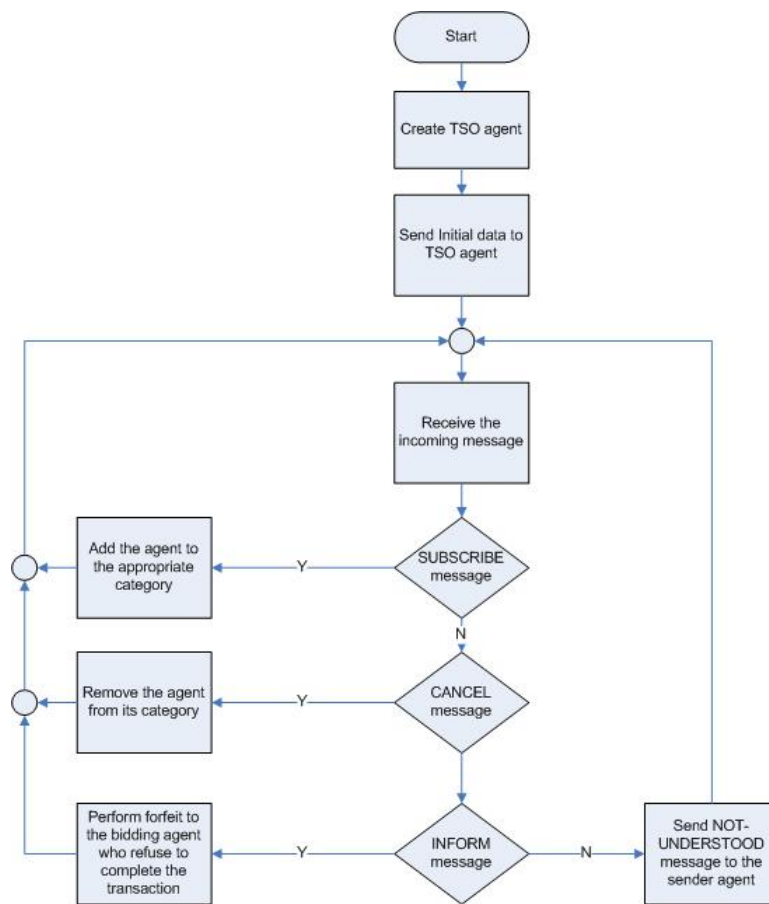


Figure 3.5: Monitor Agent Flow Chart

3.4 Agents Interaction Protocol

Agents Interaction Protocol indicates the sequence and order by which messages and their replies are exchanged by agents in the system.

3.4.1 Agents Communication Language (ACL)

As discussed in Chapter 2, Agents communicate using standard Agents Communication Language. The most widely used ACLs are: Knowledge Query and Manipulation Language (KQML) and the Foundation for Intelligent and Physical Agents (FIPA). In our project we used FIPA- ACL.

Why FIPA-ACL

Even though FIPA-ACL, like KQML, is designed to work with any content language and any ontology specification approach, and beyond the commonality of goals and the similarity in syntax, there are a number reasons why FIPA-ACL is preferred over KQML:

- In FIPA-ACL's semantic model, agents are not allowed to directly manipulate another agent's virtual KB. Therefore KQML performatives such as insert,uninsert,delete-one, delete-all, undelete are not meaningful.
- The FIPA-ACL limits itself to primitives that are used in communication between agent pairs. The FIPA architecture has an AMS (Agent Management System) specification that specifies services that manage agent communities. This eliminates the need for register/unregister, recommend,recruit,broker and (un)advertise primitives in the ACL (to be more precise, these primitives are moved to the AMS).
- FIPA-ACL and KQML have philosophical differences in terms of what is done in the wrapper language and what is delegated by the wrapper language to the content language. An agent A could tell an agent B to achieve a goal X in a couple of different ways (where the ACL performatives are in orange and the content language clauses in blue):
 - from A to B: achieve goal X, - thus agents A asks agent B to achieve goal X (all in the ACL vocabulary)
 - from A to B: tell (achieve goal X). - here agent A says something to agent B that the ACL does not understand (but the content language interpreter interprets as a request to achieve goal X).

In the former (the KQML way), the wrapper language has a specific achieve performative for achieving goals, as it does not assume the content language necessarily has this capability. In the latter case (the FIPA-ACL way), the achieve performative is pushed into the content language, thus drawing a line between the content language and ACL.

Note:

The FIPA-ACL specification and message structure can be found in Appendix A of this report

3.4.2 Agents Interaction Protocol Design

Our next step towards system implementation was to design the interaction protocol. We use AUML sequence diagrams.

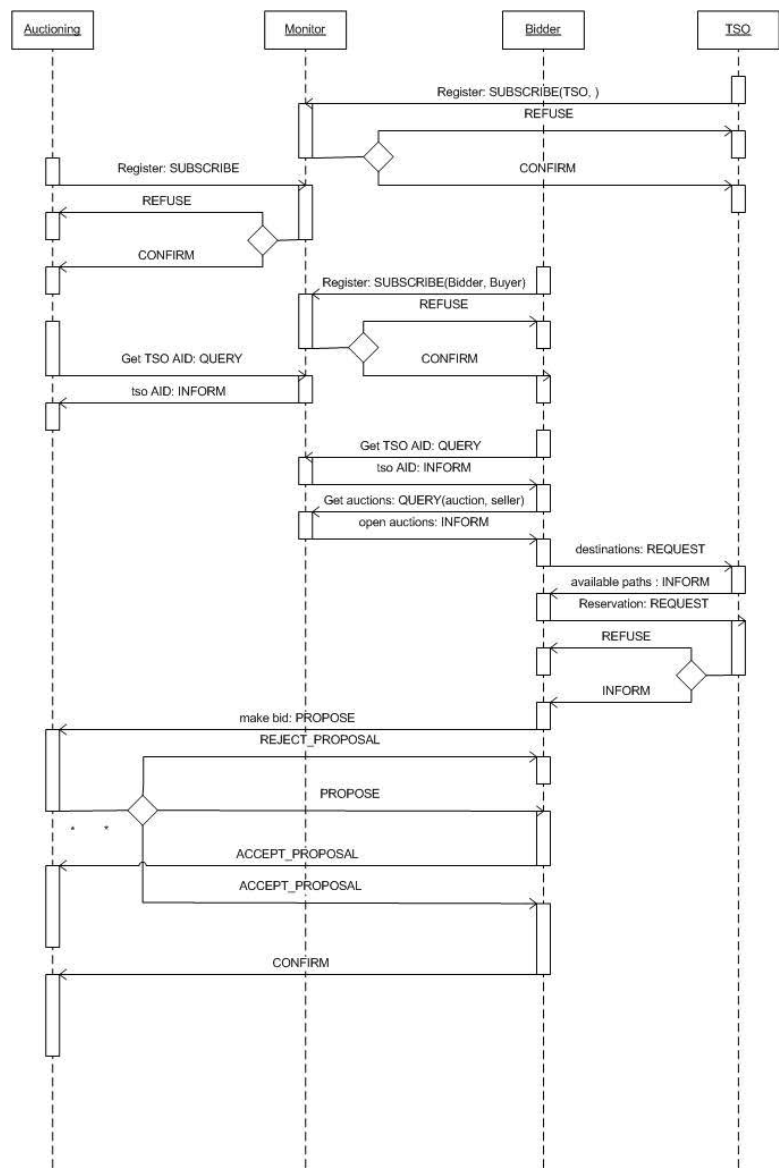


Figure 3.6: Agents Interaction Protocol

Chapter 4

Agency Implementation

In this chapter we will discuss the actual implementation of the system, the environment where the agents will live on, the behaviours of the agents, and the company registration module implemented as a web site.

4.1 JADE Platform

We discussed in the previous chapters how agents communicate, described the agents in our system, and their actions. However, we did not mention where the agents will live and the nature of the environment containing the agents. This is what we call the Jade Platform. We decided to choose JADE as our environment in developing multi-agent systems since it seemed to provides us with the requirements we need in our system.

JADE, short for Java Agent DEvelopment, is a middleware that facilitates the development of multi-agent systems. It includes:

- A **runtime environment** where JADE agents can live and that must be active on a given host before one or more agents can be executed on that host.
- A **library** of classes that programmers can use (directly or by specializing them) to develop their agents.
- A suite of **graphical tools** that allows administrating and monitoring the activity of running agents.

Before running any agent, an instance of JADE runtime environment must be running to hold the agents. Any running instance of JADE runtime environment is called a **Container** since it contains several agents. A

Platform is a set of active containers. Any platform must have a single special container called the **Main-Container**, and all other containers must be normal, i.e. non-main, and they must also register themselves with the main container. So, the first container to start is made main by default. If another main-container is started anywhere in the network, it will constitute a different and independent platform where any normal container can possibly register.

Main-Containers are special since they have two special agents: the Agent Management System agent (AMS) and Directory Facilitator agent (DF), which are started automatically when the main container is launched. The **AMS** maintains a directory of agent identifiers (AID) that contain transport addresses (amongst other things) for agents registered with the agent platform (AP). The AMS is responsible of managing the operation of an AP, such as the creation of agents, the deletion of agents, deciding whether an agent can dynamically register with the AP and overseeing the migration of agents to and from the AP (if agent mobility is supported by the AP). Since different APs have different capabilities, the AMS can be queried to obtain a description of its AP. The **DF** provides a yellow pages service to make agents find the providers of services they need. An illustration of containers and platforms is depicted in 4.1

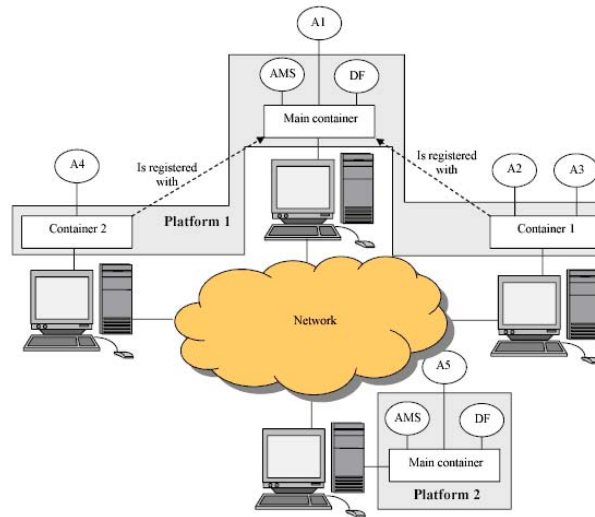


Figure 4.1: Containers and Platforms

4.2 Agents Platform for Our System

In our system, each container represents a single company where the auctioning and bidding agents for that company live. The TSO agent can live in any of these containers or in an independent one. There is a server where the Monitor agent lives. The Monitor agent is only used to help agents in the system identify and find each other then these agents communicate directly independent of the Monitor agent thus reducing the load on the server.

4.3 Agents Implementation

4.3.1 Seller Auctioning Agent Implementation

Registration Behaviour:

This is the first behavior that executes. It is a one-shot behavior that is responsible of registering the auctioning agent with the monitor agent. It proceeds as follows: Send the message:

To: Monitor Agent

Performative: SUBSCRIBE

Parameters:

Role: Auction

Type: Seller

Company: company name

Password: company password

Capacity: capacity to sell

Price: price per watt

Minimum-contract amount

Start and end delivery time

The auctioning agent then waits for a confirmation or refusal message from the monitor.

On End: start the Get TSO AID behavior.

Get TSO AID Behaviour:

This is a one-shot behavior responsible of obtaining the Transmission System Operator agent (TSO) agent identifier.

Send the message:

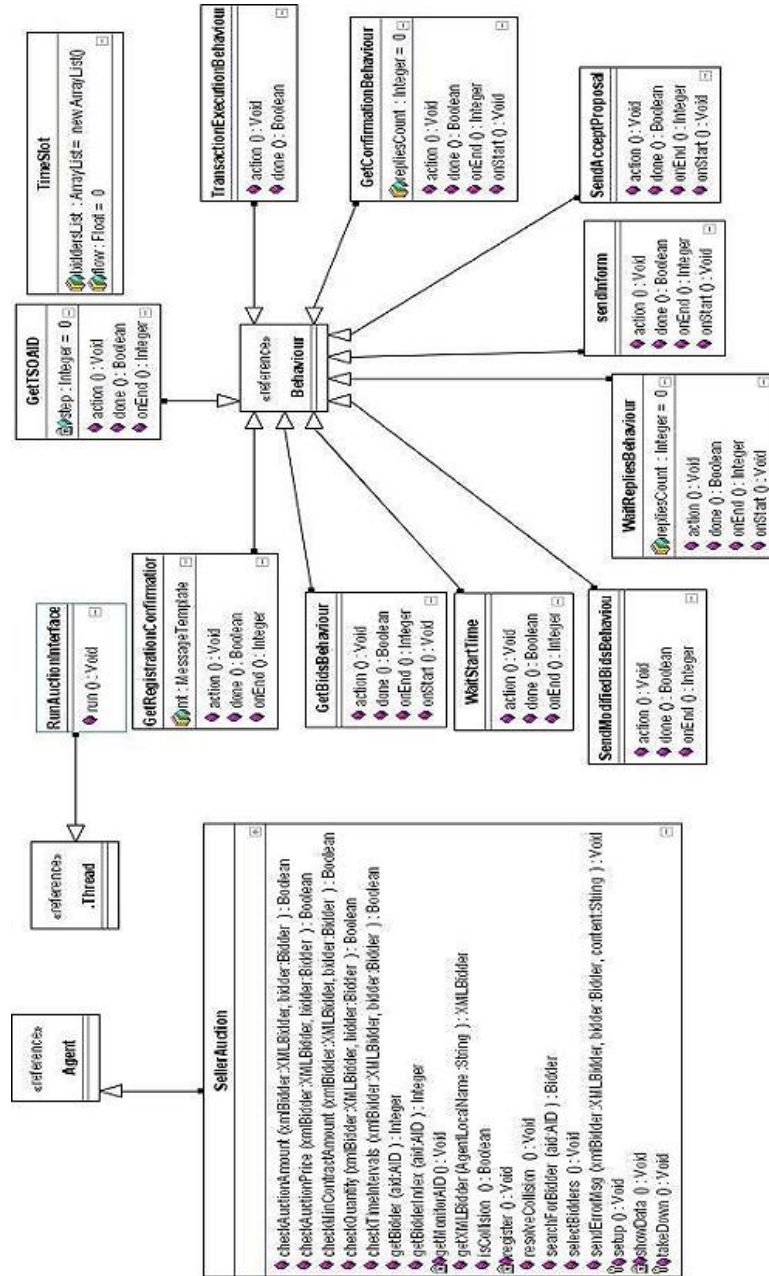


Figure 4.2: UML class diagram for seller auctioning agent

To: Monitor Agent
Performative: QUERY-REF
Parameters:
 role = TSO

Wait to receive the message:

From: Monitor Agent
Performative: INFORM
Content Object: TSO aid

On End: start the Get Bids behavior.

