CRATI: A COMBINATORIAL REVERSE AUCTION TRADE INFRASTRUCTURE

A MASTER'S THESIS

in

Computer Engineering

Atılım University

by

HAKAN BAYINDIR

July 2010

CRATI: A COMBINATORIAL REVERSE AUCTION TRADE INFRASTRUCTURE

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

HAKAN BAYINDIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JULY 2010

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

(Title and Name)

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

(Title and Name)

Head of Department

This is to certify that we have read the thesis "Thesis Name" submitted by "Candidates Name" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____ _____

(Title and Name) (Title and Name)

Co-Supervisor Supervisor

Examining Committee Members

..... _____

..... _____

..... _____

..... _____

..... _____

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: HAKAN BAYINDIR

Signature:

ABSTRACT

CRATI: A COMBINATORIAL REVERSE

AUCTION TRADING INFRASTRUCTURE

Bayındır, Hakan

M.S, Computer Engineering Department

Supervisor: Assist. Prof. Dr. Hürevren Kılıç

July 2010, 67 Pages

With quick evolution of the Internet, e-trading is on the rise for some time and auctions are one of the fastest growing part in this trend. The English auction which is used in real life is also the most popular auction type on the Internet and is seller oriented instead of buyer resulting in profiting sellers instead of enabling buyers to obtain a good with a better price. This thesis addresses this issue by designing, implementing, and presenting an agent-based, automated, combinatorial reverse auction platform, which can be used by many markets with different needs, and enables buyers to obtain goods at instant lowest price possible.

Keywords: Agent-oriented programming, combinatorial reverse auction, business to customer (B2C) e-trading, weighted set covering, multi-agent systems.

ÖZ

CRATI: KOMBİNE TERS İHALE TİCARET SİSTEMİ ALTYAPISI

Bayındır, Hakan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Hürevren Kılıç

Temmuz 2010, 67 Sayfa

Günümüzde hızla evrilen internet teknolojileri elektronik ticaretin yaygınlaşmasını da beraberinde getirdi. Açık arttırma ile e-ticaret ise internette en yüksek hızda yayılan sistemlerden bir tanesi. Günlük hayatta da yaygın bir şekilde kullanılan İngiliz tipi açık arttırmalar internette de aynı oranda rağbet görüyorlar fakat; bu tip açık arttırmalar satıcı odaklı olduklarından dolayı alıcıya ekonomik bir alışveriş yolu sağlamaktan daha çok satıcının mümkün olduğunca kar etmesine yardımcı olmakta. Bu tezde, erkin temelli, tamamen otomatik bir kombine ters ihale ticaret sistemi altyapısı tasarlanmış, geliştirilmiş ve anlatılmıştır. Farklı piyasaların ihtiyaçlarına cevap verebilecek olan bu sistem, alıcının istediği ürünleri mümkün olan anlık en ucuz fiyata almasına olanak sağlamaktadır.

Anahtar kelimeler: Erkin yönelimli programlama, kombine ters ihale, firmadan müşteriye e-ticaret, ağırlıklı küme kapsama problemi, çok erkinli sistemler.

To My Parents, My Grandfather
&
To My Teachers Who Taught Me the Art of Computer Engineering

ACKNOWLEDGMENTS

I want to deeply and sincerely thank my thesis advisor Hürevren Kılıç not just for being such a wonderful motivator and mentor but also for teaching all the courses which have made me a better computer engineer. Without his help and advices I would never complete this work successfully and on time. Also I want to sincerely thank to Çiğdem Turhan and Fügen Selbes who not only gave wonderful lectures but great insights of life. Last but not least my great thanks go to my parents who supported me not only during this period but my whole life and to my grandfather who bought me my first personal computer and completely changed my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

- **JADE:** Java Agent DEvelopment Framework
- **FIPA:** Foundation for Intelligent Physical Agents
- **ACL:** Agent Communication Language
- **B2C:** Business to Consumer
- **C2C:** Consumer to Consumer
- **B2B2C:** Business to Business to Consumer
- **IEEE:** Institute of Electrical and Electronics Engineers
- **WSCP:** Weighted Set Covering Problem
- **VCS:** Version Control System

CHAPTER 1

Introduction

Auction is one of the oldest trading mechanisms in the world, with a history going back to old Roman Empire. The basic aim of the auction is to sell an item to the right person by identifying the person who wants the item most and trying the maximize the profit of the seller during the process [1].

While auction tries to serve a simple purpose, it has many variants and all these variants have different properties and different effects on the market. Two of these variants are reverse auctions and combinatorial auctions. Reverse auction swaps the sides of the seller and the buyer, while combinatorial auction enables selling of multiple items to the right people at the same time by allowing bidders to bid on items they are most interested in. While ingenious and useful, combinatorial auctions brings the important problem of winner determination, which is an NP-complete problem (that is, polynomial time algorithm which guarantees to compute the optimal allocation is unlikely to exist unless $P = NP$).

In today's economy of fast trading and limited money flow, reverse auctions are widely used in many markets like outsourcing, e-procurement, big volume or high valued good purchasing [2, 3, 4, 5]. Similarly, combinatorial auctions in traditional or reverse format are highly used in government contract auctions and in markets where a limited supply of intangible but related goods are sold and getting them in combinations is in best interest of both seller and the buyer such as bus transportation routes, logistics, airport runway time slots, nation wide food supplying contracts and similar [4, 6, 7]. As seen on Chilean food supplying contracts [4], reverse combinatorial auctions can successfully fulfill the requirements of the today's economy of efficient and fast trading with minimum money however, the hardness

of the winner determination problem is one of the biggest factor that prevents widespread adoption of this auction mechanism. Recently, with the evolution of both software engineering and the Internet, computer software is evolving towards a more connected and user friendly form and this simplification process converts some of the most complicated tools to consumer usable and convenient tools. One of these examples is the eBay [8] which can be worded as the biggest consumer oriented auction house powered by the consumers themselves. In eBay, users can bid to auctions or sell the goods they have in a classical single item English auction format. Again, with the help of the aforementioned evolution and the advances in processor technology which results in processors with higher computing capacities every year, computationally hard problems like the winner determination problem of meaningful sizes become possible to solve in meaningful time frames. As a result, this enables us to conduct combinatorial reverse auctions with bigger item sets and more merchants.

In this thesis, the aforementioned problem of winner determination in combinatorial reverse auctions is solved with a problem model based on weighted set covering problem and a trading infrastructure built on this foundation that allows conduction of reverse combinatorial auctions with the initiation of the buyer is proposed. The proposed infrastructure, called CRATI (A Combinatorial Reverse Auction Trading Infrastructure), is a multi-agent based, scalable, and real time infrastructure which, can conduct a completely automated, real-time combinatorial reverse auction on the behalf of the buyer, with the request from the buyer. Proposed infrastructure impersonates everyone in the system as agents and every step including item list generation, bidding, winner determination, and notification of related parties are completed without any human intervention. The motivation behind the work is the exploitation of the recent advancements in hardware and software technology with the aim of a useful, novel and functional real-world combinatorial reverse auction infrastructure based on the conversion of winner determination problem to weighted set covering problem which can be used by consumers and businesses alike with tangible benefits like cost reduction and efficient trading with faster resolutions which are both important for businesses and consumers alike.

The remaining part of this thesis is organized as follows. In Section 2, a brief literature survey with the current applications of various auction types are given

alongside their real-world applications. In Section 3, required background information together with an in-depth explanation of the software technology and the winner determination problem alongside the tools used to tackle winner determination problem is given. In section 4, details of the CRATI infrastructure complete with the development methodology, architecture, implemented trade protocol with step by step explanation, other challenges encountered during development, and the benchmarking results for the implemented infrastructure are given. The last section of the thesis, section 5, is devoted to the conclusions and future work.

CHAPTER 2

Literature Survey

Trading on the internet has gaining much more momentum than before than recent years. While ubiquity of internet is not the only reason for this phenomena, it is certainly one of the biggest and one of the most driving reason of all.

Trading mechanisms also evolve day by day because of the evolution of technology and this evolution not only enables more sophisticated products but also more convenient and easy to use ones. As a rule of thumb, when a technology matures enough from both capability and convenience perspectives, its chance of being used by the masses increases. This is what is happening with the e-trading and e-auctions today.

E-Auction's evolution is quite parallel with the normal auctions. The simplest one, single item English auction is the currently most used and preferred auction online because of its simplicity. Web sites like eBay and GittiGidiyor [9] are providing this service. While English auctions are perfect fit for a single item, buying more items simultaneously is not an option and since it is a normal auction, the aim of the auction is to maximize profits and suspect to price inflation.

With the increasing computing power and evolution of internet towards a bigger, more connected, secure and enterprise ready communication medium, more advanced trading models also start to gain traction. One of these models is combinatorial auctions. Since combinatorial auctions both allow auctioneers to sell multiple items at once and buyers to buy only the items they deem useful for them, is extremely flexible for both parties. Unfortunately, this flexibility brings the penalty of the winner determination problem which is discussed under chapter 3.

Combinatorial auction concept is not new. Designed in 1981 by Rassenti to help allocation of the airport runway allocations [6, 7] in a way that both maximizes the profit and enable every customer to bid for the runways that they want most. While not used widespread in everyday trading, today, combinatorial auctions generally finds its place in government contract auctions and other buyer initiated big markets in a form called combinatorial reverse auction.

Reverse auction idea is similar to auction. The only difference is the initiator party for the auction. On the other hand this simple reversal leads to big consequences on the mechanism of the auction. When the auction is reversed, the goods are not supplied by demanded and as expected, the demanding party wants to buy goods with minimum price in turn minimizes the profit of the seller. As the net result, the task of the auctioning action changes from maximizing the profit for the seller to the minimization of the cost for the buyer.

While not widespread like combinatorial auctions, reverse auctions has some real-life applications too. For example, Imandi [2] as an expert marketplace works with reverse auction methodology. In Imandi, buyer submits a service request about its need and this request is propagated to the service providers. Then, these providers provide their price estimates on the service requested. After all requests are gathered, the demanding user selects the winning bids and obtains the service from the selected winner. While this is not a pure reverse auction where the lowest bidder is guaranteed to win, the buyer has the right to the select the bid that fits its budget as the winner.

Another service, SorCity [3] again provides a reverse auction environment. The difference of the SorCity is the scale of the auctions. While Imandi's auctions are somewhat personal, SorCity's auctions are much more bigger with longer waiting times. Also SorCity's auction portfolio ranges from small goods to big capital investments. While the size of the auctions get bigger, the model is mostly same with Imandi. Buyers publish their requests to the site and since the auctions are much bigger, they stay open until the end time. During this period sellers can ask questions and these questions & answers are visible to asking bidder only. During the auction period every bidder can lower its bid and monitor the activity on the auction. At the end of the auction, all bids are presented to the buyer and buyer selects the winning

bid. When the buyer the winning bid, the winning provider is notified and a sales commission is paid by seller to the SorCity. After that point buyer and seller works directly and SorCity is out of the loop.