Get Bids Behaviour:

This behaviour is responsible of getting the bidders' bids, the auctioning agent validate these proposals, reject non-valid bids and notify the bidders. The auctioning agent waits to receive the message:

From: bidder agent
Performative: PROPOSE
Parameter:
 Company: bidder's company name
 amount = capacity to bid with.
 price = auction price.
 minMonth = bidder's start month for electric power delivery
 minYear = bidder's start for electric power delivery
 maxMonth = bidder's end month for electric power delivery
 maxYear = bidder's end year for electric power delivery

The auctioning agent validates the bid and checks auction price with his own. Also, checks if the bidder minimum delivery time is not before auction start delivery time, and maximum delivery time is not after auction end delivery time. If the bid was valid then, the auctioning agent store the bid, otherwise, the auctioning agent reject the bid and notify the bidder by sending the message:

To: Bidder Agent.
Performative: REJECT-PROPOSAL
Parameters:
 Amount: capacity to bid with.
 Price: auction price.

The auctioning agent collect bids till bidding time expires, we assume that bidding time will be quarter and half hour.

On End: the auctioning agent detect the collision, if exist the auctioning agent starts Resolve Collision Behaviour. Otherwise, the auctioning agent sends for each bidder the message:

To: Bidder Agent.

Performative: ACCEPT-PROPOSAL

Parameters:

Amount: capacity to bid with.

Price: auction price.

After sending the message to all bidders, the auctioning agent starts Get Conformation Behaviour.

Resolve Collision Behaviour:

This is a one-shot behaviour responsible to resolve the collision and distribute the auction capacity amongst the bidders. The auctioning agent tries to maximize its profit, and because of the price is constant for all bidders then, the auctioning agent will distribute the capacity amongst the bidders according to bidders agents required capacity and time slots.

The objective function is:

$$\text{Maximize } Z = \sum_n^1 B_i * N_i * B'_i$$

Where:

n = number of bids received.

B_i = Bid for bidder i.

N_i = Time slots count required by bidder i.

B'_i = The bid modifier for bidder i.

And the constraints are:

For each time slot: $\sum \text{Bids Received} \prec \text{auction capacity}$

Using the simplex algorithm to solve these linear equations, B'_i will take a value from 0 to 1, so that, the modified bid for bidder i will be $B'_i * B_i$

The auctioning agent excludes the bidders whose modified bids fall below the minimum-contract amount, and sends to the other bidders the message:

To: Bidder agent

Performative: PROPOSE

Parameters:

Amount: Modified Bids.

Price: auction price.

On End: The auctioning agent then starts Get Accept Behaviour.

Get Accept Behaviour:

This behaviour is responsible of getting the bidders approval or refusal for the modified bids proposal, it is a cyclic behaviour that repeats until all bidders send replies or waiting time expires.

Waits to receive the message:

To: Bidder agent

Performative: ACCEPT-PROPOSAL

Action:Add that bidder to winningBidders list.

OR

Wait to receive the message:

To: Bidder agent

Performative: REFUSE

Action:Add that bidder to rejectedBidders list; return the rejected capacity to auction capacity.

If the waiting time expires the auctioning agent assume that bidders with no reply are rejected bidders and add them to rejectedBidders list, return the rejected capacity to the auction capacity.

On End: If auction remaining capacity can satisfy one or more bidder from the excluded bidder, the auctioning agent sends the message:

To: Bidder Agent.

Performative: ACCEPT-PROPOSAL

Parameters:

Amount: amount of bid.

Price: auction price.

After sending the message to all bidders, the auctioning agent starts Get Confirmation Behaviour.

Get Confirmation Behaviour:

This behaviour is responsible to get accepted bidders confirmations.
The auctioning agent waits for the message:

To: Bidder agent

Performative: CONFIRM

Action:the auctioning agent adds the bidder to winningBidders list

OR

Waits for the message:

To: Bidder agent

Performative: DISCONFIRM

Action:the auctioning agent adds the bidder to forfeitedBidders list

The auctioning agent repeats the behaviour till waiting time expires or receives messages from all bidders. If time expires, the auctioning agent adds any un-replied bidder to the forfeitedBidders list.

On End: the auctioning agent starts Transaction Execution Behaviour If the forfeitedBidders list is not empty, the auctioning agent sends the message:

To: Monitor

Performative:INFORM

Conversation Id:forfeited agents

Parameters:

Price modifier:auction's price modifier

Content Object: forfeitedBidders list

Transaction Execution Behaviour:

This is a one-shot behaviour responsible to finalize the transaction.

The auctioning agent sends the message:

For each bidder at excludedBidders list

To:Bidder agent

Performative:REJECT-PROPOSAL

And

For each winning bidder

Sends the message:

Performative:INFORM

Conversation Id:Transaction Execution

Parameters:

Total Price:the total price will be bay by that bidder.

On End: If the remaining capacity at each time slot is greater than the minimum contract amount, sign out with the monitor and sign in with the updated capacity. Otherwise, sign out and kill the agent.

Algorithms

Detect Collision:

For each time slot, the auctioning agent checks if the total capacity required at that time slot exceeds the offered capacity then, there exist a collision, otherwise, no collision.

Resolve Collision:

The objective function is:

$$\text{Maximize } Z = \sum_n^1 B_i * N_i * B'_i$$

Where:

n = number of bids received.

B_i = Bid for bidder i .

N_i = Time slots count required by bidder i .

B'_i = The bid modifier for bidder i .

And the constraints are:

For each time slot: $\sum \text{Bids Received} \prec \text{auction capacity}$

4.3.2 Buyer Auctioning Agent Implementation

The buyer auctioning agent is very similar to the seller auctioning agent but, it accepts only the bidder who can supply auctioning agent with the capacity for the whole period. The buyer auctioning can receive bids exceeds the capacity to purchase (i.e., collision condition). The buyer auctioning can resolve collision by distributing the capacity amongst the bidders proportionally to their bids.

$$B'_i = C * \frac{B_i}{\sum_{i=1}^n B_i}$$

Where:

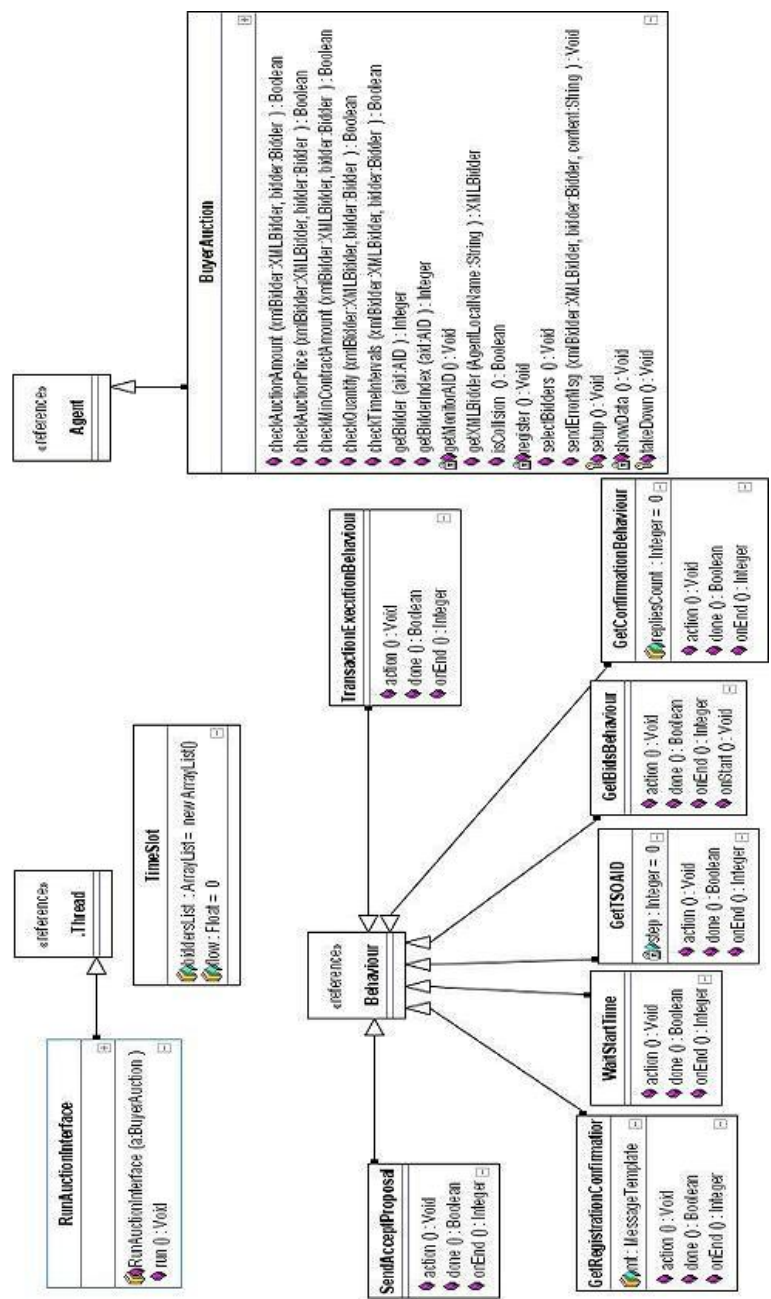


Figure 4.3: UML class diagram for buyer auctioning agent

n = number of bids received
 C = posted capacity to purchase
 B'_i = Modified Bid
 B_i = Original Bid

This solution works in most cases but problems can arise if the number of bidders is large and/or there is a great disparity between the sizes of the bids. In either event, the modified bids can drop below the minimum contract amount. In that event the following heuristic is proposed:

For each bidder whose modified bid falls below the minimum contract amount:

- Add that bidder to the excluded bidders list
- Return the bid amount to the auction remaining capacity

After that:

While the remaining capacity exceeds the minimum contract amount:

The auctioning agent tries to select one bidder from the excluded bidders whose bid amount less than or equal the remaining capacity. If exists, Sends the message:

To: Bidder Agent.

Performative: ACCEPT-PROPOSAL

Parameters:

Amount: amount of bid.

Price: auction price.

Remove that agent from the excluded list.

Decrement the remaining capacity by bid amount.

4.3.3 Buyer Bidding Agent Implementation

This is the first behavior that executes. It is a one-shot behavior that is responsible of registering the auctioning agent with the monitor agent. It proceeds as follows: Send the message:

To: Monitor Agent

Performative: SUBSCRIBE

Parameters:

Role: Bidder

Type: Buyer

Company: company name

Password: company password

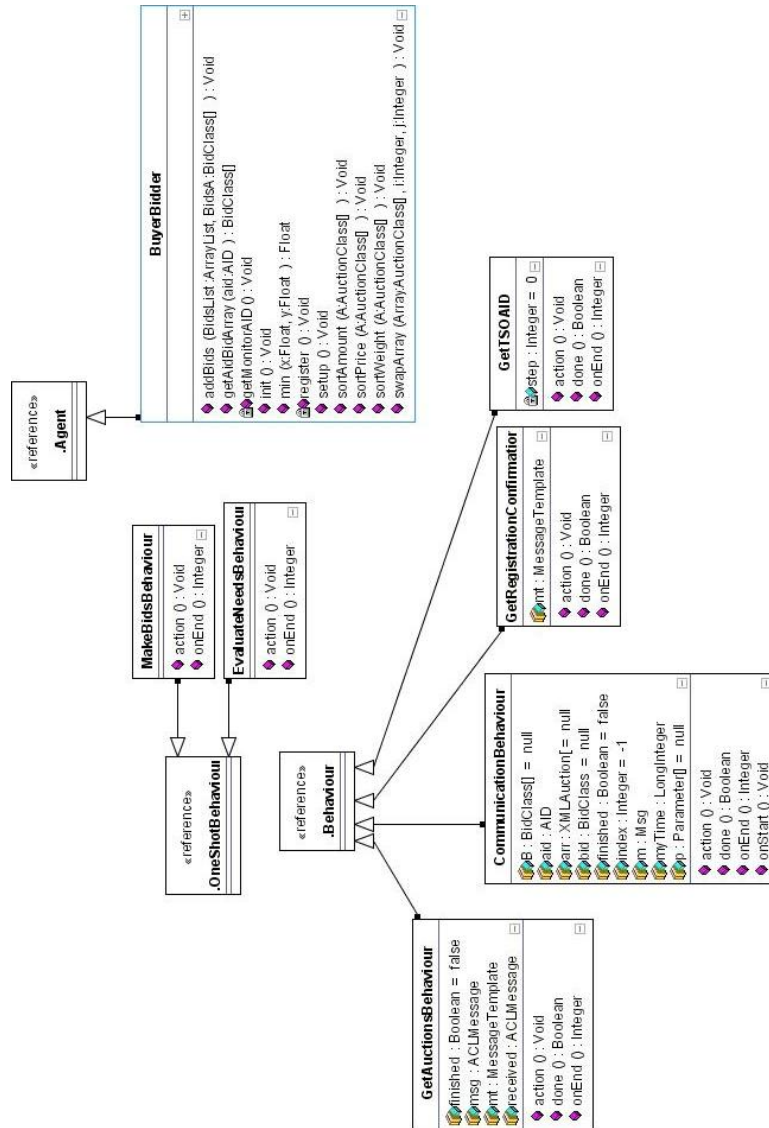


Figure 4.4: UML class diagram for buyer bidding agent

The auctioning agent then waits for a confirmation or refusal message from the monitor.

On End: start the Get TSO AID behavior.

Get TSO AID Behaviour:

This is a one-shot behavior responsible of obtaining the Transmission System Operator agent (TSO) agent identifier.

Send the message:

To: Monitor Agent

Performative: QUERY-REF

Parameters:

role = TSO

Wait to receive the message:

From: Monitor Agent

Performative: INFORM

Content Object: TSO aid

On End: start the Get Auctions behavior.

Get Auctions Behavior:

This behavior is responsible of getting the current open auctions from the Monitor Agent. It is a cyclic behavior that repeats until at least one of the current auctions satisfies the bidder's constraints.

Send the message:

To: Monitor Agent

Performative: QUERY-REF

Parameters:

role = auction

type = seller

Wait to receive the message:

From: Monitor Agent

Performative: INFORM

Parameters:

next round time = time for the next auction to begin

end bid time = time when bidding in the current session ends

Content Object: Array of open auctions.

Action: If the time to end of bid time $j =$ time constant then return

For each auction in the array

if

the auction's min contract amount j the bidder's min contract amount, and

the auction's price $j =$ the bidder's max price

then add this auction to array temp

Send the message:

From: TSO Agent

Performative: INFORM

Content Object: Array of available paths from each of the auctioning agents' companies in array temp represented as the possible electric power flow and transmission cost.

Action: For each company in the array

Calculate the average transmission cost from each company, weighted according to flow on paths. If auction price + transmission cost $j =$ max price the bidder can pay

then add the corresponding auction to the array AuctionsArray.

If AuctionsArray is empty then block to the next round time, and repeat this behavior.

On End: start the Make Bids Behavior.

Make Bids Behavior:

This is a one-shot behavior responsible of selecting the auctions to bid in, the amount to bid with and performing the actual bidding.

The auctions to bid in and the amounts to bid with are selected according to the bidding algorithms, which will be discussed later.

Once an auction from the AuctionsArray has been selected for bidding, the bidding agent sends the message:

To: TSO Agent

Performative: REQUEST

Parameters:

sender company = bidder's company name

destination = auctioning agent company name

minimumFlow = bidding agent minimum contract-amount

maximumCost = the average transmission cost calculated earlier

startMonth = bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery
endMonth= bidder's end month for electric power delivery
endYear = bidder's end year for electric power delivery

Content Object: Array temp of auctions that initially satisfy bidder's constraints.

Wait to receive the message:

From: TSO Agent

Performative: REFUSE

Action: Abandon this auction and move on to the next.

OR

From: TSO Agent

Performative: CONFIRM

Action: Add this bid to the array BidsArray.

For every bid in the BidsArray
sends the message:

To: Seller Auctioning Agent

Performative: PROPOSE

Parameters:

amount= capacity to bid with.

price= auction price.

companyName= bidder's company name.

startMonth =bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery

endMonth= bidder's end month for electric power delivery

endYear = bidder's end year for electric power delivery

set that bid state to 'SENT'.

On End: start the Communication Behavior.

Communication Behavior:

This is a cyclic behavior that receives the replies from the auctioning agents where the bidder has just bid in and responds to them. It is repeated until the next round time comes.

Wait to receive the message:

From: Seller Auctioning Agent

Performative: ACCEPT-PROPOSAL

Parameters:

Amount: amount of bid.

Price: auction price.

Action: Get the bid corresponding to that auctioning agent
send the message:

To: Seller Auctioning Agent

Performative: CONFIRM

Parameters:

Amount: amount of bid.

Price: auction price.

set that bid state to 'Accepted'.

OR

Wait to receive the message:

From: Seller Auctioning Agent

Performative: REJECT-PROPOSAL

Parameters:

Amount: amount of bid.

Price: auction price.

Action: Get the bid corresponding to that auctioning agent set that bid
state to 'Rejected'.

OR

Wait to receive the message:

From: Seller Auctioning Agent

Performative: PROPOSE

Parameters:

Amount: amount of bid.

Price: auction price.

Action: Get the bid corresponding to that auctioning agent.

If amount \prec bidder's minimum contract amount
send the message:

To: Seller Auctioning Agent

Performative: REJECT-PROPOSAL

Parameters:

Amount: amount of bid.

Price: auction price.

Else

send the message:

To: Seller Auctioning Agent

Performative: ACCEPT-PROPOSAL

Parameters:

Amount: amount of bid.

Price: auction price.

set that bid state to 'CHANGED'.

While there's still time to the next session time, repeat the behavior

On End: start the Evaluate Needs Behavior.

Evaluate Needs Behavior:

This is a one-shot behavior that is responsible of reevaluating the bidder's needs after each auction session and adjusting the bidder's constraints.

For every bid in the BidsArray

If its state is 'CHANGED' or 'ACCEPTED'

Then reduce the bidder's capacity by the bid amount

Adjust the bidder's constraints according to the bidding algorithm used.

If needs < 0 then perform clean up and kill agent.

Else block till next session time

On End: start the Get Auctions Behavior.

Bidding Algorithms

We have provided several bidding algorithms from which the user can choose from, in an attempt to meet the different strategies that companies can have.

First: The Greedy Algorithm

At any time the target price is the maximum price the bidder can pay.

AuctionsArray is sorted ascendingly according to totalCost (posted price + average transmission cost), then for all auctions with same totalCost, sort descendingly according to the auction's posted capacity (the greater the capacity, the greater the probability that the bidder will most of his needs). Auctions are selected for bidding in the order of their appearance in the

AuctionsArray.

The amount of the bid is selected as follows:

If the auction's posted capacity \succ bidder needs

Then

if bidder needs \geq auction's posted minimum contract amount

then

bid amount = needs

needs = 0

else don't bid in that auction

else

bid amount = auction's posted capacity

needs = needs - bid amount

repeat the process until needs = 0

Note:

There is no adjustment of constraints in this algorithm.

Second: The Cool and Lazy Algorithms

This is the same as the Greedy algorithm except that instead of using the target price is initially the best won bid in the last 24 hours in the case of the Cool Algorithm and the best won bid in the last 24 hours - a price step in the case of the Lazy Algorithm.

The algorithms proceed as in the Greedy case except that there is a target price adjustment. In order to ensure that the bidder satisfies all its needs, the target price is increased as auctions sessions go by and when remaining time till start of delivery reaches some known value the algorithm would turn into the Greedy method. Some heuristics are also used such as: if the time remaining to delivery time $\leq x$ and bidder need's $\geq z\%$ of initial capacity then target price = maximum price - y * price step.

Third: The Weights Algorithm

This algorithm requires the user to enter two values: weight on cost (w_c) and weight on flow (w_f) such that $w_f + w_c = 1$. The objective function would be to maximize:

$$w_c * \frac{PostedCapacity}{TotalNeeds} + w_f * \frac{TargetPrice}{TotalCost}$$

This can be interpreted as follows:

When $w_f \succ w_c$, this user would prefer to bid in an auction with a larger capacity compared to the bidder's needs than in an auction with less capacity and less cost compared to the bidder's target price. While when $w_f \prec w_c$, this user would prefer to bid in an auction with a lower cost compared to the bidder's target price than in an auction with a higher cost and a higher capacity compared to the bidder's needs.

AuctionsArray is sorted descendingly according to weighting function. The bidder starts with a high target value for weighting function (e.g. target value ≥ 1). To ensure that the bidder satisfies all its needs, the target value is decreased as auctions sessions go by and when remaining time till start of delivery reaches some known

value the algorithm would turn into the Greedy method. Also similar heuristics to those in the Cool and Lazy algorithms are used here.

4.3.4 Seller Bidding Agent Implementation

Seller Bidding agent is very similar to the Buyer Bidding agent except that as a seller it is also faced with collisions which it must resolve. Therefore after the Seller determines the auctions that satisfy its constraints (i.e. the auctions where it may bid), it uses the simplex method, the same way as in the Seller Auctioning agent, to get as much a good combination of auctions and bid amounts in order to maximize its profit as it can.

Detect Collision:

For each time slot, the bidding agent checks if the total capacity posted at that time slot exceeds the required capacity, then there exist a collision, otherwise, no collision.

Resolve Collision:

The objective function is:

$$\text{Maximize } Z = \sum_{n=1}^i N_i * F_i * \min(Q_i, Tflow_i) * (P_i - Tcost_i)$$

Where:

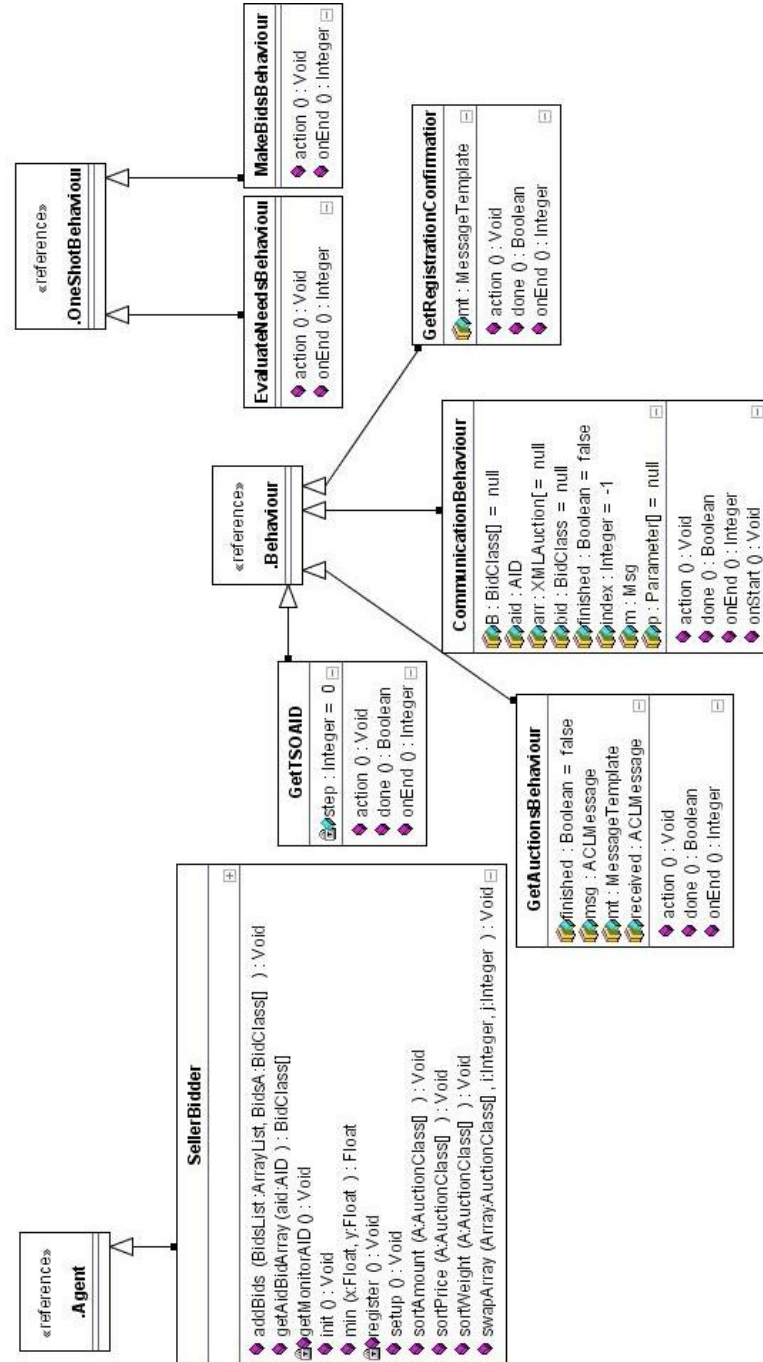


Figure 4.5: UML class diagram for seller bidding agent

n = number of feasible auctions.
 N = Time slots count required by auction i .
 Q = bid amount for auction i .
 T_{flow} = available transmission flow.
 T_{cost} = available transmission cost.
 F = fraction of auction posted quantity with which bidder will bid.

And the constraints will be:

For each time slot: $\sum \text{Bids made} \prec \text{Bidder Capacity}$

4.3.5 TSO Agent Implementation

TSO Behaviour:

It is a cyclic behaviour that receives the following messages types:

1. Information from the monitor agent including adding/dropping a company/line.
2. Request from the bidder agent asking for all the feasible paths between its company and each one of the auctioning agents' companies during a specific time interval.
3. Request from the bidder agent asking for reserving the best path(s) between its company and a specific auctioning agent's company with a given flow during a specific time interval with a minimum acceptable flow and a maximum acceptable transmission cost (per unit flow).
4. Confirmation from the auctioning agent in order to confirm the flow reservation between its company and a specific bidder agent's company by sending the difference between the previously reserved flow and the actual consumed one during a specific time interval.

The TSO agent receives information from the monitor agent including dropping a specific line:

In that case it will be waiting for the following message:

From: monitor agent

Performative: INFORM

Parameters:

companyName: source company name.

destCompany: destination company name.