Reverse auctions can also be conducted in a combinatorial way. With combinatorial reverse auctions, big service requirements can be obtained from many service providers. Combinatorial reverse auctions work similar to normal combinatorial auctions [10]. The buyer declares its needs and calls for a combinatorial reverse auction. Every bidder gives a bid for the services they are willing to provide. According to rules of the game, bidders can submit iteratively or in a single sealed bid. After the end of the auction the winner can be determined by the solution of the resulting winner determination problem. The winner determination problem for combinatorial reverse auctions can be formulated as Weighted Set Covering Optimization Problem and discussed in further in this thesis. The generated problem can be solved with the help of a solver with very high efficiencies for smaller problem sizes. Note that weighted set covering optimization problem is known to be NP-Hard [11, 12]

While reverse combinatorial auctions are hard to come by in daily life, they are widely used in government contract tenders and other big, important and price sensitive markets. Some real life applications of these auctions can be seen below.

In 1997, Chile has started to use combinatorial reverse auctions to make government contracts with the companies which provide food for the 1,300000 low-income school children. Utilization of this auction methods saved them $40 million of their $180 million budget on the first year while improving the meal quality, ration and the coverage [4].

Chile made this possible with a simple combinatorial reverse auction mechanism. Every company is made a technical proposal alongside with different bids at different financial points. These bids contained the territories where the company bids and the prices of the services they will provide on that territories. Chilean land contains 90 territorial units but the auctions only made for 30 of them for contracts that last three years. The complexity of the auction is increased because of the different food requirements of the children at different ages, the requirement of selection of food

according to the economic situation of the school and of regional food habits. These complexities are conquered with the development of a mathematical model, similar to set covering problem that we use in our system and the utilization of the CPLEX solver. In this application, many optimization runs are made on different scenarios differentiated by cost, quality, menus and other considerations. A winner combination is selected according to the outcome of the simulations made with the model [4].

All of the aforementioned systems regardless of their auction types and mechanisms require some type of human interaction. More precisely, reverse auction sites does not support combinatorial auctions and hence leaves the winner determination to the demand owner. The only combinatorial reverse auction case solves the winner determination problem using CPLEX but with human intervention again. None of the aforementioned systems has the capacity or the feature of completely automating the call for proposal, bidding and winner determination procedure. While these systems and solutions give demand owners the ultimate power, they are not suitable for the fast trading markets.

A system called CAWP (A Combinatorial Auction Web Platform) automates the winner determination problem in consumer to consumer combinatorial auction process. Essentially CAWP is a web application like eBay with support for combinatorial traditional auctions [13]. CAWP is a platform capable of advertising and opening auctions, collecting bids and determining the winner(s) of the combinatorial auction in a totally autonomous manner. CAWP obtains its solving power from two different solvers which implement different algorithms called CASS & CABOB [14, 15]. CAWP successfully integrates these solver to the web platform it provides and automatically generates the required models to solve this problems at the end of the auction time. CAWP can successfully scale up to 30 items and 5000 bids without requiring any significant processing time [15].

While CAWP determines the winners autonomously, the auctions that CAWP conducts are traditional C2C combinatorial auctions which are unfortunately not very popular and useful in fast trading and consumer markets.

CRATI infrastructure can be used on C2C, B2C and B2B2C combinatorial reverse auctioning scenarios. In addition to traditional sourcing problems, CRATI can be

used in highly cost-sensitive markets like supply chains. Since CRATI can find the lowest price for a set of items (tangible items like goods and intangible items like services alike), it can find the best integration of a supply chain in a single combinatorial reverse auction session and can greatly reduce the costs of supply chain management. CRATI has CAWP like capabilities but with a completely different architecture and features. When compared, CRATI infrastructure has the following features:

- Completely automated combinatorial reverse auction processing,
- Completely unattended winner determination and selection,
- Automatic handling of seller and buyer notifications,
- Self updating dynamic goods list with true mean prices,
- Highly modular architecture in every level, hence highly adaptable,
- Being tailored for nearly real-time, fast trading environments and consumer markets,
- Completely operating system agnostic, built on completely open source tools and can easily be deployed, integrated or adapted,
- Can work on platforms other than web front end as a native application,
- Fault tolerant, scalable on-the-fly,
- Real world application ready architecture.

CHAPTER 3

Background Information

3.1 Multi – Agent Computing

Multi-agent computing is a relatively new computing paradigm that uses entities called agents. Agents are autonomous entities that can decide, interact or make actions to complete the goal they are designed to accomplish and these abilities are the main difference compared to the programming paradigms. The three main abilities of agents are as follows by Woolridge and Jennings [16]:

- **Proactiveness:** In an agent based system agents do not wait for requests to come. Instead, they act at their own initiative.
- **Reactivity:** An agent perceives its environment and respond to changes to that environment to accomplish their design goals.
- **Social ability:** An agent can interact with other agents (and possibly humans) to achieve their goals.

Multi-agent computing paradigm has been standardized and these standards are maintained by FIPA (The Foundation for Intelligent Physical Agents). FIPA has been founded in 1996 as an international non-profit organization to create these standards. The goal of the FIPA was to build the standards of the multi-agent computing in order to make software agent technology usable by the industry. Currently FIPA is the only overseeing authority over the agent technology as a part of IEEE (Institute of Electrical and Electronics Engineers) and continues to develop and oversee the relevant standards for the technology [17].

The main difference of the multi-agent systems is the introduction of competition and autonomous decision in every level of the system. Every agent does not have to accept every request or support every decision if the outcome is not in its designers' best interest. Furthermore, an agent can develop strategies or answer with proposals that satisfies its interests more. The ability to autonomously negotiate and decide leads the way for more sophisticated platforms that can cope with complex problems in a way that maximizes the interest of all involved parties.

While all these features can be implemented in objects very easily, the main technical difference of multi-agent computing is the encapsulation of every agent in its execution space (i.e. thread). In real multi-agent systems, the agents are totally isolated from each other from the programmers' perspective hence the only way to accomplish goals is limited to the three aforementioned properties of the agents. This successful isolation of agents from each other also has benefits for the cooperative multi-agent systems such as isolation of misbehaving (or malfunctioning) agent inside the system hence increasing the reliability of the system.

This programming approach has many well-fitting applications in the field. For example, mission critical high redundancy systems such as airplane computers or power plant control systems can benefit from this approach by using votes from agents for critical decisions [18]. In these scenarios, if a sensor or system (represented by an agent) fails or just malfunctions temporarily, these anomalies can be filtered in voting process. Multi-agent systems also fit well into real time trading or negotiating systems. Every agent, representing a different trading party, can buy, sell or negotiate for goods or financial instruments in real time without human interaction and these agents can compete to accomplish their goals (i.e. maximize their profit).

3.2 Java Agent Development Framework

JADE (Java Agent DEvelopment Framework) is an open source multi-agent development framework written completely in Java and conforms to the FIPA specifications. JADE provides APIs, services and interfaces to enable implementation of truly platform independent multi-agent system that can interact

with other non-JADE or non-local systems.

To understand JADE, the terminology of JADE framework must be known. Below is the basic terminology of the JADE which was compiled from the official JADE book [17] and will help the reader to understand how JADE realizes the agent computing paradigm aforementioned in the previous section and will be detailed later.

- **Platform:** A running JADE system is called a platform. A platform consists of agents and containers.
- **Main container:** The first created container in a JADE platform is special and called main container. This container also hosts the special agents required for platform functions.
- **Agent:** Agent is the basic entity in the JADE ecosystem. They are the entities which are designed and programmed by the developer of the system and they are the workers of the platform.
- **Container:** Container is a controlled environment that agents live in. These containers can consist of a single system or can span more than one physical, separated system. They are various ways of spanning many systems which yield different results, advantages and features.
- **White Pages Service:** White pages service is the directory that holds the name and address of every agent which is in a container. This service is provided by an agent that is responsible for other important tasks in the container.
- **Yellow Pages Service:** Yellow pages service is an opt-in service that agents register and unregister at their will. Yellow pages service can host more detailed information about an agent including the services it provide, languages, protocols and ontologies it supports and can understand.
- **AID:** Agent IDdentifier object. These objects contain agent's complete name and address information and widely used while sending messages. It's also

possible to construct AIDs from names of the agents (for platform-local agents only).

JADE provides many features which is needed and useful in development of such systems. Some of these features can be summarized as follows:

- **Distributed by design:** Any agent in a JADE container can directly send messages to any agent in a different container. Also, distributed JADE containers can work cooperatively on multiple systems without extra effort.

- **Asynchronous messaging:** Messaging in JADE environment is handled in asynchronous manner. This means that agents can send messages and continue their tasks without waiting other agent to receive or an agent can receive incoming messages anytime. No messages lost in the system because an agent is not listening for incoming messages at the time of message sending.

- **Global notification subscriptions:** A global notification system is integrated in JADE and any wishing agent can use this subsystem to inform interested parties (including agents and external applications alike) about events that occur on the platform.

- **Agent mobility:** An agent in any container can move to another container and under certain circumstances can continue its run from where its left. Agents are not blocked and can interact with the system during migrating from one system to another.

- **Central agent registries per platform:** Aforementioned white pages and yellow pages services enable agents to be environment aware and enable them to interact with other agents without prior knowledge about the platform. Also these decentralized services can be federated with more than one platform which enables agents to see and interact beyond the platform they live in with minimum effort.

- **In-process interface:** JADE provides an in-process interface to enable programmers to launch and control a platform and its components from an

external application hence integrate JADE to another application either for GUI purposes or otherwise.

JADE also implements these features in a distributed manner in a single container. When a JADE platform is booted, two special agents are created. These special agents are created in the main container only. The other registered containers (if any) are also managed by these two agents called AMS (Agent Management Service) and DF (Directory Facilitator Services). First of these agents, the AMS, is responsible for the management of all agents on the platform. Agents also can interact with AMS for accomplishing many tasks including life cycle management of other agents. Second special agent DF is the maintainer of the yellow pages directory. While registration to DF is not mandatory, it has many benefits for agents that except interaction from stranger agents on the platform or other platforms. Yellow pages service maintains much more information than white pages including the name of the agent, the services it provides and ontologies, protocols and languages that it supports for interaction under every service. Also any agent can subscribe to these agents for getting information about the events (including but not limited to registration, join, leave, message sending events) that occur in the container. Besides special agents, main container also hosts some special data structures called GADT (Global Agent Descriptor Table) and CT (Container Table). While GADT stores the names and the locations of every agent in the platform, CT holds references and transport addresses of the containers that make up a platform. These two tables are used by the platform to route messages to the correct agent in the correct container.

The messaging system in the JADE is completely asynchronous and handled by MTS (Message Transfer Service). MTS can carry messages between any two agents regardless of their location hence it is both container and platform transparent. MTS uses two distinct message transfer protocols for delivering messages in the JADE ecosystem [17].

Agents recover incoming messages from their inboxes which are delivered by MTS. If the agent is sleeping, it is waked but otherwise an agent's ongoing execution is not interrupted or modified in any way. Also agents are free to check their inboxes whenever necessary. If an agent has no messages simply retrieves nothing. Similarly

when sending messages, these messages are immediately handled and delivered by MTS and agents do not have to wait for sending messages. Message is sent and the execution continues without waiting message to arrive its destination [17].