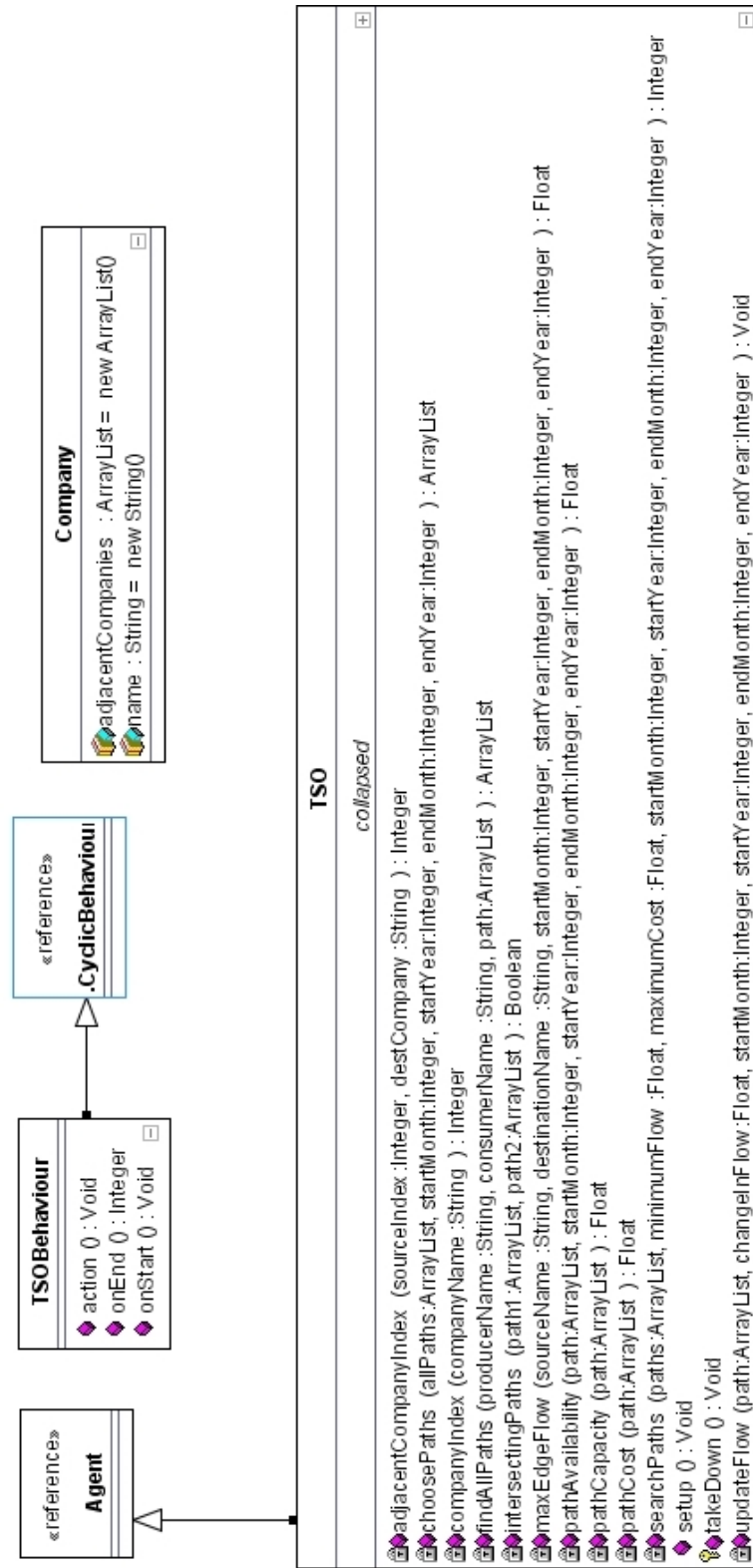


Figure 4.6: UML class diagram for TSO agent

If the line is already not existing in the grid map,
The TSO agent will send the following message:

To: monitor agent

Performative: REFUES

Content: Line is already not existing

Parameters:

companyName: source company name.

Else, the TSO agent will drop the line from the grid map,
It will send the following message:

To: monitor agent

Performative:CONFIRM

Conversation ID: DropLine

Parameters:

companyName: source company name.

OR

The TSO agent receives information from the monitor agent including adding
a specific line:

In that case it will be waiting for the following message:

From: monitor agent

Performative: INFORM

Conversation ID: AddLine

Content Object: Object of type Line: destCompany (destination
company name), cost (transmission cost per unit flow), capacity (maximum
flow that can be passed in the line), array of objects of type TimeFlow:
month, year, line flow

Parameters:

companyName: source company name.

The TSO agent will add the line to the grid map, It will send the following
message:

To: monitor agent

Performative:CONFIRM

Conversation ID: AddLine

Parameters:

companyName: source company name.

OR

The TSO agent receives information from the monitor agent including drop-
ping a specific company.

In that case it will be waiting for the following message:

From: monitor agent
Performative:INFORM
Conversation ID: DropCompany
Parameters:
 companyName: source company name.

If the company is already not existing in the grid map, The TSO agent will send the following message:

To: monitor agent
Performative:REFUSE
Content: company is already not existing
Parameters:
 companyName: source company name.

Else, the TSO agent will drop the company from the grid map, It will send the following message:

To: monitor agent
Performative:CONFIRM
Conversation ID: DropCompany
Parameters:
 companyName: source company name.

OR

The TSO agent receives information from the monitor agent including adding a specific company:

In that case it will be waiting for the following message:

From: monitor agent
Performative:INFORM
Conversation ID: AddCompany
Parameters:
 companyName: source company name.

If the company is already existing in the grid map, The TSO agent will send the following message:

To: monitor agent
Performative:REFUSE
Content: company is already existing
Parameters:
 companyName: source company name.

Else, the TSO will add the company to the grid map, It will send the following message:

To: monitor agent
Performative:CONFIRM
Conversation ID: AddCompany
Parameters:
 companyName: source company name.

OR

The TSO agent receives a request from the bidder agent asking for all the feasible paths between its company and each one of the auctioning agents' companies during a specific time interval.

In that case it will be waiting for the following message:

From : bidder agent
Performative: REQUEST
Parameters:
 sender company= bidder's company name
 destination = auctioning agent company name
 startMonth =bidder's start month for electric power delivery
 startYear = bidder's start for electric power delivery
 endMonth= bidder's end month for electric power delivery
 endYear = bidder's end year for electric power delivery

If the bidder agent's company is not existing in the grid map,The TSO agent will send the following message:

To: bidder agent
Performative:REFUSE
Content: Sender company is not existing
Parameters:
 companyName:bidder agent's company name.

Else, the TSO agent will return the non-intersecting paths and for those paths that are intersecting, it will return the one with the highest ratio: path availability / path cost.

It will send the following message:

TO : bidder agent
Performative: INFORM
Parameters:
 sender company= bidder's company name

startMonth = bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery

endMonth = bidder's end month for electric power delivery

endYear = bidder's end year for electric power delivery

Content Object: Array of objects each of type DestReply: name (auctioning agent's company name), agentID (auctioning agent's ID), array of objects each of type Path: cost (path edges sum of transmission cost per unit flow), availability (minimum (edge capacity - edge maximum flow in interval time slots) for all edge belongs to the path)

OR

The TSO agent receives a request from the bidder agent asking for reserving the best path(s) between its company and a specific auctioning agent's company with a given flow during a specific time interval with a minimum acceptable flow and a maximum acceptable transmission cost (per unit flow). In that case it will be waiting for the following message:

From : bidder agent

Performative: REQUEST

Parameters:

sender company = bidder's company name

destination = auctioning agent company name

minimumFlow = bidding agent minimum contract-amount

maximumCost = the average transmission cost calculated earlier

startMonth = bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery

endMonth = bidder's end month for electric power delivery

endYear = bidder's end year for electric power delivery

Content Object: Array temp of auctions that initially satisfy bidder's constraints.

If the bidder agent's company is not existing in the grid map, The TSO agent will send the following message:

To: bidder agent

Performative: REFUSE

Content: Sender company is not existing

Parameters:

companyName: bidder agent's company name.

Else, if the maximum acceptable transmission cost (per unit flow) is not sufficient to reserve the minimum acceptable flow, The TSO agent will send the following message:

TO : bidder agent

Performative: REFUSE

Content: maximumCost is not sufficient to reserve the minimumFlow

Parameters:

sender company= bidder's company name

destination = auctioning agent company name

minimumFlow = bidding agent minimum contract-amount

maximumCost=the average transmission cost calculated earlier

startMonth =bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery

endMonth= bidder's end month for electric power delivery

endYear = bidder's end year for electric power delivery

Else, if minimum acceptable flow can not be satisfied, The TSO agent will send the following message:

TO : bidder agent

Performative: REFUSE

Content: minimumFlow can not be satisfied

Parameters:

sender company= bidder's company name

destination = auctioning agent company name

minimumFlow = bidding agent minimum contract-amount

maximumCost=the average transmission cost calculated earlier

startMonth =bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery

endMonth= bidder's end month for electric power delivery

endYear = bidder's end year for electric power delivery

Else, the TSO reserves the best path(s) between the bidder agent's company and the auctioning agent's company during the given time interval satisfying the minimum acceptable flow and the maximum acceptable transmission cost (per unit flow), It will send the following message:

To : bidder agent

Performative: INFORM

Content: minimumFlow can not be satisfied

Parameters:

sender company= bidder's company name

destination = auctioning agent company name

Flow = reserved flow

Cost=transmission cost per unit of the reserved flow.

startMonth = bidder's start month for electric power delivery
startYear = bidder's start for electric power delivery
endMonth = bidder's end month for electric power delivery
endYear = bidder's end year for electric power delivery

OR

The TSO agent receives a confirmation from the auctioning agent in order to confirm the flow reservation between its company and a specific bidder agent's company by sending the difference between the previously reserved flow and the actual consumed one during a specific time interval.

In that case it will be waiting for the following message:

From : auctioning agent

Performative: CONFIRM

Content: minimumFlow can not be satisfied

Parameters:

sender company = bidder's company name

destination = auctioning agent company name

differenceFlow = difference between the previously reserved flow and the actual consumed one during a specific time interval.

startMonth = bidder's start month for electric power delivery

startYear = bidder's start for electric power delivery

endMonth = bidder's end month for electric power delivery

endYear = bidder's end year for electric power delivery

The TSO updates the flow reservation between the auctioning agent company and the bidder agent's company by the difference between the previously reserved flow and the actual consumed one during the given time interval

4.3.6 Monitor agent implementation

Monitor Behaviour:

It is a cyclic behaviour that receives various requests from auctioning and bidding agents.

It waits for the message:

From : auctioning agent

Performative: SUBSCRIBE

Parameters:

sender company = Auction's company name

password = company password

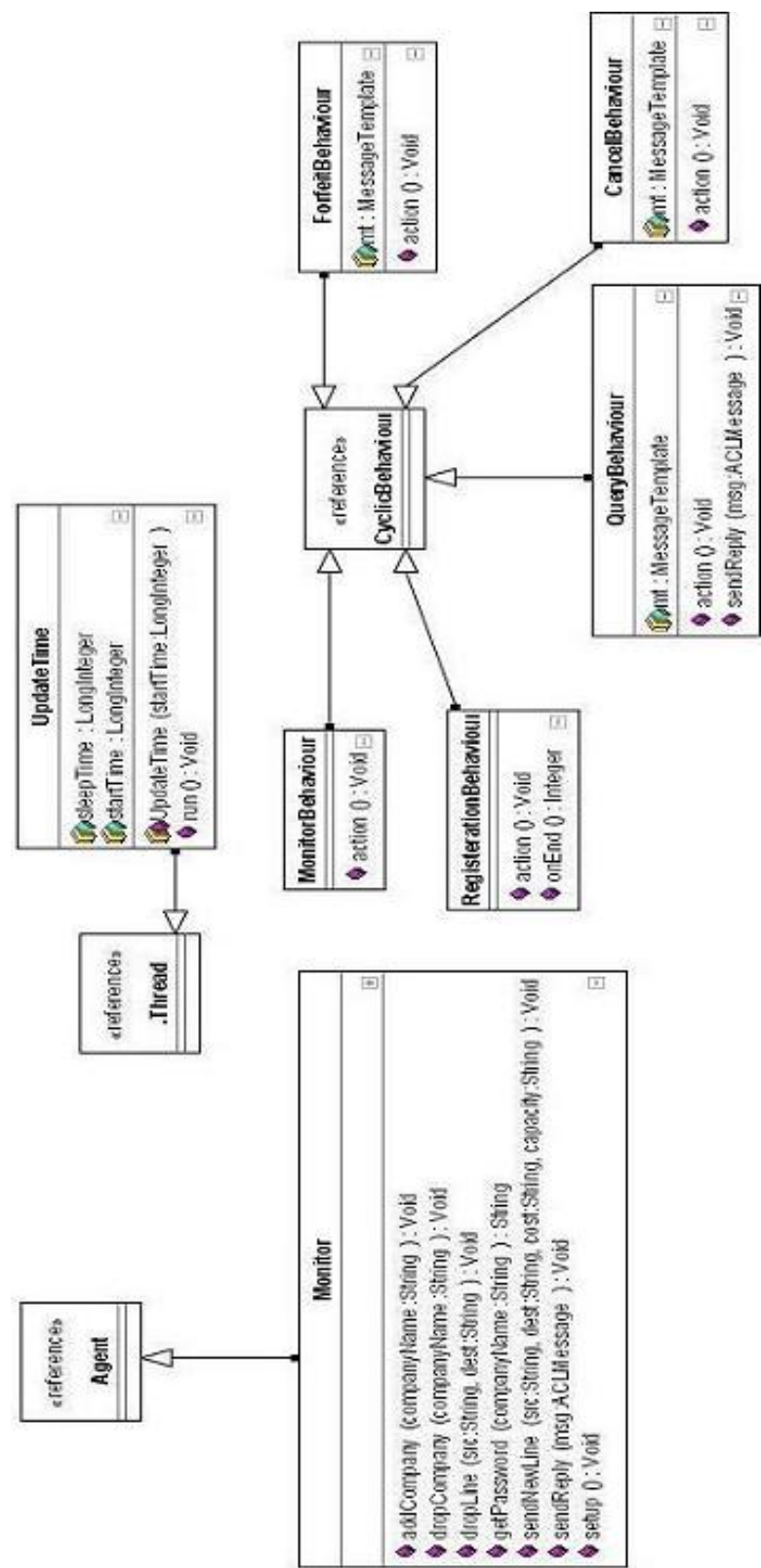


Figure 4.7: UML class diagram for monitor agent

Role =Auction
Type =Buyer or Seller
Capacity =capacity to sell or buy
Price=price per watt
Minimum contract amount
startMonth =bidder's start month for electric power delivery
startYear = bidder's start for electric power delivery
endMonth= bidder's end month for electric power delivery
endYear = bidder's end year for electric power delivery

The monitor agent checks if the company name and password matches the stored with its data then, if the type was buyer then, store the auctioning agent as "buyer auction" else, if the type was seller then, store the auctioning agent as "seller auction". And send the message:

To : auctioning agent
Performative: CONFIRM

Otherwise, send the message:

To : auctioning agent
Performative: REFUSE

OR

It waits for the message:

From : Bidding agent
Performative: SUBSCRIBE
Parameters:

sender company= Bidding's company name
password= company password
Role =Bidder
Type =Buyer or Seller

The monitor agent checks if the company name and password matches the stored with its data then, if the type was buyer then, store the bidding agent as "buyer bidder" else, if the type was seller then, store the bidding agent as "seller bidder ". And send the message:

To : auctioning agent
Performative: CONFIRM

Otherwise, send the message:

To : auctioning agent
Performative: REFUSE

OR

It waits the message:

From : Bidding agent
Performative: QUERY-REF
Parameters:
 Role =auction
 Type =Seller

The monitor agent checks if the bidding agent is an authorized agent (i.e., registered agent) then, the monitor agent sends the message:

To : Bidding agent
Performative: INFORM
Parameters:
 end bid time =time when bidding in the current session ends
 next round time = time for the next auction to begin
Content Object:Array of open seller auctions

OR

It waits the message:

From : Bidding agent
Performative: QUERY-REF
Parameters:
 Role =auction
 Type =Buyer

The monitor agent checks if the bidding agent is an authorized agent (i.e., registered agent) then, the monitor agent sends the message:

To : Bidding agent
Performative: INFORM
Parameters:
 end bid time =time when bidding in the current session ends
 next round time = time for the next auction to begin
Content Object:Array of open buyer auctions

OR

It waits the message:

From : An agent
Performative: QUERY-REF
Parameters:
 Role =TSO

The monitor agent checks if the agent is an authorized agent (i.e., registered agent) then, the monitor agent sends the message:

To : Agent
Performative: INFORM
Content Object:TSO aid

OR

It waits the message:

From : An agent
Performative: CANCEL

The monitor agent checks if the agent is an authorized agent then, the monitor remove that agent from the registered agents.

OR

It waits the message:

To : Auctioning agent
Performative: INFORM
Parameters:
 companyName=forfeited company name
 forfeitedAmount=forfeited amount

The monitor agent checks if the auctioning agent is a buyer agent and the forfeited company is a seller agent, or the auctioning agent is a seller agent and the forfeited company is a buyer agent then, it takes the forfeited amount from the forfeited company deposit and add this amount to the auctioning company.

Add new line:

This method responsible to send a new transmission line between two companies; the monitor agent sends the message:

TO: TSO agent
Performative: INFORM
Conversation ID: AddLine

Content Object: Object of type Line: destCompany (destination company name), cost (transmission cost per unit flow), capacity (maximum flow that can be passed in the line), array of objects of type TimeFlow: month, year, line flow

Parameters:

companyName: source company name.

Drop line:

This method responsible to notify the TSO that a transmission line between two companies was dropped; the monitor agent sends the message:

Performative: INFORM

Conversation ID: DropLine

Parameters:

companyName: source company name.

Add company:

This method responsible to notify the TSO that a new company was established; the monitor agent sends the message:

Performative: INFORM

Conversation ID: AddCompany

Parameters:

companyName: source company name.

Drop company:

This method responsible to notify the TSO that a company was dropped; the monitor agent sends the message:

Performative: INFORM

Conversation ID: DropCompany

Parameters:

companyName: source company name.

4.4 Data Files

4.4.1 XML files

Since the data is not that large, we use XML to store the data. Here's the XML file structure; Document Type Descriptor (DTD) used by each agent.

Auctioning agent

The auctioning agent stores the bidder's proposals and the exchanged messages.

The DTD file:

```

<!ELEMENT XMLBidders (XMLBidder*)>
<!ELEMENT XMLBidder (CompanyName,BidderID, BidAmount,Status,From,To,
Msg*)>
<!ELEMENT Msg (Performative, Contents?,Direction?, Parameter*)>
<!ELEMENT Parameter (ParameterName, ParameterValue)>
<!ELEMENT CompanyName (#PCDATA)>
<!ELEMENT BidderID (#PCDATA)>
<!ELEMENT BidAmount (#PCDATA)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT Performative (#PCDATA)>
<!ELEMENT Contents (#PCDATA)>
<!ELEMENT ParameterName (#PCDATA)>
<!ELEMENT ParameterValue (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
<!ELEMENT Direction (#PCDATA)>

```

Bidding agent

The bidding agent stores the data about the auctions it participated in, the bidder's proposals and the exchanged messages.

The DTD file:

```

<!ELEMENT XMLAuctions (XMLAuction*)>
<!ELEMENT XMLAuction (AuctionID , AuctionPrice, AuctionAmount,
Msg*)>
<!ELEMENT AuctionID (#PCDATA)>
<!ELEMENT AuctionPrice (#PCDATA)>
<!ELEMENT AuctionAmount (#PCDATA)>
<!ELEMENT Msg (Direction, Performative, Content?, Parameter*)>
<!ELEMENT Direction (#PCDATA)>
<!ELEMENT Performative (#PCDATA)>
<!ELEMENT Content (#PCDATA)>
<!ELEMENT Parameter (ParameterName, ParameterValue)>
<!ELEMENT ParameterName (#PCDATA)>
<!ELEMENT ParameterValue (#PCDATA)>

```

Monitor agent

The monitor agent stores the data about participating companies and the transmission lines between them. It is this data that it uses to initialize the TSO agent at startup.

The DTD file:

```
<!ELEMENT Companies (Company*)>
<!ELEMENT Company (Name,Password,Destination*)>
<!ELEMENT Destination (Name,Cost,Capacity)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Password (#PCDATA)>
<!ELEMENT Cost (#PCDATA)>
<!ELEMENT Capacity (#PCDATA)>
```

4.4.2 Log files

we use the package Log4-j to store the data we need in a similar format to "System.out.println" in Java but with the added advantage of storing data at different levels of importance: fatal, error, warn, info, debug. Through a configuration file we choose the level and data of that level and above will be stored. If we choose debug then all the levels are stored and if we choose inform, only fatal, error, warn and info are stored and so on. While debugging the system we set it on debug, and it was a great help. Later when we finished, we set it on inform because the user is definitely not interested in debugging information.