The standard messages in JADE (which are called ACL Messages) have many meaningful fields. Some of these are performative, language, ontology, protocol, reply by fields. Roles of these fields can be summarized as follows

- **Performative:** Performatives represent the intention and meaning of the message. An inform message is guaranteed to carry some message for informing purposes. Similarly a call for proposal message contains details about a proposal call. This field is one of the most important fields in ACL Message format.
- **Language:** Carries the name of the language of the message that is encapsulated inside this message.
- **Protocol:** Defines the protocol of the message that it belongs to.
- **Ontology:** Defines the ontology of the attached message.

All of the language, protocol and ontology fields are unchecked string fields. This means that programmer can carry any string in these fields.

In JADE, distribution lists are maintained by another service called Topic Manager. Topic Manager returns special AID addresses when a topic creation request is sent. Likewise, subscription messages are also sent to topic manager hence Topic Manager service associates this special AIDs to subscribed agents. When an agent sends a message to that special AID, the message is distributed to every subscribed agent in single shot. This system is extremely useful for notifying many agents about an event or like.

The asynchronous behavior of agents is implemented by threading. Every agent has one thread and can run independently from other agents and services in its thread. Since every agent has a single thread to work, there must be another mechanism in the framework to implement the tasks of an agent. This gap is filled with behaviors,

which can be described as task units. Behaviors are implemented during development and can be run in different order according to some rules which are embedded into the agent during the implementation. To enable intelligence via behaviors every agent runs an embedded behavior scheduler which is not visible to implementer.

3.3 Choco Solver

Choco is an open source Java library that is used to solve constraint satisfaction problems. Main function of Choco is to model and solve CSPs (Constraint Satisfaction Problems) including set covering problems. Development of Choco has been started in 1999 with OCRE project, which aims to create a constraint solver that can be used both for research and education purposes. While first versions of the Choco have been developed with CLAIRE language and compiled to C++, in 2003 ported to Java for portability [19].

Latest generation Choco consists of a problem modeler and a problem solver. Choco's problem modeler can manipulate the following variables:

- Integer variables,
- Set variables that represent a set of integer values,
- Real variables which takes their values a range of floats,
- Real based or integer based expressions via operators like plus, minus, mult, div and more.

Choco problem modeler also supports a wide range of constraints

- All classical arithmetic constraints (equal, not equal, less than, greater than, etc.) for integer and real variables,
- Binary constraints on real and integer variables,
- Table constraints that defines a set of tuples that verify (or don't verify) the intended relation for a set of variables

- A large set of classical global constraints which are commonly used and very useful (e.g. alldifferent, global cardinality and more).

Choco is also a constraint problem solver which provides many implementations for various domain types (integer, bounded, enumerated, etc.), and state of the art algorithms for constraint propagation. Choco can run in various modes such as satisfaction mode (which finds a solution or all solutions and iterate them) or in optimization mode (maximization or minimization). Also, solution search can be parametrized by the many ways including design of custom variables by user, predefined variable and value selection heuristics, definition of a decision variable and more. Last but not least, Choco can use other solvers via its plug-in mechanism and runs in preprocessor mode in this case to optimize the problem model heuristically before sending it to the solver [19].

Design of the Choco also divided into two main parts, problem modelling and solving. First of the two main APIs is dedicated to problem representation in most user-friendly and flexible way. Second of these APIs is for problem solving and optimization of this solving process by fine tuning the solver. Choco's CP solver utilizes tree search methods to satisfy constraints as requested by user and works on the model created by Choco's problem modeling API [19].

Choco project does not only maintain the solver and the modeler but also a range of visual tools to help in modeling, tracing and visualizing the inner workings of the Choco solver. These tools are called Choco Visualization Tools. These tools while not utilized in the implementation during the thesis can be used in either debugging or education [19].

3.4 Reasons Behind Selection of JADE and Choco

One of the most important steps in the design and development of a platform is the selection of the tools and libraries that will be used for development. When developing sophisticated software, it is a common exercise to use libraries and frameworks which will help achieving the goals of the project at hand.

In our case, the platform will be multi-agent, competitive, reverse combinatorial

auction platform and implementation of such platform depends on two core components. These are a multi-agent platform and a problem solver.

A multi-agent platform is a standardized and complex environment and complete implementation of such complex and big platform for a single project is not feasible and unnecessary. Because of this situation it has been decided to use one of the good quality frameworks. The requirement of the selection of the platform was as follows:

- **Mature and standards compliant:** Since multi-agent paradigm already has defined standards, it is always better to obey these standards. On the other hand, platform should be mature enough to handle a big and sophisticated system developed on top of it.

- **Cross platform:** There were three reasons for the requirement of a multi platform framework. The first one was the intention of the mobile client implementation. Secondly, the implemented code was planned to be as production ready as possible, we needed a framework that can run on variety of operating systems because the server scene is not dominated by Microsoft Windows®. Lastly the developer of the platform was not using Microsoft Windows® as an operating system and selection of a cross platform framework would greatly reduce the overhead required to start implementation.

- **Well supported:** Since the designed platform was sophisticated and probably will test the limits of the framework, a well known framework with an active community was essential.

The JADE platform which satisfied all of this requirements have been selected as a result. Another important reason for selecting JADE was the familiarity with the framework since we have implemented some small scale projects using JADE.

Solver selection is made after multi-agent platform selection. Again, the requirements for the solver was the same (maturity, cross platform compatibility and community support) with the platform. While most of the better solvers are not free to use and limited in problem size, Choco was found as an excellent choice. While satisfying all of the requirements, also has advantages like being written in same

programming language with JADE, coming in form of an API instead of binaries that use text files for input and output, and being a first class solver which has been also used by big customers like NASA made Choco is the ultimate choice for the platform.

Both platforms also have some other advantages like being fully open source which makes them easy to understand and debug in situations where it's hard to understand the source cause of the problems. Being open source also implies a big community support since anyone willing to understand any of the platforms can dissect and dissemble the platform in the process and get hold of every aspect of the tool.

During the implementation of the project both JADE and Choco proven themselves as the right choice many times from different aspects however, the problems encountered during the multi-threaded usage of Choco solver will be detailed later.

3.5 Reverse Combinatorial Auction

Auction is a method of selling objects and its history is old as Roman Empire and basically involves selling an item to the buyer who is willing to pay the most. Actually, the word auction comes from Latin word augere, which means "to increase" [1].

Auction has many forms which has different principles. Most common type of auction is the English auction. In English auction, the auction starts with a low price announced by the auctioneer. Then this price is iteratively increased by bidders until there's a single bidder left interested to the item and the item is sold to the last interested bidder who actually given the highest price. English auction has a close relative, called Dutch auction tries to achieve same result differently. Dutch auction start from a very high price set by the auctioneer and then price is gradually lowered. Then the item is sold to the buyer who shows his/her interest first and again paid the highest price for the item. While Dutch auctions are not as popular as English auctions, they draw conceptual interest. Not every auction type has open incremental/decremental bidding. Auctions can also can be conducted with sealed, single shot bids. These auctions are similar to English auctions but every bidder submits one, closed bid and the winner pays his bid or the second bidder's price

depending on the rules of the auction [1].

All of the aforementioned auctions are working best with single items but sometimes some related items are auctioned together and the concept of combinatorial auction emerge. Combinatorial auction is the concept of multiple items at the same time but letting bidders to bid on parts of the package instead of forcing them to buy the whole package. This auction mechanism is very sound and well fitting for some cases and benefits both the seller and the buyer because in the end, auctioneer sells all of the items in its hand to the bidder(s) who pay(s) the most and the bidder(s) only pay for the items they deem worthy. But this mechanism brings with a big problem called winner determination problem. Since there are many items, many bids and many bidders, the bids which maximize the revenue of the auctioneer must be determined and this is not an easy problem [6].

There are two well-known approaches to combinatorial auction winner determination problem solving. One of these is CASS (Combinatorial Auction Structured Search) which combines a brute force search approach with significant heuristic improvements [13]. CASS structures the search space in order to find the solution faster. The second approach is called CABOB (Combinatorial Auction Branch on Bids) which uses a tree search algorithm that branches on bids. CABOB can prune large sets of fruitless bids with upper and lower bounding so can limit the search space significantly while searching for the optimal solution [14].

If the buyer and the seller in an auction changes their roles and the auctions are initiated not by the seller but by the buyer, these auctions are called reverse auctions. In reverse auction every rule turns upside down. In single item reverse auctions, lower bids win. In combinatorial reverse auctions the problem model completely change. The problem transforms from a maximization problem to a minimization one with the constraint of supplying every item buyer demands [10]. On the other words, buyer tries to obtain every item he/she wants for the minimum price using the bids given by the sellers and if a bidder wants to win a combinatorial reverse auction, it must give the lowest price for the preferred set of items. In fact, winner determination problem in combinatorial reverse auctions can be formulated as a weighted set covering problem. SCP decision problem is a member of Karp's 21 NP-

complete problems and has been proven to be NP-complete in 1972. [1, 21]

The developed formulation is given in the next section.

3.6 Determining the Winners vs. Weighted Set Covering Optimization Problem

The winner determination problem in the reverse combinatorial auction scenario fits into the optimization problem of the WSCP since the goal is similar. The version of the problem that covers the cost of using the sets is called WSCP (Weighted Set Covering Problem) and can be stated as follows:

There's a finite family of finite sets $S_1, S_2, \ldots, S_n$ and each set S_i has a given weight w_i, which is a real non-negative number. Let U be $\bigcup_{i=1}^{n} S_i$ (the universal set). One wants to find a subset of the family specified by $I \subseteq \{1,2,\ldots,n\}$ such that

$$\bigcup_{i \in I} S_i = U, \quad (1)$$

for which total weight

$$\sum_{i \in I} w_i \quad (2)$$

is minimum. This problem is known to be NP-hard, even if all weights are equal to 1 [12].

In our case, winner determination problem for reverse combinatorial auction fits to the weighted set covering problem as follows

- **Set to be covered:** Items requested by the auctioneer. Since our auction is reverse, items are demanded, not offered by the auctioneer.
- **Weight:** The cost of every bid. Minimization of the weights ensures cheapest combination of goods.
- **Candidate sets:** Bids offered by the sellers. In our problem we are trying to cover the set to be covered (the demand set) with the candidate sets (bids).

As seen above, the winner determination problem can be perfectly mapped to the weighted set covering problem. This means that we can easily solve the problem with any method that efficiently solves weighted set covering problem. Since in weighted set covering problem, we are trying to satisfy a constraint, which is minimization of (2), the problem is a constraint satisfaction problem. This means that we need to construct formulas that reflect the constraints in the problem. This constraints then fed to Choco solver alongside with the main cost formula to find the optimal solution.

The constraints for the optimization problem can be constructed as following

- **Every bid is an integer variable which is either 0 or 1:** Since we are taking a bid or not, the bids are represented by integer variables that are either 0 or 1. In the model, these variables behave as boolean selectors.

- **Every item must be obtained from at least one bid:** This requirement results in a series of equations. To successfully cover item set I, for every item i, at lease one bid b, which is $b \in B$ and contains item i must be present in final selection. This can be formulated as following for every item $i \in I$

  $$1 \leqslant b_1 + b_2 + ... + b_n$$

  where $b_1, b_2, ..., b_n$ are bids that contain item $i \in I$.