The configuration file we used for log4-j

```
log4j.rootLogger=debug, stdout, R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log
log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
```

```
log4j.appender.R.MaxBackupIndex=1  
log4j.appender.R.layout=org.apache.log4j.PatternLayout  
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

Chapter 5

Running the Electric Power Market Agency

5.1 Prerequisites

You must have Java 1.4 or higher installed. If you have not done this yet, we suggest that you get the latest Java Runtime Environment from Sun at the URL *[http : //java.sun.com/products/](http://java.sun.com/products/)* where versions for Windows, Linux and Solaris are available.

If you use OS/2, have a look at some 3rd party Java implementations at *[http : //www.goldencode.com/company/software/](http://www.goldencode.com/company/software/)*.

For Apple Macintosh you can use the Java runtime build into MacOS X, as described on *[http : //www.apple.com/](http://www.apple.com/)* without any additional installation.

All runtime environments are available free of charge for personal use. We recommend the latest version of JDK 1.5.

5.2 Application scenario

First, the administrator starts the monitor agent by running "monitor.bat". The monitor agent starts the TSO agent, reads registered companies data, transmission lines between companies, and sends companies and transmission lines data to the TSO agent. After that, the monitor and TSO agents are ready to serve any incoming requests from other agents.

If we assume that the registered companies construct the following graph. And we have two seller auctioning agents (Boston and Miami), two buyer

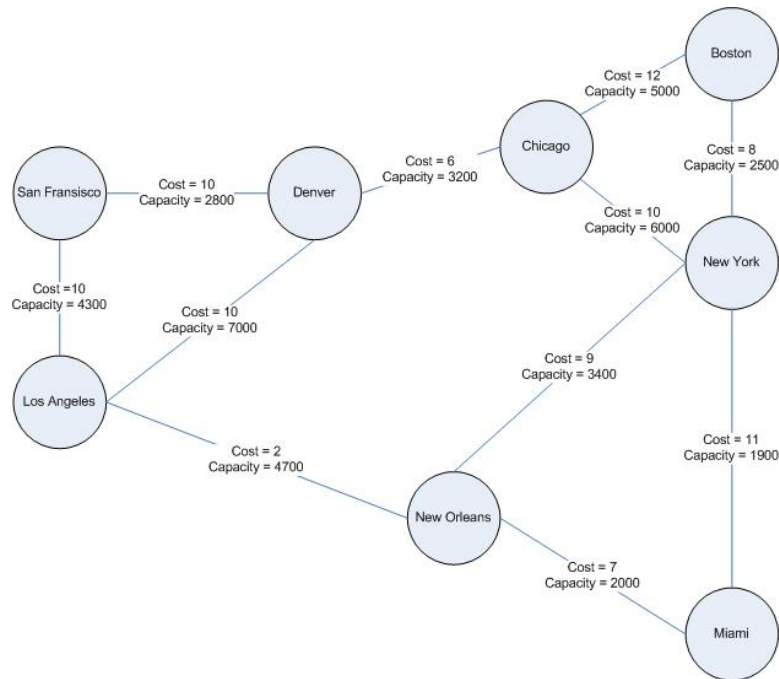


Figure 5.1: Companies Graph

auctioning agents (Denver and Chicago), two seller bidder agents (New York and San Francisco), and two buyer bidder agents (Los Angeles and New Orleans).

Boston seller auction's data

Capacity = 1500 watt

Minimum-contract amount = 300 watt

Price = 10 dollars per watt

Start delivery time = 8-2005

End delivery time = 12- 2006

Miami seller auction's data

Capacity = 1500 watt

Minimum-contract amount = 250 watt

Price = 10 dollars per watt

Start delivery time = 7-2005

End delivery time = 7- 2006

Denver buyer auction's data

Capacity = 3000 watt

Minimum-contract amount = 500 watt

Price = 120 dollars per watt

Start delivery time = 7-2005

End delivery time = 6- 2006

Chicago buyer auction's data

Capacity = 2000 watt

Minimum-contract amount = 300 watt

Price = 120 dollars per watt

Start delivery time = 8-2005

End delivery time = 5- 2006

New York seller bidder's data

Capacity = 1000 watt

Minimum-contract amount = 100 watt

Price = 12 dollars per watt

Start delivery time = 6-2005

End delivery time = 12- 2006

San Francisco seller bidder's data

Capacity = 1000 watt

Minimum-contract amount = 200 watt

Price = 8 dollars per watt

Start delivery time = 6-2005

End delivery time = 12- 2006

New Orleans buyer bidder's data

Capacity = 1000 watt

Minimum-contract amount = 100 watt

Price = 120 dollars per watt

Start delivery time = 10-2005

End delivery time = 10- 2006

Los Angeles buyer bidder's data

Capacity = 1000 watt

Minimum-contract amount = 12 watt

Price = 100 dollars per watt

Start delivery time = 10-2005

End delivery time = 12- 2006

5.2.1 Starting the agents

Each agent sends a message to the monitor agent with a SUBSCRIBE performative and attaches its data to the message. The agents wait for the registration confirmation from the monitor agent. After receiving the confirmation message, the buyer bidder agents send to the monitor agent a message with a QUERYREF performative to know the seller auctioning agents also, the seller bidder agents send to the monitor agent a message with a QUERYREF performative to know the buyer auctioning agents.

Each seller bidder agent checks for each buyer auctioning agents whether the auction amount is larger than the minimum bid amount, also the auction price is larger than the minimum price, if valid, the bidder agent sends a message to the TSO agent with a REQUEST performative to get the valid paths between its company and the auctioning agent's company, otherwise, the bidding agent ignores that auction. After receiving the paths, it checks whether the auction price is larger than the summation of transmission cost and target price, also the auctioning agent delivery interval lies inside the bidder agent delivery interval, if valid, the bidding agent stores the auctioning agent data, otherwise, it ignores that auction.

Each buyer bidder agent checks for each seller auctioning agent whether the auction amount is larger than the minimum bid amount, also the auction price is larger than the maximum price, if valid, the bidder agent sends a message to the TSO agent with a REQUEST performative to get the valid paths between its company and the auctioning agent's company, otherwise, the bidding agent ignores that auction. After receiving the paths, it checks whether the summation of auction price and the transmission cost is less than or equal target price, also the bidder agent delivery interval lies inside the auctioning agent delivery interval, if valid, the bidding agent stores the auctioning agent data, otherwise, it ignores that auction.

Each auctioning agent waits for bidders to send their bids for quarter and half an hour. After that time expires, it checks if there exists a collision, if exists, the auctioning agent tries to resolve it, otherwise, it sends a message with an ACCEPT-PROPOSAL performative to the bidder agent.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY77

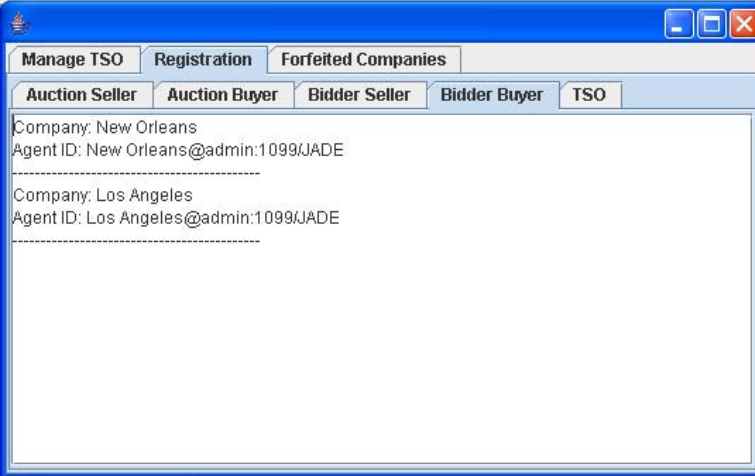
Company: Chicago
Agent ID: Chicago@ragab:1099/JADE
Amount: 2000.0
Maximum Price: 120.0

Company: Denver
Agent ID: Denver@ragab:1099/JADE
Amount: 3000.0
Maximum Price: 120.0

Company: Boston
Agent ID: Boston@ragab:1099/JADE
Amount: 1500.0
Maximum Price: 10.0

Company: Miami
Agent ID: Miami@ragab:1099/JADE
Amount: 1500.0
Maximum Price: 10.0

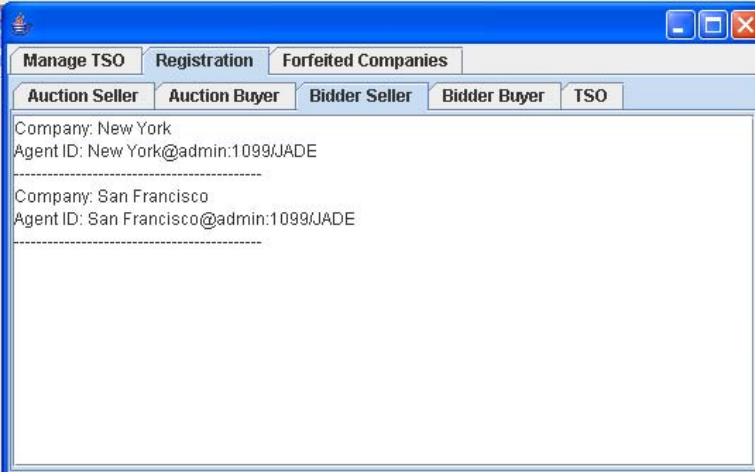
CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY78



The screenshot shows a window titled 'Manage TSO' with a blue border and standard Windows window controls. It has three tabs: 'Registration' (selected), 'Forfeited Companies', and 'Manage TSO'. Below the tabs is a sub-header with five buttons: 'Auction Seller', 'Auction Buyer', 'Bidder Seller', 'Bidder Buyer', and 'TSO'. The main area contains the following text:

Company: New Orleans
Agent ID: New Orleans@admin:1099/JADE

Company: Los Angeles
Agent ID: Los Angeles@admin:1099/JADE



The screenshot shows the same 'Manage TSO' window with the 'Registration' tab selected. The sub-header and main area content are as follows:

Company: New York
Agent ID: New York@admin:1099/JADE

Company: San Francisco
Agent ID: San Francisco@admin:1099/JADE

For New York agent:

It gets Denver and Chicago agents, both have the same price (120 dollars per watt) but Denver's delivery time is larger than Chicago's delivery time, so the simplex algorithm will return the modified bid value for Denver equals 1000 and exclude Chicago.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY80

bidder New York

Company Name

New York

Agent Name

New York

Duration

From: month

5

year

2005

To: month

11

year

2006

Min Contract Amount

100.0

Initial Capacity

1000.0

Current Capacity

0.0

Chicago@ragab.1099/JADE

Denver@ragab.1099/JADE

auction name: Denver@ragab.1099/JADE

auction capacity: 1000.0

auction min contract amount: 500.0

auction price: 120.0

send Propose amount: 1000.0 price: 120.0

receive ACCEPT amount: 1000.0 price: 120.0

send Confirm amount: 1000.0 price: 120.0

Auctions List

Chicago@ragab.1099/JADE

Denver@ragab.1099/JADE

For San Francisco agent:

It gets Denver and Chicago agents, both have the same price (120 dollars per watt) but Denver's delivery time is larger than Chicago's delivery time, so the simplex algorithm will return the modified bid value for Denver equals 1000 and exclude Chicago.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY82

bidder San Francisco

Company Name

San Francisco

Agent Name

San Francisco

Duration

From: month

5

year

2005

To: month

11

year

2006

Min Contract Amount

200.0

Initial Capacity

1000.0

Current Capacity

0.0

Chicago@ragab:1099/JADE

Denver@ragab:1099/JADE

auction name: Denver@ragab:1099/JADE

auction capacity: 2000.0

auction min contract amount: 500.0

auction price: 110.0

send Propose amount: 1000.0 price: 110.0

receive ACCEPT amount: 1000.0 price: 110.0

send Confirm amount: 1000.0 price: 110.0

Auctions List

Chicago@ragab:1099/JADE

Denver@ragab:1099/JADE

*CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY*⁸³

For New Orleans agent:

It gets Boston and Miami, but Miami's delivery time does not fit New Orleans delivery time so it will be excluded. New Orleans agent selects to bid at Boston with bid amount equals 1000.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY84

bidder New Orleans

Company Name

New Orleans

Agent Name

New Orleans

Duration

From: month

9

year

2005

To: month

9

year

2006

Min Contract Amount

100.0

Initial Capacity

1000.0

Current Capacity

500.0

Boston@ragab:1099/JADE

auCTION name: Boston@ragab:1099/JADE

auCTION capacity: 1500.0

auCTION min contract amount: 200.0

auCTION price: 10.0

send PROPOSE amount: 1000.0 price: 10.0

received PROPOSE amount: 500.0 price: 10.0

send ACCEPT new amount: 500.0 new price: 10.0

Auctions List

Boston@ragab:1099/JADE

*CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY*⁸⁵

For Los Angeles agent:

It gets Boston and Miami, but Miami's delivery time does not fit New Orleans delivery time so it will be excluded. Los Angeles agent selects to bid at Boston with bid amount equals 1000.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY86

bidder Los Angeles

Company Name

Los Angeles

Agent Name

Los Angeles

Duration

From: month 9 year 2005

To: month 10 year 2006

Min Contract Amount

100.0

Initial Capacity

1000.0

Current Capacity

0.0

Boston@ragab:1099/JADE

auction name: Boston@ragab:1099/JADE

auction capacity: 1000.0

auction min contract amount: 100.0

auction price: 10.0

send PROPOSE amount: 1000.0 price: 10.0

received PROPOSE amount: 1000.0 price: 10.0

send ACCEPT new amount: 1000.0 new price: 10.0

Auctions List

Boston@ragab:1099/JADE

For Boston Agent:

There exists a collision because within the intervals from 10-2005 to 10-2006 the summation of the bids equals 2000 watt which exceeds the auction amount (1500 watt). The auctioning agent uses the simplex algorithm to resolve the collision which returns the modified bids for New Orleans equals 500 watt, and for Los Angeles equals 1000 watt. The auctioning agent sends a message with a PROPOSE performative to the modified bids bidders and waits their acceptance, if accepts, the auctioning agent adds the accepted bidder to the winning bidders.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY88

Boston@ragab:1099/JADE

Auction's ID

Boston

Capacity

1500.0

Price

10.0

Minimum Price

5.0

Minimum Contract Amount

300.0

Bidders

New Orleans

Los Angeles

Message Information:

Message Performative: PROPOSE

auction amount 1500.0

auction price 10.0

auction minimum contract amount 300.0

bid amount 1000.0

Bidder New Orleans received message from Boston

Message Information:

Message Performative: PROPOSE

Message Content: the auction offer 500.0

modified amount 1000.0

price 10.0

Bidder New Orleans sent message to Boston

Message Information:

Message Performative: ACCEPT_PROPOSAL

Boston@ragab:1099/JADE

Auction's ID

Boston

Capacity

1500.0

Price

10.0

Minimum Price

5.0

Minimum Contract Amount

300.0

Bidders

New Orleans

Los Angeles

Bidder Los Angeles sent message to Boston

Message Information:

Message Performative: PROPOSE

auction amount 1500.0

auction price 10.0

auction minimum contract amount 300.0

bid amount 1000.0

Bidder Los Angeles received message from Boston

Message Information:

Message Performative: PROPOSE

Message Content: the auction offer 1000.0

modified amount 1000.0

price 10.0

Bidder Los Angeles sent message to Boston

Message Information:

Message Performative: ACCEPT_PROPOSAL

*CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY*89

For Miami Agent:

It did not receive any bids at this round.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY90

Miami@ragab:1099/JADE

Auction's ID

Miami

Capacity

1500.0

Price

10.0

Minimum Price

5.0

Minimum Contract Amount

250.0

Bidders

For Denver Agent:

No collision was happened, so it sends a message with an ACCEPT-PROPOSAL performative to the bidder agents and waits their confirmations. After receiving the confirmation, it adds the confirmed bidder to the winning bidders.

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY92

Denver@ragab:1099/JADE

Auction's ID

Denver

Capacity

3000.0

Price

120.0

Minimum Price

5.0

Minimum Contract Amount

500.0

Bidders

New York

San Francisco

Bidder New York sent message to Denver

Message Information:

Message Performative: PROPOSE

auction amount 3000.0

auction price 120.0

auction minimum contract amount 500.0

bid amount 1000.0

Bidder New York received message from Denver

Message Information:

Message Performative: ACCEPT_PROPOSAL

Message Content: Accept your bid

bid amount 1000.0

price 120.0

Bidder New York sent message to Denver

Message Information:

Message Performative: CONFIRM

Denver@ragab:1099/JADE

Auction's ID

Denver

Capacity

3000.0

Price

120.0

Minimum Price

5.0

Minimum Contract Amount

500.0

Bidders

New York

San Francisco

Bidder San Francisco sent message to Denver

Message Information:

Message Performative: PROPOSE

auction amount 3000.0

auction price 120.0

auction minimum contract amount 500.0

bid amount 1000.0

Bidder San Francisco received message from Denver

Message Information:

Message Performative: ACCEPT_PROPOSAL

Message Content: Accept your bid

bid amount 1000.0

price 120.0

Bidder San Francisco sent message to Denver

Message Information:

Message Performative: CONFIRM

*CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY*93

For Chicago agent:

It did not receive any bids at this round

CHAPTER 5. RUNNING THE ELECTRIC POWER MARKET AGENCY94

Chicago@ragab:1099/JADE

Auction's ID

Chicago

Capacity

2000.0

Price

120.0

Minimum Price

5.0

Minimum Contract Amount

300.0

Bidders

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Agents are quite suited to implement electronic marketplaces and electronic auctions. We have used agents in our system to act on behalf of companies to perform the task of buying and selling electric power. The problems of choosing the auctions to bid in, as a bidding agent, and of choosing the bids to accept when collisions occur, as an auctioning agent, are much harder in electric power markets than in more typical markets. Therefore our system can easily be extended with little effort to serve a variety of market goods.

We have used the JADE platform for implementing our Agent-based Bidding System. JADE has proven to be a good tool for implementing agents, though the documentation that comes with the packages can be better. We have used XML files to store our data (e.g. the messages exchanged and the winning bids). This has also proven to be a good choice since the size of the data is not that large so the overhead of a database management system would be unacceptable.

The agents interaction protocol that we designed and later built on, turned out to be very good. We came to this conclusion because very minor changes were made to the original design while in the implementation stage.

Through our system we have seen how individual agents can adopt the goals and intentions of its stakeholder. The stakeholder can directly make critical decisions and pass them to the task-doing agent. We conclude our project saying that the distributed multi agents model is significantly more useful and practical than other models (e.g. the central server model).

6.2 Future Work

- As a future work, a few more features could be added to the system:
- Extend the system to support a variety of market goods.
- Implement more than one Monitor agent in order to have redundancy that increases system reliability.
- Extend the system to support wireless services and wireless devices (e.g. PDA).
- Implement a variety of auction formats and analyze them.
- Implement artificial intelligence techniques to get the market expected price, the amount of goods available for purchase and sell

Appendix A

Java Agent DEvelopment Framework (JADE)

This appendix discusses the java agent development environment. This appendix illustrates the jade platform, the key technologies jade is built on, and how to use jade along with some advanced features in jade and developing multi-agent systems.

A.1 JADE Overview

JADE, as mentioned in section 4.1, is an enabling technology, a middleware for the development and run-time execution of peer-to-peer applications which are based on the agents paradigm and which can seamless work and interoperate both in wired and wireless environment.

Inspired by the vision that agents will remain just a dream if end-to-end interoperability across different manufacturers and operators is not preserved in 1996 TILAB (formerly CSELT) promoted the creation of FIPA (Foundation for Intelligent Physical Agents), an international non-profit association of companies and organizations sharing the goal and the effort to produce standard specifications for agent technology. TILAB, and in particular the JADE team, supported and leaded at several levels this initiative continuing with the editing of specifications and the leadership of Technical Committees and of the FIPA Architecture Board JADE also participated with success to both FIPA Interoperability Tests, in 1999 and in 2001. that is why JADE is assumed to be a middleware, since it provide higher-level libraries that enable easier and more effective application development by providing generic

services useful not just for a single application but rather for a variety of applications. For instance data access, communication, encodings, resource control. These same services are provided by operative systems, but the idea behind middleware is to provide better, OS independent APIs aggregating native facilities into simple-to-reuse building blocks. Middleware based approaches allows to reduce footprint and development time of applications. The capability of reusability across several application domains suggests the name of 'horizontal' approach as opposed to 'vertical' approach where an ad-hoc solution for a specific application should be provided. A figure showing the middleware location in the software development process and a comparison between the horizontal and the vertical approach is shown in figure:A.1

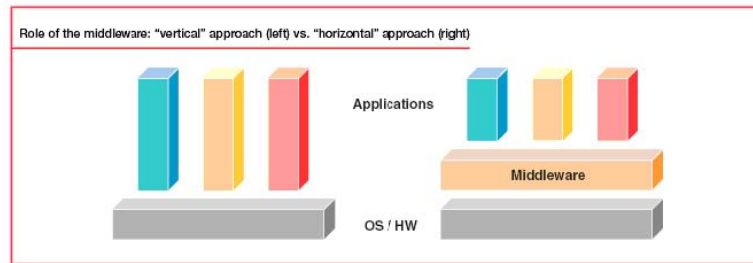


Figure A.1: Role of the middleware: "vertical" approach (left) vs."horizontal" approach (right)

JADE has been implemented fully in Java language and it can be seamlessly executed on every type of Java Virtual Machine with exception of the Java Card. JADE is extremely versatile and therefore, not only it fits the constraints of environments with limited resources, but it has already been integrated into complex architectures such as.NET or J2EE where JADE becomes service to execute multi-party proactive applications. The limited memory footprint allows installing JADE on all mobile phones provided that they are Java-enabled. The JADE run-time can be executed on a wide class of devices ranging from servers to cell phones, for the latter the only requirement being Java MIDP1.0 (or higher versions). JADE architecture is illustrated in figure:A.2.

A.2 JADE Programming

JADE includes both the libraries (i.e. the Java classes) required to develop application agents and the run-time environment that provides the basic services and that must be active on the device before agents can be executed.

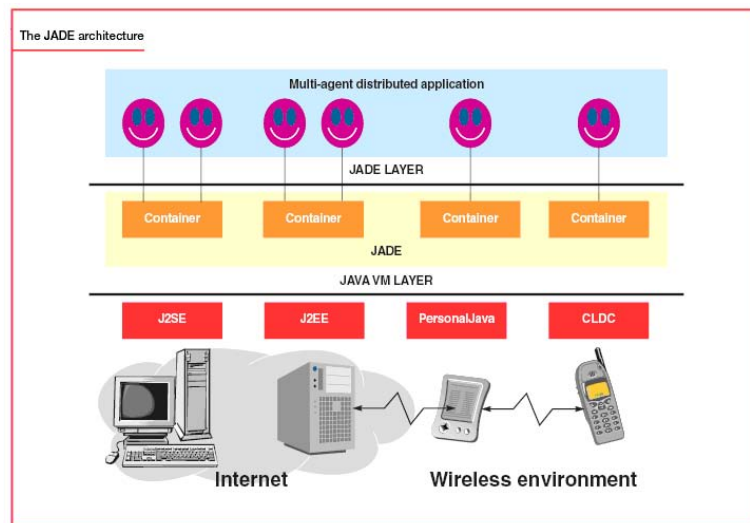


Figure A.2: The JADE Architecture

Each instance of the JADE run-time is called a container (since it "contains" agents). The set of all containers is called a platform and provides a homogeneous layer that hides from agents (and from application developers also) the complexity and the diversity of the underlying tier (hardware, operating systems, types of network, JVM).

Creating a JADE agent requires extending the class `jade.core.agent` and implementing the `setup` method.

Example:

```
import jade.core.Agent; public class myFirstAgent extends Agent {
protected void setup() { // Printout a welcome message
System.out.println("Hallo! My-Agent-No1 " + getAID().getName() + " is ready."); }
. Each agent has an "agent identifier" which is represented as an instance of jade.core.AID. To run the agent in the previous example, we first compile it as follows:
```

```
javac -classpath < JADE - classes > myFirstAgent.java
```

and type the following also in the command line:

```
java -classpath < JADE - classes > ; jade.Boot HelloAgent:myFirstAgent
```

To terminate an agent the method `doDelete()` must be called, when an

agent is trying to terminate the JADE framework invokes the function `takeDown()` which includes agent clean-up operations.

Since agents are considered autonomous entities, their rules are carried out within a behaviour class. A behaviour represents a task that an agent can carry out and is implemented as an object of a class that extends `jade.core.behaviours.Behaviour`. There are multiple types of behaviours which provide flexibility to the developer to choose between them according to agent characteristics.

To make an agent execute a specific behaviour the method `addBehaviour()` is used in the agent class or in the initialization process, i.e. in the `setup()` method. Each behavior is executed whenever its `action()` method is implemented, and it stops execution only if the `done()` method of the behaviour returns true. Scheduling of behaviours can be done while taking into account that it is not pre-emptive but cooperative, since each agent is represented as a single thread. This single thread eases the implementation of agents, eliminates synchronization matters and increases performance because behaviour switching is faster than Java thread switching. The path of execution for agent thread is depicted in figureA.3.

JADE implements FIPA-ACL for agent communication. The adopted communication paradigm is the asynchronous message passing. Each agent has a sort of mailbox (the agent message queue) where the JADE runtime posts messages sent by other agents. Whenever a message is posted in the message queue, the receiving agent is notified. However, if and when the agent actually picks up the message from the message queue to process it is completely up to the programmer.

A number of fields in the ACL message are like: sender, list of receivers, communication intention (performative), content. Filling in these fields is the way to prepare a message to other agents. To send a message to another agent we just prepare the message (i.e. prepare the message fields with the appropriate data), then we simply call the `send()` method with the message as the argument for this method. Receiving a message is very simple. The following code fragment for message receiving is a little professional since it blocks the agent thread until a message comes to the queue, instead of a busy wait state waiting for a message, this is more effective and increases performance as well. A performative, `blockingReceive`, is available which does exactly as this code. The following code is the typical pattern for receiving messages inside a behaviour.

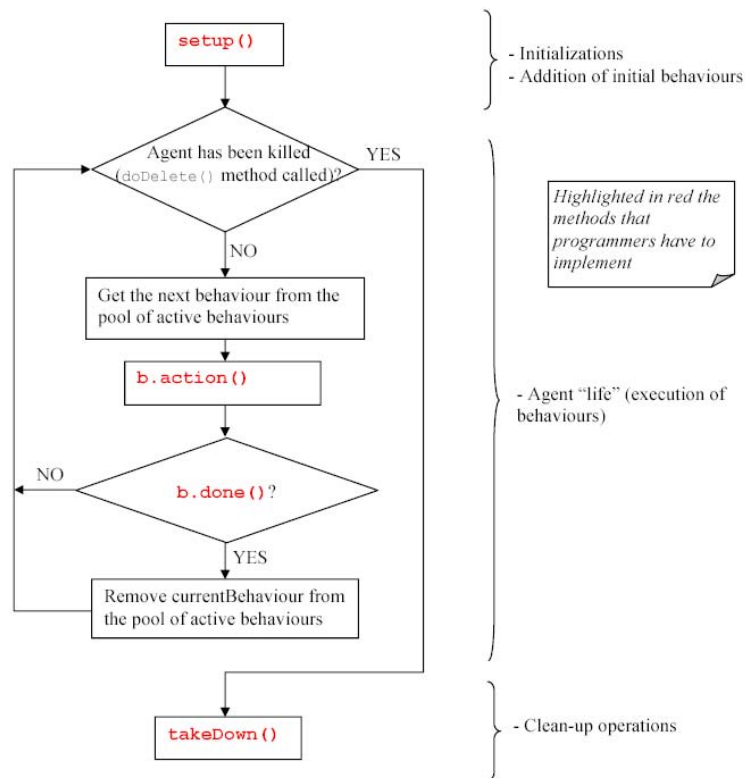


Figure A.3: Agent Thread path of execution