- **Cost of the selected sets must be minimized:** In order to obtain all items for minimum price, the cost of selecting sets must be minimized. This requirement makes cost variable as our optimization target which is also the target in the original weighted set covering problem and can be stated as follows where w_b is the cost of the bid $b \in B$ and B is the set of all possible bids.

  $$cost = \sum_{n=1}^{|B|} w_i \cdot b_i$$

When the above formula is given to the Choco solver alongside with the constraints

aforementioned and asked for the minimization of the cost, the resulting cost is our minimal auction closing price. Further analyze of the variables yields the selected bids which are simply the winner bids.

Furthermore, the availability of a solution that satisfies the request of the auctioneer is guaranteed in two ways. Firstly, the infrastructure only present goods that are already obtainable and present in the marketplace. Secondly every seller in the infrastructure must place single item bids for every item it can supply so these bids can be used to fill small missing items from other combinatorial bids, if any. This two aspects always guarantee that the items requested are present in the marketplace and be obtainable hence guaranteeing a full item coverage.

CHAPTER 4

CRATI

4.1 Introduction to the CRATI Platform and to Rules of the Game

CRATI (Combinatorial Reverse Auction Trading Infrastructure) is a software infrastructure designed to conduct client originated, on-demand combinatorial reverse auctions on a multi-agent environment. CRATI aims to provide a diverse and large market environment with merchants that have partially substitutable stocks in order to promote positive competition in the market. In short, when a client needs items, a reverse auction on behalf of the client is opened and conducted. Then this auction's winners are determined as quick as possible and the outcome is presented to the user. On the other hand, if the winner merchants prefer, their winning bid(s) can be announced to the marketplace to further enhance and promote competition.

CRATI uses a distributed approach which employs a number of different agent types. Every agent type has its own roles in the system and runs totally independent from each other. Also since these agents run in different threads, system can run in a parallel manner and can handle multiple auctions simultaneously. Details of the CRATI is given in the next sections of this chapter.

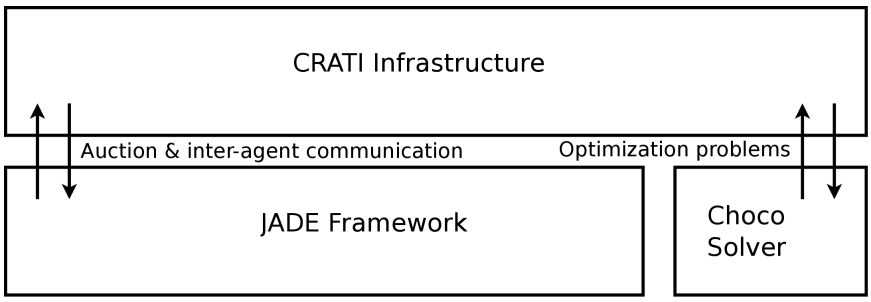


Figure 1: Interaction of CRATI with JADE & Choco Solver

4.2 Design Philosophy and Requirements

Like every software product, CRATI is designed around functional and non-functional requirements. The non-functional requirements of the CRATI platform is highly parallel with the non-functional requirements of a real-world trading platform. These are performance, correctness, reliability, availability, fault tolerance and security.

One of the biggest requirements of a trade platform is performance. During a traditional trading process, neither selling nor buying party wants to wait too much. Also, if the transaction time is long, the good being traded may lose its utility. In order to minimize transaction time in CRATI platform, platform is designed for performance and scalability from ground up with the usage of high performance data structures and high performance algorithms. More details about performance is given on architecture section.

Trading is revolving around one of the most valuable and sensitive asset, money. During trading, exactness and correctness of the calculation carries utmost importance. To satisfy these requirements a Java provided, special data structure called BigDecimal is used. BigDecimal's most important feature is its exactness, since stored number is not stored in traditional IEEE float formats, it does not suffer from approximation errors that floating point formats may suffer. Also, to ensure the reliability of the optimization process which is the core of the CRATI infrastructure, we selected a solver that is well supported and used in real world. Our selected solver, Choco is used by many research institutions and some serious organizations

like NASA [22]. Also the optimization problem is modeled as the well known weighted set covering problem. Usage of specialized data structures alongside with a proven solver using well understood and founded problem model guarantees the mathematical reliability of the CRATI infrastructure.

Another important requirement in trading is security. Security in trading is a multifaceted problem and requires attacking from many sides. CRATI platform has no connection security between client and server but on the other side, platforms inner transactions are highly secured against object tampering. None of the messages or objects inside CRATI platform has direct access to its variables. Everything can be read via getter methods but, cannot be modified since objects has no setter methods. The only exception to this rule is the bid envelope structure which has setter methods but these envelopes must be sealed before sent and sealing an envelope not only disables its setter methods but also causes a generation of a timestamp which is again uneditable. If any of the messages are sniffed and intended to be tampered, these objects must be re-created hence making attacking infeasible and somewhat time consuming.

Fault tolerance and reliability of these systems also important and CRATI has no compromises in this area too. Since CRATI runs many agents in completely isolated threads and an agent type has more than once instance for scalability, in an unlikely event of an agent crash, the only thing is lost is the auctions or bids connected to that agent. In such an event other agents and JADE platform can continue without any adverse effects and outcomes. To further reinforce the stability of the platform every agent has been developed with highest fault tolerance possible. Since message transport mechanisms and the underlying code that powers the JADE framework is also extremely stable, the resulting system is highly dependable and stable. This stability can be improved another notch by utilizing the redundancy features of the JADE platform. With these redundancy features every aspect of the CRATI platform and the JADE framework can run in fully backed-up mode.

While not a functional requirement, usage simplicity without compromising the functionality of the system is a well received trend which continues to gain traction. Currently all major software developers are trying to make their systems easier to use

while keeping the utility of the software at maximum. Since trading can be a hard process sometimes, making it as simple as possible without cutting the benefits was one of the design considerations of the CRATI platform. To accomplish this, the platform makes a lot of automation and assumption instead of the users. Currently trading in CRATI platform requires only three steps. Selecting items, their quantities and the maximum price. In order to give an idea to customers, CRATI calculates the mean price of every item in the marketplace using the prices supplied by sellers. While a simple mean price has been chosen for its simplicity to accuracy ratio, more advanced approaches can be used to improve the accuracy or other aspects of the framework. Everything required to make this platform work is handled automatically at the background, without any interaction.

System is designed to be as practical as possible for the sellers. The only thing sellers need to provide is their stocks in a standard XML file. Since system is agent based and fully automatic, their delegated agents are handling everything from bidding to handling the winning even further the bidding process can be realized in an intelligent way supported by various artificial intelligence and/or machine learning techniques and algorithms. The platform is also highly scalable and this means that there's virtually no limits for the number of sellers. Also CRATI is not tailored for any markets. Everything that can be sold over normal markets can be theoretically sold on CRATI the only limit is the suitability of the item for this type of market. For example, when the market competition among sellers is high.

Even if the sellers decide to not to give any combinatorial bids during the reverse auction process, the trading continues as normal and the solution is calculated over the given single item bids. In other words, combinatorial bids are not a must in CRATI.

As a result, CRATI platform is designed for reliable and easy trading without any compromises on performance, scalability and flexibility.

4.3 Development Methodology of the Software

While CRATI platform was developed by a single developer, the development process has been also designed alongside with the platform architecture in order to

protect the clarity, maintainability and the ability to plan ahead while keeping the progress made visible. In order to satisfy all these requirements, usage of SVN alongside with Trac has been decided. For stability and time constraint reasons, these services are bought from a firm called Assembla [23]. Assemble is a company that provides free/open source project hosting spaces alongside with private paid spaces for private projects. The setup used in this thesis was a rather simple Trac & SVN setup. More information and usage of these tools follows.

SVN (Subversion) is an open source and free VCS (Version Control System). It's one of the most advanced VCSs and widespread in use. SVN supports every major feature that one expects from a VCS (automatic versioning, branching, rollbacks, merging and more) and these features are accessible via many tools ranging from simple command line tools to plug-ins or extensions for Integrated Development Environments (IDEs) [24].

Second major tool, Trac is a project management extension to SVN. While Trac can work with other VCSs, SVN was selected for its familiarity and widespread use. Trac extends SVN with a wiki, a ticket tracker, a milestone planner and tracker and a more advanced source code browser. With these enhancements to a SVN repository, a Trac based system can be used as the single point for managing and planning the project with a very advanced documentation tool which can be either used as a whiteboard or a serious documentation center. Last but not least, Trac supports many users with different roles and this makes Trac suitable for crowded development teams who either develop free software or otherwise [25].

The platform has been implemented using an IDE. The IDE of choice was another free and open source development environment, Eclipse. Eclipse project is the open sourced and free version of the IBM's internal development platform and while being open source and free, it is one of the most powerful and universal development environments in the scene. Since Eclipse is written with a mix of Java and platform specific codes, supports many operating systems including GNU/Linux, Apple Mac OS X® and Microsoft Windows® [26].

The CRATI platform was developed using two computers using two different operating systems. A laptop that runs Mac OS X® and a desktop computer that runs

Debian GNU/Linux. Eclipse IDE with same features has been installed on both computers.

The development of the software is done in a planned manner. Before starting implementation, the general architecture is laid according to prior knowledge and experience about the framework. After the initial architecture design, inter-agent interaction have been thought out and designed together with the auction work flow. During the design of the auction protocol and inter-agent interactions, the steps that possibly need more processing power and be probable bottlenecks are spotted and alternative and high performance data structures or approaches that will possibly help are designed. Choco solver is also selected at this point.

After the completion of the initial design, the development cycle started. Development has been divided into planned milestones. A milestone consists of the following elements:

- **A focus:** Every milestone has a general focus. It can be either adding a new functionality or extend an existing but incomplete to become more complete or both. During the development cycle, project had only one maintenance and cleaning cycle.

- **Tickets:** Every milestone has some tickets to be closed. This tickets are opened according to both the focus of the current milestone and the outcome of the previous milestone.

- **One or more code commits per milestone:** During the development, the code has been submitted as often as possible, generally after closing a ticket or fixing an important bug. These commits always accompanied by complete change logs to maintain progress visibility & transparency.

During the development, there were some principles about the milestones and the tickets included in the milestone. They can be listed as

- **No postponing without reason:** A ticket in a milestone cannot be postponed into another milestone without any legitimate reason. The only legitimate reason for postponing a ticket was to requirement of one or more feature

implementations which are not in the focus of the current milestone. Even in this situation, tickets were implemented to the limit of possibility and then postponed after updating with necessary information.

- **Discovered bugs must be fixed in current milestone:** If a bug has been found during milestone, that bug must be fixed at that moment if possible. If not, a ticket must be opened as a blocker and the bug must be fixed in that milestone.

- **Integrated complete testing after implementation:** The implemented code must be tested before closing tickets both for module integration and platform integration. With that requirement, the code is always guaranteed to work as expected and the need for lengthy testing runs has been eliminated.

During development, the next milestone is always designed after closing current milestone and the next milestone is designed in such a way that every implemented feature shall be relied upon already implemented features so the whole platform can be tested and verified over and over during the development.

The design of other elements, e.g. the internal behavior design of the agents and data structures required during agent interaction, has been designed on the fly according to the design philosophy and requirements of the agent and its tasks.

4.4 The Architecture of the CRATI Infrastructure

CRATI is designed with scalability, extensibility and flexibility in mind. The main idea behind the design is to give dedicated jobs to agents in the platform and making them work cooperatively. When this design combined with the asynchronous and multithreaded nature of the JADE, resulting platform is responsive, ready to scale and ready to be distributed to multiple systems with the help of container extension.

CRATI contains six types of agents with completely dedicated roles (see Figure 1).

- **Client:** Client is the main agent in the infrastructure that requests items and hence auctions. Client interacts with Gate Keeper and Auction Coordinator agents during the trading process.

- **Gate Keeper:** Gate Keeper agent is the welcoming agent to the trading infrastructure, as its name implies. When a client connects to the infrastructure it first searches for Gate Keepers to interact. Gate Keepers give the client the item list (which is always up to date) and also sends the address of an Auction Coordinator. Gate Keeper agents interact with Clients, Directory Facilitator Service and the Stocker during the trading process.

- **Auction Coordinator:** Auction coordinator is probably the busiest agent in the whole infrastructure. It handles incoming auction requests from clients, announces the auction, dispatches them to one of the optimizer agents. Also relays the results of the optimization process to client and the interested merchants. Auction coordinators interact with Clients, Merchants, Optimizers and the Stocker during the process.

- **Optimizer:** Optimizer is the agent that hosts the Choco solver. Optimizers receive finished auction with the attached bids and finds the best solution alongside with the winner bids and their owners with the help of Choco. Optimizers only interact with Auction Coordinators during their life cycle.

- **Merchant:** Merchants are the agents that provide the goods that Clients request. Merchants send their bids to the auction coordinator when an auction is announced. Their bids are only accepted until closing time and then not considered as a valid entry afterwards.

- **Stocker:** Stocker is the only agent in the infrastructure which cannot scale. Stocker contacts with Merchants and inform them about allowed item categories and request their complete stock information. This information is used to generate the complete item list which is sent to the Gate Keepers and Clients via Gate Keepers. Also Stocker creates the subscription topics for announcements and sends them to new Merchants and Auction Coordinators. Stockers interact with Auction Coordinators, Merchants and Gate Keepers during their life cycle.

Every agent in the CRATI infrastructure, excluding the Stocker, can have more than one instance during the infrastructure's lifetime. While this is a requirement for

Merchants and Clients, it's not mandatory for other agents but in order to make load balancing and scaling possible, other agents are enhanced to support this feature. The reason for exclusion of the Stocker is the work pattern of the agent. Stocker only works when a Merchant, Gatekeeper or Auction coordinator arrives or leaves and this is not a common event in the platform. An overview of the CRATI platform's architecture, complete with connections and communication can be seen in Figure 2.

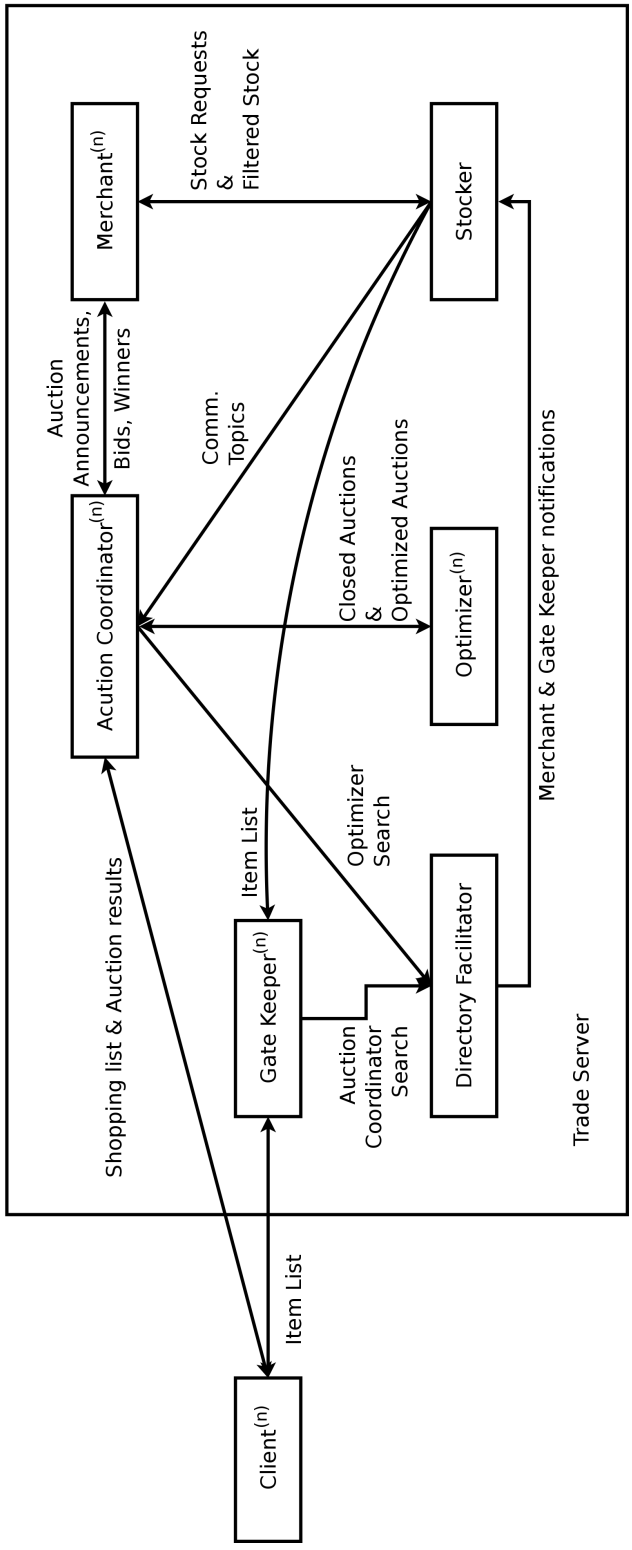


Figure 2: CRATI Platform Architecture

In Figure 2, the agents denoted with *(n)* can exist more than one.

Following is a legend for the items that are transferred between agents during the lifetime of the Infrastructure:

- **Item List:** Item list is the list that contains all items that are present on the infrastructure and available for purchasing. This list is updated whenever a merchant agent is joined or left the infrastructure. This list contains item brands, names and their mean prices. Sent from Stocker to gatekeeper upon change and whenever a new Gate Keeper joins the infrastructure. Forwarded to every incoming client by a Gate Keeper.

- **Shopping List:** Shopping list is the list of items that client requested to buy with their requested quantities and the global upper price limit. This list contains brands, names, requested quantities, and the global upper price limit in percents (e.g. 15% price flexibility means the price can go up to 115% of the estimated price).

- **Auction Results:** Carries the outcome of the auction to the client. Includes closing price with winning bids, upper price limit, and estimated original price for verification purposes.

- **Auction Announcements:** These messages carry the notifications of new auctions. They carry the requested item for a category and the categories of the opened auction. With this information merchants can assemble all the messages required for an auction.

- **Bids:** These messages carry the bids of a merchant. Bids are big containers, called envelopes, that carries all of the bids a merchant for a particular auction. These envelopes have self-generated, tamper-proof time stamp for verification.

- **Winners:** Sent by auction coordinators, these messages announce the winners of an auction. Their receivers vary greatly because every merchant can set their privacy level.

- **Auction Coordinator Search:** These are Directory Facilitator searches executed by Gate Keepers in order to get the list of Auction Coordinators in the infrastructure. Executed when preparing a reply to the incoming client.

- **Optimizer Search:** Similar to Auction Coordinator Search but executed by Auction Coordinators to get a list of Optimizers. Executed when dispatching an auction.

- **Stock Request:** Sent by the Stocker to the new Merchant that just joined to the infrastructure. Contains the categories that are allowed to be sold.

- **Filtered Stock:** Contains the stock of the Merchant, filtered according to the message sent by Stocker earlier. Sent to the Stocker of the infrastructure.

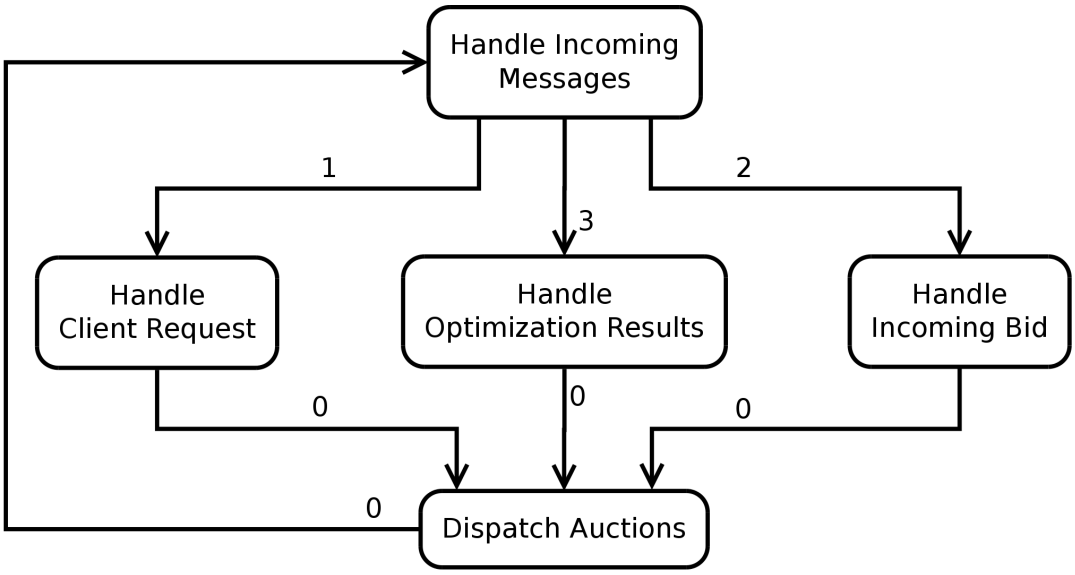
- **Closed Auctions:** These are the auctions whose time window has passed and officially closed. The sent bundle contains bid envelopes of every bidding agent.

- **Optimized Auctions:** Returned to sending Auction Coordinator by the Optimizer, these messages contain original message sent by the Auction Coordinator plus, the results of the auction including closing price, winning bids, and privacy preferences of the agents participated into that particular auction. Privacy information is extracted from bid envelopes of the agents during bid extraction.

- **Merchant & Gate Keeper Notifications:** These are the notifications sent by the Directory Facilitator to the Stocker whenever a new Merchant and Gate Keeper joins the infrastructure. These messages are sent because of the Stocker's subscriptions to the Directory Facilitator service during boot process. Messages contains the contact information of the aforementioned agents alongside other information.