```
public void action()
```

```
    ACLMessage msg = myAgent.receive(); if (msg != null) // Message
received. Process it ... else block();
```

A.3 The Yellow Pages Service:

In the last sections we assumed that we know the name and location of the agent we want to talk to. But this is not the case in practical. In this section we describe how to get rid of this assumption and exploit the yellow pages service provided by the JADE platform. Like this any agent can discover dynamically other agents available at a given point in time.

A "yellow pages" service allows agents to publish one or more services they provide so that other agents can find and successively exploit them, as shown in figure A.4. The yellow pages service in JADE (according to the

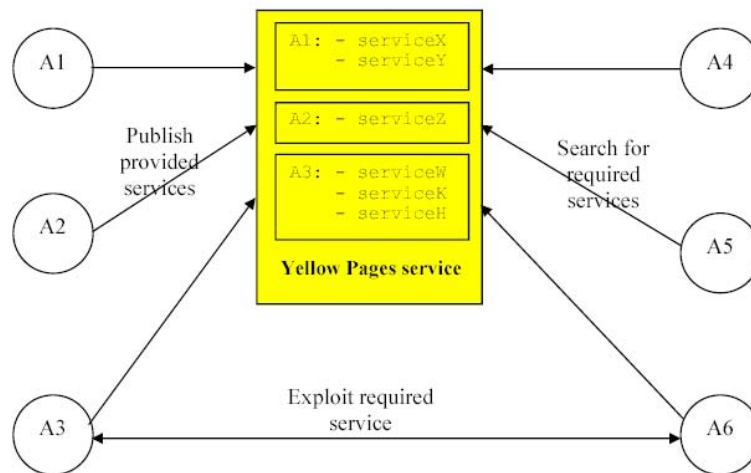


Figure A.4: The Yellow Pages service

FIPA specification) is provided by an agent called Directory Facilitator (DF). Each FIPA compliant platform hosts a default DF agent (whose local name is "df"). Other DF agents can be activated and several DF agents (including the default one) can be federated so that to provide a single distributed yellow pages catalogue. We can interact with the DF agent in two ways, either

by registering a service in the DF, or by searching for a service in it.

A.4 Why JADE

JADE has been recently made available under Open Source License, and that is a great motivation for using JADE. Other reasons for using JADE are as follows:

- JADE eases development of **distributed applications composed of autonomous entities**. By applying these properties to mobile phones and smart devices, JADE ignites a new trend of evolution could be names: smart-device smart-interconnection. Here hand-held devices are equipped with autonomy, intelligence not as previous trends of technology where the power of the system was relying on the content of the system and the capability of accessing this content from anywhere.
- **Pro-activity**: JADE agents control their own thread of execution and, therefore, they can be easily programmed to initiate the execution of actions without human intervention just on the basis of a goal and state changes. This feature, that is usually called pro-activity, makes JADE a suitable environment for the realization of machine-to-machine (m2m) applications, for example, for industrial plant automation, traffic control and communication network management.
- **Multi-party applications**: Peer-to-peer architectures are more efficient than client-server architectures for developing multi-party applications, as the server might become the bottleneck and the point of failure of the entire system. Because JADE agents can both provide and consume services, they remove any need to distinguish between clients and servers.
- **Interoperability**.
- **Openness**: JADE is an open-source project that involve the contributions and collaborations of the user community. This user-driven approach allows both users and developers to contribute with suggestions and new code, which guarantees openness and usefulness of the APIs.

- ***Versatility:*** JADE provides a homogeneous set of APIs that are independent from the underlying network and Java version. It in fact provides the same APIs both for the J2EE, J2SE and J2ME environment. This, of course, enables us easily to extend our system to work on hand-held devices with nearly no recognized effort.

Appendix B

FIPA ACL Message Structure Specification

B.1 FIPA ACL Message Structure

A FIPA ACL message contains a set of one or more message parameters. Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is the performative, although it is expected that most ACL messages will also contain sender, receiver and content parameters.

If an agent does not recognize or is unable to process one or more of the parameters or parameter values, it can reply with the appropriate not-understood message.

Specific implementations are free to include user-defined message parameters other than the FIPA ACL message parameters specified in Table 1. The semantics of these user-defined parameters is not defined by FIPA, and FIPA compliance does not require any particular interpretation of these parameters. The prefatory string "X-" must be used for the names of these non-FIPA standard additional parameters.

Some parameters of the message might be omitted when their value can be deduced by the context of the conversation. However, FIPA does not specify any mechanism to handle such conditions, therefore those implementations that omit some message parameters are not guaranteed to interoperate with each other.

The full set of FIPA ACL message parameters is shown in Table 1 without regard to their specific encodings in an implementation. FIPA-approved encodings and parameter orderings for ACL messages are given in other specifications. Each ACL message representation specification contains precise syntax descriptions for ACL message encodings based on XML, text strings and several other schemes.

A FIPA ACL message corresponds to the abstract parameter message payload identified in the [FIPA00001].

Parameter	Category of Parameters
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
Protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

Table B.1: FIPA ACL Message Parameters

The following terms are used to define the ontology and the abstract syntax of the FIPA ACL message structure://

- **Frame:** This is the mandatory name of this entity that must be used to represent each instance of this class.
- **Ontology:** This is the name of the ontology, whose domain of discourse includes their parameters described in the table.
- **Parameter:** This identifies each component within the frame. The type of the parameter is defined relative to a particular encoding. Encoding specifications for ACL messages are given in their respective specifications.

- **Description:** This is a natural language description of the semantics of each parameter. Notes are included to clarify typical usage.
- **Reserved Values:** This is a list of FIPA-defined constants associated with each parameter. This list is typically defined in the specification referenced.

All of the FIPA message parameters share the frame and ontology shown below

Frame	fipa-acl-message
Ontology	fipa-acl

Table B.2: FIPA ACL Message Frame and Ontology

B.2 Type of Communicative Act

B.2.1 Performative

Parameter	Description	Reserved Values
performative	Denotes the type of the communicative act of the ACL message	See [FIPA00037]

Table B.3: FIPA ACL Message Performative

Notes: The performative parameter is a required parameter of all ACL messages. Developers are encouraged to use the FIPA standard performatives (see [FIPA00037]) whenever possible.

B.3 Participants in Communication

B.3.1 Sender

Notes: The sender parameter will be a parameter of most ACL messages. It is possible to omit the sender parameter if, for example, the agent sending the ACL message wishes to remain anonymous. The sender parameter refers to the agent which performs the communicative act giving rise to this ACL

Parameter	Description
sender	Denotes the identity of the sender of the message, that is, the name of the agent of the communicative act.

Table B.4: FIPA ACL Message Sender

message.

B.3.2 Receiver

Parameter	Description
Receiver	Denotes the identity of the intended recipients of the message.

Table B.5: FIPA ACL Message Receiver

Notes: Ordinarily, the receiver parameter will be a part of every ACL message. It is only permissible to omit the receiver parameter if the message recipient can be reliably inferred from context, or in special cases such as the embedded ACL message in proxy and propagate.

The receiver parameter may be a single agent name or a non-empty set of agent names. The latter corresponds to the situation where the message is multicast. Pragmatically, the semantics of this multicast is that the sender intends the message for each recipient of the CA encoded in the message. For example, if an agent performs an inform act with a set of three agents as receiver, it denotes that the sender intends each of these agents to come to believe the content of the message.

Parameter	Description
reply-to	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.

Table B.6: FIPA ACL Message Reply-To

Parameter	Description
content	Denotes the content of the message; equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver.

Table B.7: FIPA ACL Message content

B.3.3 Reply To

B.4 Content of Message

B.4.1 Content

Notes: Most ACL messages require a content expression. Certain ACL message types, such as cancel, have an implicit content, especially in cases of using the conversation-id or in-reply-to parameters.

B.5 Description of Content

B.5.1 Language

Parameter	Description
Language	Denotes the language in which the content parameter is expressed.

Table B.8: FIPA ACL Message Language

Notes: The ACL content parameter is expressed in a formal language. This field may be omitted if the agent receiving the message can be assumed to know the language of the content expression.

B.5.2 Encoding

Parameter	Description
Encoding	Denotes the specific encoding of the content language expression.

Table B.9: FIPA ACL Message Encoding

Notes: The content expression might be encoded in several ways. The encoding parameter is optionally used to specify this encoding to the recipient agent. If the encoding parameter is not present, the encoding will be specified in the message envelope that encloses the ACL message.

B.5.3 Ontology

Parameter	Description
ontology	Denotes the ontology(s) used to give a meaning to the symbols in the content expression

Table B.10: FIPA ACL Message Ontology

Notes: The ontology parameter is used in conjunction with the language parameter to support the interpretation of the content expression by the receiving agent. In many situations, the ontology parameter will be commonly understood by the agent community and so this message parameter may be omitted.

B.6 Control of Conversation

B.6.1 Protocol

Parameter	Description
Protocol	Denotes the interaction protocol that the sending agent is employing with this ACL message.

Table B.11: FIPA ACL Message Protocol

Notes: The protocol parameter defines the interaction protocol in which the ACL message is generated. This parameter is optional; however, developers are advised that employing ACL without the framework of an interaction protocol (and thus directly using the ACL semantics to control the agent's generation and interpretation of ACL messages) is an extremely ambitious undertaking.

Any ACL message that contains a non-null value for the protocol parameter is considered to belong to a conversation and it is required to respect the following rules:

- the initiator of the protocol must assign a non-null value to the conversation-id parameter,
- all responses to the message, within the scope of the same interaction protocol, should contain the same value for the conversation-id parameter, and,
- the timeout value in the reply-by parameter must denote the latest time by which the sending agent would like to have received the next message in the protocol flow (not be confused with the latest time by which the interaction protocol should terminate).

B.6.2 Conversation Identifier

Parameter	Description
Conversation-Id	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.

Table B.12: FIPA ACL Message Conversation Identifier

Notes: An agent may tag ACL messages with a conversation identifier to manage its communication strategies and activities. Typically this will allow an agent to identify individual conversations with multiple agents. It will also allow agents to reason across historical records of conversations.

It is required the usage of globally unique values for the conversation-id parameter in order to allow the participants to distinguish between several concurrent conversations. A simple mechanism to ensure uniqueness is the concatenation of the globally unique identifier of the sender agent to an identifier (for example, a progressive number) that is unique within the scope of the sender agent itself

Appendix C

The Simplex Method

C.1 Introduction

A main feature of the simplex method is that it solves the LP in iterations. Each iteration moves the solution to a new corner point that has the potential to improve the value of the objective function. The process ends when no further improvements can be realized.//

C.2 LP solution space in equation form

For the sake of standardization, the algebraic representation of the LP solution space is made under two conditions.

1. All the constraints (with the exception of the nonnegative restrictions) are equations with a nonnegative right-hand side.
2. All variable are nonnegative.

C.3 Converting Inequality into equations

In (\leq) constraints, the right hand side can be thought of the representing the limit on the availability of a resource, in which case the left hand side would represent the usage of this limited resources by activity (variables) of the model. The difference between the right-hand side and the left-hand side of the (\leq) constraint thus the unused or slack amount of the resource.

To convert a (\leq) inequality to an equation, a nonnegative slack variable is added to the left-hand side of the constraint. For example, if the constraint

is given as

$6x_1 + 6x_2 \leq 24$ Defining s_1 as the slack or unused amount, the constraint can be converted to the following equation

$$6x_1 + 6x_2 + s_1 = 24, s_1 \geq 0$$

C.4 Computational Details of the Simplex Algorithm

This section provides the computational details of the simplex iteration that include the rules for determining the entering and leaving variables as well as for the stopping computations when the optimum solution has been reached. The vehicle of explanation is a numerical example.

Example

Maximize $z = 5x_1 + 4x_2 + 0s_1 + 0s_2 + 0s_3 + 0s_4$

Subject to:

$$6x_1 + 6x_2 + s_1 = 24$$

$$x_1 + 2x_2 + s_1 = 6$$

$$-x_1 + x_2 + s_3 = 1$$

$$x_2 + s_4 = 2$$

$$x_1, x_2, s_1, s_2, s_3, s_4 \geq 0$$

The variables s_1, s_2, s_3 and s_4 are the slacks associated with the respective constraints. Next, we express the objective equation as
 $z - 5x_1 - 4x_2 = 0$

Basic	z	x_1	x_2	s_1	s_2	s_3	s_4	solution	
z	1	-5	-4	0	0	0	0	0	z-row
s_1	0	6	4	1	0	0	0	24	s_1 -row
s_2	0	1	2	0	1	0	0	6	s_2 -row
s_3	0	-1	1	0	0	1	0	1	s_3 -row
s_4	0	0	1	0	0	0	1	2	s_4 -row

The design of the tableau specifies the set of the basic and non basic variables as well as provides the solution associated with the starting iteration. The simplex iteration start at the origin $(x_1, x_2) = (0, 0)$. Thus, the associated set of non basic and basic are defined as

Non-basic (zero) variables: (x_1, x_2)

Basic variables: (s_1, s_2, s_3, s_4)

Given the non-basic variables $(x_1, x_2) = (0, 0)$ and noting that the special 0-1 arrangement of the coefficients of the basic variables (s_1, s_2, s_3, s_4) in the tableau, the following solution is immediately available (with out further computations).

$$\begin{aligned} z &= 0 \\ s_1 &= 24 \\ s_2 &= 6 \\ s_3 &= 1 \\ s_4 &= 2 \end{aligned}$$

This information is shown in the tableau by listing the basic variables in the left-most Basic-column and their values in the right-most Solution-column. In effect, the tableau defines the current corner point by specifying its basic variables and their values, as well as the corresponding value of the objective function, z . Remember that the non-basic variables (those not listed in the Basic-column) always equal zero.

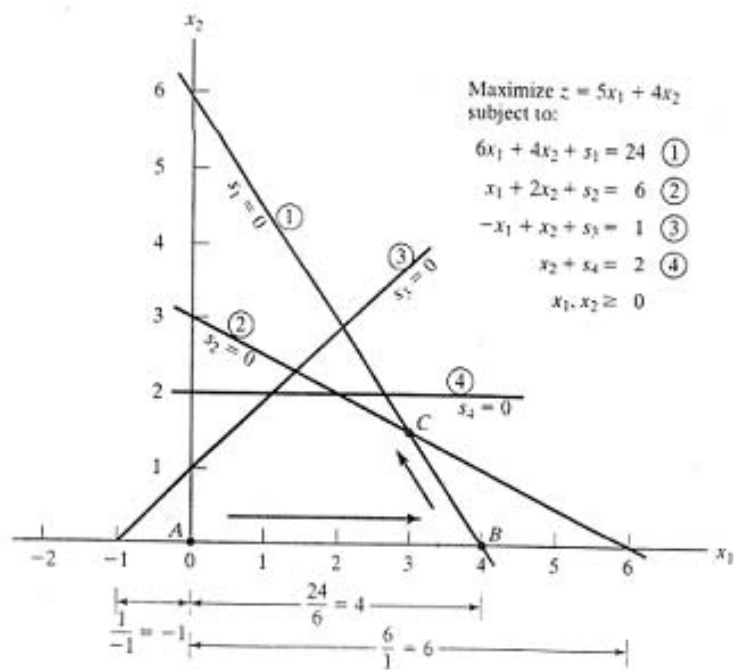
Is the starting solution optimal? The non-basic variable with the most positive coefficient in the maximization objective function is selected to enter the basic solution. This rule is based on expressing the objective function as $z = 5x_1 + 4x_2$. Because the simplex tableau expresses the objective function as $z - 5x_1 - 4x_2 = 0$, the entering variable is x_1 because it has the most negative coefficient in the objective function. If all the objective coefficients happen to be ≥ 0 , no further improvement in z is possible, signifying that the optimum has been reached.

To determine the leaving variable directly from the tableau, we compute the intercepts of all constraints with the non-negative direction of the x_1 -axis (recall that x_1 is the entering variable). The intercepts are the ratios of the right-hand side of the equations (Solution column) to the corresponding constraint coefficients under the entering variable, x_1 , as the following table shows.

As the following figure shows, the non-negative ratios equal the intercepts in the increasing direction of x_1 . The ratios (intercepts) corresponding to s_3 and s_4 are ignored because they do not limit x_1 in the negative direction.

The minimum nonnegative ratio corresponds to basic s_1 , signifying that s_1 is the leaving variable (its value is zero at the next iteration). The value

Basic	Entering x_1	Solution	Ratio (or intercept)
s_1	6	24	$x_1 = 24/6 = 4 \leftarrow \text{minimum}$
s_2	1	6	$x_1 = 6/1 = 6$
s_3	-1	1	$x_1 = 1/-1 = -1$ (ignore)
s_4	0	2	$x_1 = 2/0 = ?$ (ignore)



of the entering variable x_1 in the new solution also equals the minimum ratio.

The end result of "swapping" the entering and the leaving variable is that the non-basic and basic variables at the new solution point (point B) will be given as

Non-basic (zero) variables: (s_1, x_2)

Basic variables: (x_1, s_2, s_3, s_4)

We now need to manipulate the equations in the last tableau so that the Basic-column and the Solution-column will identify the new solution at point B. The process, called the **Gauss-Jordan row operations**.

The following is the replica of the starting tableau. It associates the **pivot column** and the **pivot row** with the entering and leaving variables respectively. The intersection of the pivot column and the pivot row is called the **pivot element**.

Basic	z	x_1	x_2	s_1	s_2	s_3	s_4	solution	
z	1	-5	-4	0	0	0	0	0	
s_1	0	6	4	1	0	0	0	24	Pivot row
s_2	0	1	2	0	1	0	0	6	
s_3	0	-1	1	0	0	1	0	1	
s_4	0	0	1	0	0	0	1	2	
		Pivot column							

The Gauss-Jordan computation needed to produce the new basic solution include two types.

1. Pivot row New pivot row = Current pivot row / pivot element
2. All other rows, including z New Row = (Current row) - (Its pivot column coefficient) (New pivot row)

These computations are applied to the preceding tableau in the following manner:

1. New pivot $s_1 - row = current s_1 - row / 6$
2. New z-row = current z-row - (-5) new pivot row
3. New $x_1 - row = current x_1 - row - (1) \text{ new pivot row}$

4. New $s_3 - \text{row} = \text{current } s_3 - \text{row} - (-1)$ new pivot row

5. New $s_4 - \text{row} = \text{current } s_4 - \text{row} - (0)$ new pivot row

The new tableau corresponding to the new basic solution (x_1, s_2, s_3, s_4) thus becomes (verify!)

Basic	z	x_1	x_2	s_1	s_2	s_3	s_4	solution
z	1	0	-2/3	5/6	0	0	0	20
x_1	0	1	2/3	1/6	0	0	0	4
s_2	0	0	4/3	-1/6	1	0	0	2
s_3	0	0	5/3	1/6	0	1	0	5
s_4	0	0	1	0	0	0	1	2

Pivot
column

Pivot row

By repeating the previous method the final (optimal) tableau will be
 You can verify that the values $s_1 = s_2 = 0, s_3 = 5/2, s_4 = 1/2$ are consistent with the given values of

Basic	z	x_1	x_2	s_1	s_2	s_3	s_4	solution
z	1	0	0	3/4	1/2	0	0	21
x_1	0	1	0	1/4	-1/2	0	0	3
x_2	0	0	1	-1/8	3/4	0	0	3/2
s_3	0	0	0	3/8	-5/4	1	0	5/2
s_4	0	0	0	1/8	-3/4	0	1	1/2

The rule for selecting the entering and leaving variables are referred to as the **optimality** and **feasibility conditions**. For convenience, these conditions and the steps of the simplex method are summarized below.

Optimality Condition. The entering variable in the maximization problem is the non-basic variable having the most negative coefficient in the z-row. Ties are broken arbitrary. The optimum is reached at the iteration where all the z-row coefficients of the non-basic variables are nonnegative.

Feasibility Condition. For maximization problem, the leaving variable is the basic variable associated with the smallest nonnegative ratio (with strictly positive denominator). Ties are broken arbitrary.

The steps of the simplex method are

- Step 0. Determine the starting basic feasible solution
 Step 1. Select an entering variable using the optimality condition. Stop if

there is no entering variable; last solution is optimal.

Step 2. Select a leaving variable using the feasibility condition.

Step 3. Determine the new basic solution by using appropriate Gauss-Jordan computations. Go to step 1.

Bibliography

- [1] <http://www.fipa.org/>
- [2] <http://www.auml.org/>
- [3] <http://www.jade.tilab.com/>
- [4] <http://www.epri.com/>
- [5] <http://www.agentcities.org/>
- [6] <http://logging.apache.org/log4j/docs/>
- [7] Hamdy A.Taha, Operation Research, seventh edition, Prentice Hall, 2000.