Agents in the CRATI infrastructure has a unified architecture which has been designed long before actual implementation. Every agent in the CRATI implements a common interface which exposes some (mostly logging related) variables inside the class and uses the same logging and configuration file parsing mechanism. Also

agents share same core behavior architecture which has been designed and implemented using the FSM Behavior provided by JADE. FSM Behavior provides a true finite state machine inside the agent and agent can switch from one state to another according to outcome of the current state. Utilization of FSM Behavior resulted in a truly modular internal design. Since features are consolidated in behaviors, they can be easily debugged, modified, enhanced, removed or replaced. Last but not least, every agent in the platform uses a very fast and extensible message receiving and deciding system that we call Hierarchical Response Engine. Details of this system can be found in Challenges and Solutions section. The main loop of the auction coordinator can be seen on Figure 1 alongside with the table 1 which gives the details about the transitions, which is an FSM behavior where transitions are dictated by the Hierarchical Response Engine which is in integrated into topmost behavior. The graph can be read as follows. Topmost behavior can return three different values. 1 means incoming message is a new auction request by client. 2 means a new bid from a merchant has been received and 3 means optimizer has returned results of a finished optimization. The 0s returned by the other behaviors are default transitions and always the same. Other behaviors doesn't return any values other than 0s since it is unnecessary to do so.



Another aspect that CRATI architecture takes into account is the minimization of

overheads and traffic for better scalability and performance. To minimize messaging and overheads CRATI makes use of the subscription services of the JADE framework to the fullest extend possible. For example, Merchants need some information to complete their booting in the platform and instead of searching for the Stocker, Stocker finds them with the help of DF subscription services and sends required information.

CRATI infrastructure uses aforementioned language, protocol, ontology triplets extensively. infrastructure has one official language and ontology but two disjoint protocols. One of these protocols for internal communication, other one is for trading. CRATI doesn't facilitate encoding decoding facilities provided by JADE framework instead, system relies on transportation of the serialized java objects alongside with human readable and meaningful string descriptors. The language that infrastructure implements is called String Classified, Mixed Mode Language. Since the messages are handled according to their string counterparts, the messages are string classified. Language is called Mixed Mode because these messages also carry a serialized object payload in most instances and this was achieved by extending the stock ACL message structure, which can either carry information in text or object mode, by making it carry our special message wrapper.

CRATI infrastructure has been designed to work in a distributed environment from start. Ideally, CRATI infrastructure works with at least two containers on two independent agent platforms. One of these platforms is called the Trade Server. Trade Server is the platform that contains majority of the agents. Every agent except the client agent lives here either in one or more containers and work both competitively and cooperatively during the trading process. Other platform is simply called the Client and lives on another platform either in a mobile fashion or otherwise. If the platform is living in a single platform for any reason, this logical division still exists and the working mechanisms of the CRATI doesn't change since such change is completely unnecessary.

4.5 Protocols Design, Details and Implementation

In order the fulfill the communication requirements of the CRATI infrastructure, two

different communication protocols has been designed and implemented. One of these protocols is for internal communications of trade server agents and the other protocol is for auctions.

The necessity for two different protocols has been born from the need of inter-agent communication unrelated to auctions during the life of trade server. While the communication is unrelated to auctions, crucial for the health and availability of the server. The internal communication protocol handles the following situations

- Supplying the allowed categories to the merchants during merchant boot-up,
- Supplying the channel names that merchants subscribe for waiting for the auction announcements,
- Sending the item list to the gate keeper agents either on new gate keeper arrival or item list update,
- Notification of stocker agent when a merchant leaves the platform,
- Sending the global channel list to the auction coordinators upon auction coordinator boot up.

Second protocol, the auction protocol, which is a variation of the Contract Net protocol [27], is designed and implemented to handle the auctions that occur in the platform from start to finish. All communication done regarding to an auction is made using this protocol. Auction protocol handles the following situations:

- Handling the arrival of a new client as a potential customer and transfer of the latest item list alongside with the auction coordinator address to the client,
- Sending the shopping list that contains the requested items by the client,
- Announcement of the auction that contains the items requested by the client,
- Collection of bids regarding to auctions,
- Dispatching a closed auction to the optimizer for winner determination,

- Handling of optimized auctions that returns from an optimizer,
- Announcement of the winners to the trade server according to the privacy preferences of the merchant(s) that won the auction.
- Notification of the client about the outcome of the auction that it requested.

Both of the protocols have been designed for minimum overhead, simplicity and fault-tolerance. For example, during the boot up of the platform, stocker agent must be essentially the first agent to boot up but, if any other agent try to boot up before stocker will simply wait until stocker agent becomes available and continue booting thereafter. Similarly during an auction, only auction start is announced alongside with a time window. Participating merchants will submit their bids. Neither auction end nor merchant messages about not participating is present in the protocol in order the save internal bandwidth.

4.6 Booting and Auction Progression Details

As aforementioned, all of the communication inside the CRATI infrastructure is made over two distinct but complementary protocols. While one of these protocols help trade server to stay alive, the other protocols enable the CRATI infrastructure to accomplish its intended task, execute combinatorial reverse auctions. In this section, these two processes will be given in detail and discussed.

While booting up may seem like a natural thing and should be simple, the process may not be so simple when a highly parallel, asynchronous multi-agent system with inter-dependent agents try boot up at the same time to form a trading platform. In the CRATI infrastructure, five of the six agent types are in trade server and take active role from boot up. Following is an idealized booting procedure (see Figure 3). If the agents doesn't created with this order, system will not malfunction but will wait other agents in order to boot in the order described below.

1. **JADE platform boots up:** JADE creates the core agents, AMS & DF before creating any agents so the platform becomes ready to accommodate agents.
2. **Stocker boots up:** Stocker is one of the two agents that doesn't require any

information from other agents during boot up. It boots up and subscribes to the DF to listen for new coming merchants, gate keepers and auction coordinators. During boot up, stocker also reads some files to understand which item categories are allowed in this CRATI instance and generates required topics.

3. **Auction coordinator boots up:** An auction coordinator boots up and registers itself to the DF. After finishing its preliminary tasks, it asks the stocker about the topic list it created before. Stocker replies to auction coordinator with a list of all communication topics.

4. **Gate keeper boots up:** A gate keeper boots up and registers itself to the DF but only with internal communication protocol. The aim is to stay invisible to the clients since booting didn't finish yet. Also gate keeper also registers itself to a predetermined topic which stocker uses to send global item list changes. Stocker gets the notification about the gatekeeper and sends it a new, complete item list. After getting the item list, gate keeper modifies its DF description and adds auction protocol into its supported protocol list because it has everything it needs to serve the incoming clients.

5. **Merchant boots up:** Merchant boots up and reads its stock information as the first step. After parsing its stock information, it registers to the DF. Like gate keeper, stocker gets the notification about the merchant and sends it a message that contains the allowed trade categories. When merchant receives the message, it filters the stock according to these rules and forgets about the stock that is not allowed for trading here. After filtering stock information, merchant sends this filtered stock back to the stocker. Stocker updates internal (unified) item list and sends the new item list to the gate keepers via the predetermined channel.

6. **Optimizer boots up:** Optimizer boots up and registers itself to the DF. Like stocker, optimizer doesn't need any information from other agents.

After 6th step, the CRATI instance is ready for auctions. It is also worth mentioning that every agent except stocker can have more than one instance but in the

aforementioned example only one instance is given to keep process simple. For greater understanding, booting process has been visualized and can bee seen on Figure 3.

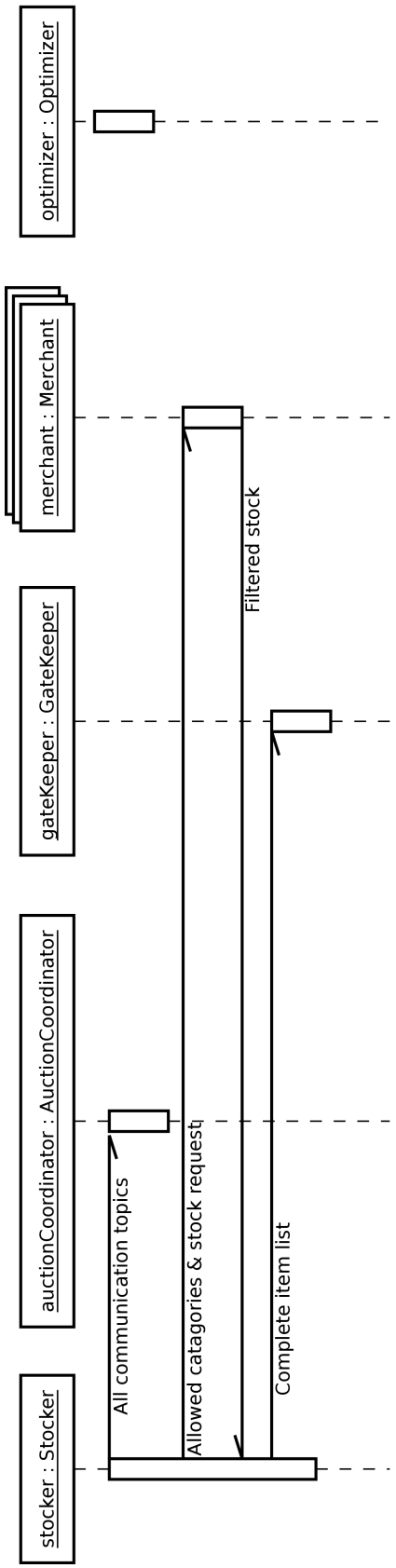


Figure 4: Boot process

When compared to booting, auction process is more straightforward but requires more communication and data processing. In CRATI, every auction is started by the client. CRATI infrastructure is designed to handle multiple auctions simultaneously, regardless of the auction coordinator and optimizer agent counts in the infrastructure. An idealized auction progresses as follows (see Figure 4)

1. Before the auction start, client runs a remote DF search on the target trade server and looks for gate keepers that support auction protocol. As aforementioned, gate keepers enable their auction protocol support after completing initialization.

2. Client selects a random gate keeper and sends a *"hello"* message. If a trade platform runs more than one gate keeper, which is very likely, client selects a random one among them and sends a simple *"hello"* message.

3. The receiving gate keeper agent runs a DF search for auction coordinators inside the platform. If the platform is running more than one auction coordinator, again very likely, selects one of them and writes its address to the reply-to field of the answer message it prepares. Gate Keeper also attaches the most recent item list to the message and sends the prepared answer to the client.

4. Client unpacks the message and gets the item list that contains all items currently on sale complete with their average prices. Client selects the item that wants to buy alongside the quantity of the items. After finishing selecting items, client also enters an upper price bound in terms of percent that represents the maximum amount of money that the client willing to pay for this set of goods. After selecting the upper limit, client sends the message to the auction coordinator that gate keeper selected.

5. Auction coordinator receives the message and extracts the item request list from the message. Stores the upper limit requested by the client in an inner map and divides the request message to categories. Then auction coordinator creates a single message per item category which contains a unique auction ID, categories present in that auction, the category contained in this message

and extracted items from request list of the client in that category. Auction coordinator also writes the end time of the auction to the reply-by date field present in the Agent Communication Language Message format (ACL).

6. One or more merchant receives the message and analyzes the category list, for every category they are in, they wait another auction announcement message. After receiving all messages, a merchant can decide to enter an auction or not. This decision is added to make the infrastructure resemble real-life and the percent of participation possibility is a tunable parameter. If merchant(s) decide to bid to the auction, they check their stocks for the items they have and create a list of items that they can bid for. After creating the list, they bid for every single item with their original list price or with a price which is inside a given, configurable variation percent (e.g. an allowance of 1% variation means the prices can move up to 1%, in either direction, at random) and create another bid that contains every item that they can bid. This combinatorial bid's price is discounted a random amount and between the 60% and 100%, which is again a tunable parameter added for increased realism of the simulation, of the original price of the bid, which is sum of the prices of every item in the bid. After creating the bids, merchant creates a bid envelope, puts the bids into that envelope alongside with its privacy level and seals the envelope. Sealing the envelope causes the envelope not to accept any more bids also creation of a time stamp that marks the sealing time. This time is created by the envelope and not by the merchant, so tampering is impossible. Later bidding merchants send their bid envelopes to the auction coordinator which made the announcement. If a merchant decides not to bid, does nothing.

7. Auction coordinator receives the bid envelopes and adds them to the auction package if the seal time is not after the bid ending time. In order to minimize lost bid envelopes due to internal latencies that may occur in high loads, the bids that are sealed before auction closing time are accepted to the non-dispatched auctions even if the wall time has passed the auction closing time. Since the closed auctions are dispatched to optimizer as soon as they detected, this time generally does not excess 2-3 ms.

8. A behavior inside the auction coordinator checks the auctions periodically and dispatches the closed ones. Before every dispatch, a DF search is made to get the latest list of the optimizers possible and a random one is selected. The complete auction information is sent to the optimizer for winner determination.

9. When optimizer receives the auction information, opens all bid envelopes, stores all bids in a flat storage and anonymizes the bids with generic names. Then optimizer builds the weighted set covering problem model in the solver using the information from bids. After building the model, the solver is invoked to optimize the model. Solver returns the optimized price alongside with the winning bids. This information is used to generate a result object that contains the winner bids, resulting price and the complete bid information sent to the optimizer in its unmodified form. Winner bids are determined by backtracking the actual bids from the generated flat bid storage. In other words, bids are unanonymized after processing. This packaged information is sent to the auction coordinator which sent the message to the optimizer.

10. The auction coordinator receives the results from the optimizer and disassembles the message and extracts the winning bids and final price from it. After extracting winning bids and the price, auction coordinator checks whether the price is within the range that client requested. If yes, messages for the client and merchants are prepared. The winner announcement for the merchant is sent to other parties according to their privacy level preference of the winning agent and this preference can range from nobody to every merchant on the platform. Also a message for the client that contains the results and details is prepared and sent. Postage of these messages mark the end of the auction process. If the price is larger than the client anticipated, a notification message is sent only to the client about the outcome.

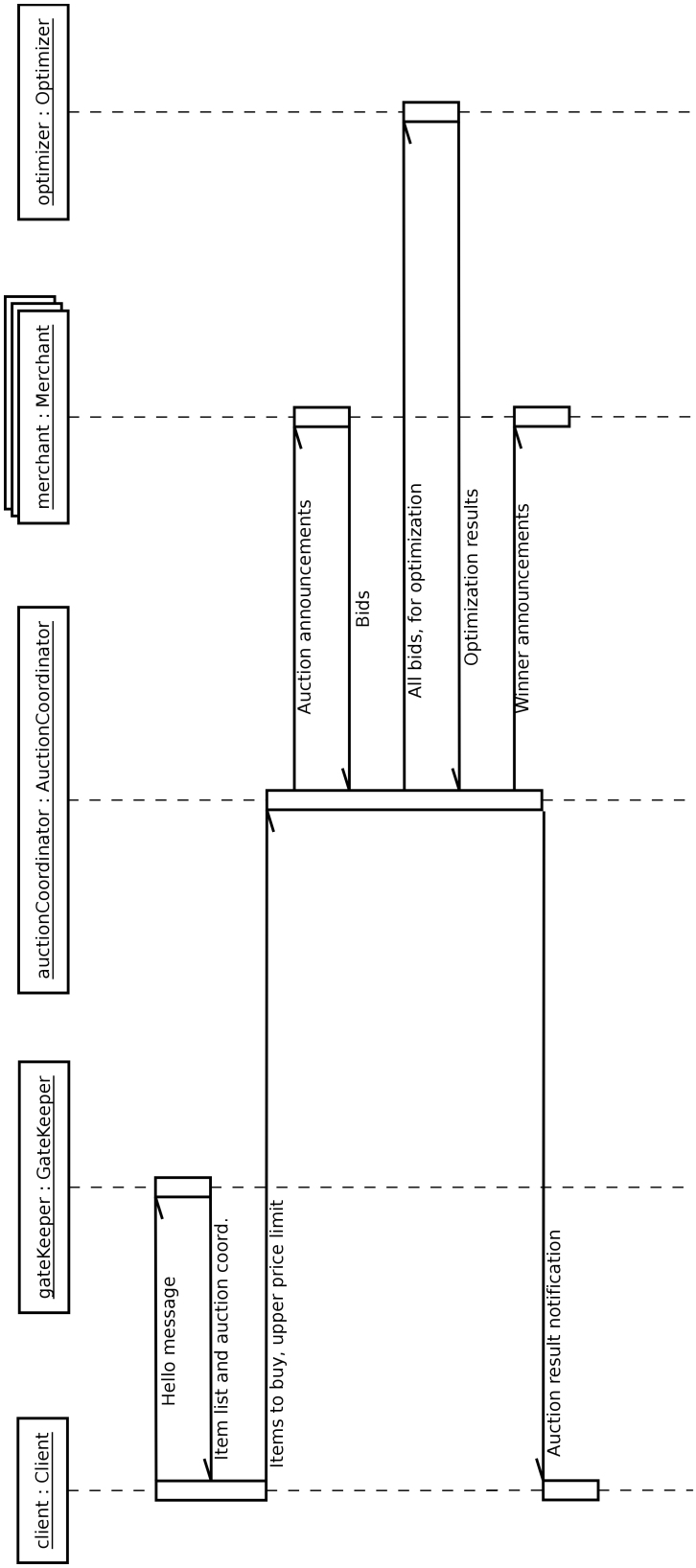


Figure 5: Auction Progress

4.7 Challenges and Solutions

During the implementation of CRATI some challenges surfaced to make system both high performance and scalable. The challenges were tackled by elegant solutions and dirty coding practices are avoided with great caution. The result was a responsive and stable platform that can work with high reliability and minimum overhead.

First challenge was to enable high speed searches in the memory. Since the platform deals with many number of goods and complex data structures which evolve around this big lists (e.g. The complete stock information which contains every item that is present on the infrastructure), it was crucial to enable high performance searches when necessary. This challenge is defeated with the usage of Hash Maps which comes as a standard data structure in Java. Hash Maps are map objects that maps an object to another object. Hash Map object can be parametrized for further programming ease and security & all of the Hash Map structures in the CRATI is parametrized because of these two reasons. The performance of Hash Maps are impressive with an average case of near O(1) complexity and O(n) complexity at worst case. The effect of using Hash Maps with string keys, which enabled us to construct keys on the fly rather than hard coding them into code, instead of linear data types such as vectors and arrays was a code base that doesn't iterate for any object which it can directly access with a single call.

Next challenge in making a high performance multi-agent system is to ensure that every agent can decide their next action as quickly as possible. A pleasant side effect of this feature is reduced CPU loads because fast decisions are only possible with fast algorithms when you have a single thread. The solution was to create a system that we call Hierarchical Fast Response Engine that uses two nested Hash Maps on the receiver side and the message performative and a String that is embedded into every message that we send. Another problem encountered during the implementation of the system was the limitations of the standard message container. A standard ACL Message contains a single payload slot and either can carry an object or a string. To work around this limitation a data structure that we call Common Message Wrapper has been implemented. This simple data structure carries a two payload slots. A mandatory string message and an optional object payload.

Since ACL Messages can carry objects, this data structure can be easily piggybacked to the ACL Message. This extension of the message format made carrying a string identifier with every message possible and implementation of this decision engine as a result.

Decision engine works as follows. When agent receives a message, it feeds the performative of the message to the first Hash Map and receives another Hash Map as a result. Feeding the string payload inside the extended message container to the received Hash Map returns an integer to the agent. This integer is the transition value that we need. Setting this value as the transition value for the behavior and storing the received message in the shared Data Store finishes deciding process. Storing the message in a place where every behavior can access is crucial since generally behaviors make use of the information sent alongside with the message. When access times of Hash Maps are taken into account, performance advantage of this approach is clearly visible. Also since messages are classified according to both their performative and string contents, system is virtually collision free and freely extendible.

One of the hardest challenge was to use Choco as the solver. Initial impression on the Choco solver implied that it can solve any constraint satisfaction problem, including the mixed integer class. Unfortunately during the implementation it was seen that this was indeed not possible and Choco was either able to solve problems that contain only integer or only real variables. Instead of using another solver a simple and non-damaging solution has been implemented. This solution involves fixing the scales of the prices to the two digits after the integer part and multiplying these numbers by 100 to achieve integer versions. These integers are fed into Choco instead of real numbers and the result is divided by 100 again to obtain real result. This approach also bring more advantages when compared with a mixed integer solver because the solver only works with integers and at the end the result's accuracy is not compromised. It's also noteworthy that no ordinary selling mechanisms or markets honor more than two digits after the integer in the prices and the system is designed with this assumption. If ever needed in the future, it's easy to enhance the system precision to any desired value.

Since the performance of the CRATI platform is essential, unnecessary messaging and processing inside the trading platform must be avoided. To minimize the message traffic and wasted processing power, communication channels concept has been introduced into the system during the design stage. Communication channels are subscription channels that use the topic subscription feature of the JADE platform and works as follows

- When a merchant boots up, sends its stock information to the stocker agent.
- Stocker agent analyzes the stock of the new merchant and determines the categories of the items that the merchant provides. .
- Then stocker sends a list of topics to the new merchant. These topics only cover the items that are present in the merchant's stock.
- Merchant agent then subscribes to these channels and waits for auction announcements.

With this mechanism, if a merchant have items in 10 categories, it can receive at most 10 messages but will never receive a message for an auction if the merchant has no items related to this auction. This way both message traffic is reduced because none of the merchants receive spam auction notifications and no processing power is wasted because merchants that do not have any items suitable for auction doesn't use CPU cycles to check their stocks to see whether they can participate or not.

The last challenge we tried to tackle was implementation of a mobile client agent which is able to run on mobile devices that support Java Runtime Environment. While JADE supports this type of implementation very well and the code is also designed in an easily portable way, lacking of expertise in this mobile device implementation and lack of time resulted in shelving of the feature.

4.8 Building an Application Using CRATI Infrastructure

While CRATI provides all of the crucial components and most of the components that are needed in a real-world application, it is not a fully assembled and ready to use platform. In order to implement a complete application powered by CRATI,

some steps must be taken. Below, these steps are summarized alongside with their brief explanations.

- **Implementation of a complete client:** While the client agent in the CRATI infrastructure provides all of the basic functionality required, it acts more like a simulation helper. The developed client lacks mobile abilities, doesn't have a GUI and continuously requests items with randomly selected goods lists. A complete client that has a GUI and provides user the ability to select items of his/her choice with the ability to run on mobile devices is necessary.

- **Implementation of a better merchant:** Again, the merchant in the CRATI infrastructure provides all the foundation for a complete merchant but this foundation is insufficient for real-life applications. A complete agent may need a better mechanism to decide to enter the auctions and a better analysis system, perhaps a statistical one, to calculate the variations or discounts on the single or combinatorial bids and a real B2C system that can check whether the merchant can provide the given goods in the quantities requested. Also a GUI for the merchant that can control the different aspects or provide control or even manual conductance of trade of the agent's behavior can greatly improve the usability and adaptability of the CRATI infrastructure by the merchants since the sellers behind the agents can tailor their agents according to their needs with ease.

- **Implementation of a better stocker price calculation:** Currently the prices that are presented to the clients are the true mean prices calculated in a straightforward way. A better approach is to predict the prices more accurately via analyzing the auction bids and the results.

- **Introduction of secure communication between client and trade server:** Currently CRATI does not use encryption between client and the trade server but JADE already has an encryption subsystem. Therefore the only requirement is the utilization of this subsystem rather than a complete implementation.

If the CRATI platform is improved in order to fix these shortcomings of the

infrastructure, CRATI can be used as a real-world trading platform with great ease. It is worth mentioning again that CRATI already provides the fundamental functions required and only need is the improvements that adapt the infrastructure to the real-world instruments that are used in trading.

4.9 Benchmarks and Performance of the CRATI Platform

In order to understand the performance of the CRATI platform, some benchmarks with different problem configurations has been conducted. These benchmarks both included feasible and infeasible number of goods with different number of merchant and client agents and different market configurations. In all of the testing, some of the parameters have been fixed to make benchmarks meaningful. This parameters are other agents' counts, the time between auctions, the bidding scheme and the hardware platform that the auction has been performed. Details of the fixed parameters are as follows

- **Number of Optimizers:** 1
- **Number of Auction Coordinators:** 3
- **Number of Gate Keepers:** 2
- **Number of Stockers:** 1

In order to make the benchmarking more realistic, some parameters are added to the system. These are sleep time between auctions, single bid price variations, random combinatorial bid discounts and an optimizer timeout. Optimizer timeout is the time that the optimizer stops processing the problem at hand if it can't find the optimal solution. Another reason for introducing this timeout is the problems encountered during multi-instance execution of the solver which will be detailed later. Values of the parameters and brief explanations are below

- T**ime between auctions:** 1 minute. This means every client waits for 1 minute after requesting items and even if the client doesn't receive the results of its earlier request, makes another request.
- **Bidding Scheme:** Every agents generates a single bid to every item they have

and also generate a single combinatorial bid that contains every item they have generated single bids for. Combinatorial bids have lower prices than the sum of individual bids.

- **Configured price for combinatorial bids:** Between 60% and 100% of the total price of the items in the combinatorial bid.
- **Single bid price variation:** 1%. This means every single bid will have a price between 99% and 101% of the original price of the item.
- **Optimizer timeout:** 1 minute. This means optimizer stops processing problems that cannot find any solution for 60 seconds.
- Market configuration: All of the items used in the benchmarking process have been taken from Kangurum [28], which an internet marketplace which works with Migros markets and maintains a browsable catalog on their website under Migros Sanal Market category.

The tests are performed on a 2.4GHz quad core computer with 4GB of high performance DDR2-800 RAM installed. The operating system of choice was GNU/Linux because of its performance and the developer's experience. All tests conducted on latest version of Java to date, Java Standard Edition, Version 6, Update 20.

During benchmarking process, the hardness of the problem and limitations of some components of the platform became evident. During testing, JADE platform and the implemented agents have behaved very reliably but Choco solver refused to work reliably when run in more than one agent in a parallel fashion. When executed in more than one agent, Choco solver exhibited loss of speed and memory-leak like behavior but if the optimizer agent is not run in more than one instance, none of these adverse effects are observed and Choco ran in a stable manner without allocating excess memory or losing performance.

While it was impossible to run optimizers in parallel to solve more than one winner determination problems at the same time, Running a single solver doesn't cripple the infrastructure's ability to fulfill its functional requirements in any way. Since JADE's

messaging system stores the messages in agents' inboxes and never lose them, this inbox mechanism acts as a practical queue. Since there is one solver, every auction coordinator sends their optimization requests to only solver and they're essentially queued in the inbox of the optimizer. Since optimizer always checks its inbox for jobs when it is not busy with optimizing a problem, it sequentially handles all optimization requests one by one. As a result every auction that is conducted in the CRATI infrastructure is processed and finalized. The only side effect of this situation is the increased auction completion duration.

A problem of the Choco surfaced during the testing surfaced is the scarcity of the public documentation. During initial benchmarking runs the performance degraded too quickly. Further investigation of the documentation yielded some fine tuning parameters which improved the performance of the Choco dramatically but, unfortunately not for all cases so, it was impossible to eliminate rogue cases completely. To complete testing and measure the feasibility of the tested cases, a 60 second time barrier was also introduced into the Choco solver. With this, both benchmarks ran without interruption and feasibility studies have conducted with the problem combinations. Also it is worth mentioning that, the rogue combinations are practically impossible to eliminate completely in real world. Even if Choco solver was able to run in multiple instances, the need of introduction of such time limit would not go away and the system will have the same time limit on optimizers in order to prevent system from coming to an halt.

Before introducing the results, it's important to understand what market configuration is. During testing, as aforementioned, two market configurations have been used and they can be summarized as follows

- **Disjoint Seller Market:** Every item on the market can be obtained from only one seller. While it's not realistic and practical to have a market with every item only suppliable by a single merchant, This configuration yields easier problems which provides a comparative base for the speed of the infrastructure.

- **Overlapping Seller Market:** An item can be obtained from many sellers.

Below are the results of the benchmarks that have been conducted.

CRATI Benchmark Results – Disjoint Market – 1 Client				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	67,7	8,1631	0,0892
50	2	67,8	8,2961	0,1973
100	2	107,4	8,4200	0,3142
20	5	72,7	8,1061	0,0615
50	5	69,6	8,2648	0,1852
100	5	97,7	8,3457	0,2689
20	10	70,6	8,1733	0,1033
50	10	92,1	8,2481	0,2044
100	10	136,3	9,6956	3,8660
20	20	136,9	8,1379	0,0805
50	20	88,4	9,3450	1,2694
100	20	122,9	12,3460	3,9003

CRATI Benchmark Results – Disjoint Market – 2 Client				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	72,2	8,2038	0,0927
50	2	84,8	8,3569	0,1813
100	2	84,1	8,5044	0,2356
20	5	87	8,1370	0,0670
50	5	94,2	8,3941	0,1893
100	5	101,5	26,4384	0,3119
20	10	88,1	8,1787	0,0964
50	10	76,6	8,4260	0,2104
100	10	106,5	8,6254	0,3542
20	20	133,7	8,2846	0,1615
50	20	103,5	10,9003	1,8749
100	20	137,6	36,7415	2,7932

Table 2: Disjoint Market 2 Client Benchmark Results

CRATI Benchmark Results – Disjoint Market – 3 Clients				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	71,9	8,2353	0,0661
50	2	90,2	8,3510	0,1718
100	2	86,7	8,6559	0,2320
20	5	87,8	8,2069	0,0780
50	5	72,7	8,3560	0,1502
100	5	92,8	8,5310	0,2654
20	10	88,4	8,2728	0,0983
50	10	87,8	8,4295	0,1549
100	10	123,3	8,4985	0,3264
20	20	111,9	8,4463	0,1222
50	20	154,8	8,8828	0,5132
100	20	124,9	13,2996	1,9732

Table 3: Disjoint Market 3 Client Benchmark Results

CRATI Benchmark Results – Disjoint Market – 4 Clients				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	78,3	8,3271	0,0697
50	2	72,1	8,3896	0,1256
100	2	133,5	9,2452	0,2469
20	5	69,8	8,3548	0,0800
50	5	69,8	8,3856	0,1133
100	5	88,8	8,5891	0,2399
20	10	87,4	8,4303	0,0929
50	10	108,2	8,5781	0,1532
100	10	126,7	138,2867	2,3204
20	20	200	8,4252	0,1198
50	20	163,6	13,1649	1,9013
100	20	152,1	40,0954	2,4305

CRATE Platform Discrete Market Benchmark				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	77,9	8,2919	0,0521
50	2	70,5	8,4750	0,1425
100	2	88,2	8,4636	0,1840
20	5	71,1	8,2834	0,0747
50	5	80,8	8,6607	0,1564
100	5	116,5	8,8299	0,3351
20	10	126,8	8,6637	0,3548
50	10	689	18,2678	5,0534
100	10	213,4	10,6056	0,5073
20	20	196,4	9,2559	0,4550
50	20	2765,2	8,5919	0,1333
100	20	2860,8	122,6308	76,7865

Table 5: Disjoint Market 5 Client Benchmark Results

CRATI Benchmark Results – Overlapping Market – 1 Client				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	88,4	8,1414	0,0826
50	2	86,9	8,2353	0,1601
100	2	Not possible to solve under 60 seconds		
20	5	69,4	8,5195	0,4541
50	5	92,8	21,7963	13,7074
100	5	Not possible to solve under 60 seconds		
20	10	88,1	20,6501	12,5885
50	10	Not possible to solve under 60 seconds		
100	10	Not possible to solve under 60 seconds		
20	20	174,4	39,6513	31,4355
50	20	Not possible to solve under 60 seconds		
100	20	Not possible to solve under 60 seconds		

Table 6: Overlapping Market 1 Client Benchmark Results

CRATI Benchmark Results – Overlapping Market – 2 Clients				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	69,7	8,1259	0,0735
50	2	82,1	8,3458	0,1331
100	2	Not possible to solve under 60 seconds		
20	5	65,4	8,6533	0,0390
50	5	101,4	50,7030	20,1544
100	5	Not possible to solve under 60 seconds		
20	10	93,2	10,1266	1,5847
50	10	Not possible to solve under 60 seconds		
100	10	Not possible to solve under 60 seconds		
20	20	188,7	41,6435	17,6165
50	20	Not possible to solve under 60 seconds		
100	20	Not possible to solve under 60 seconds		

Table 7: Overlapping Market 1 Client Benchmark Results

CRATI Benchmark Results – Overlapping Market – 3 Clients				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	70	8,2210	0,1016
50	2	86,7	8,3856	0,1530
100	2	Not possible to solve under 60 seconds		
20	5	71,3	8,8732	0,4085
50	5	71,9	67,7674	24,3585
100	5	Not possible to solve under 60 seconds		
20	10	97,6	33,0261	14,6262
50	10	Not possible to solve under 60 seconds		
100	10	Not possible to solve under 60 seconds		
20	20	134,2	155,7380	20,1544
50	20	Not possible to solve under 60 seconds		
100	20	Not possible to solve under 60 seconds		

Table 8: Overlapping Market 3 Client Benchmark Results

CRATI Benchmark Results – Overlapping Market – 4 Clients				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	70,2	8,3427	0,0815
50	2	77,7	8,6261	0,1582
100	2	Not possible to solve under 60 seconds		
20	5	73,3	8,6921	0,1933
50	5	123,4	186,1118	39,7644
100	5	Not possible to solve under 60 seconds		
20	10	103,3	31,3433	10,3504
50	10	Not possible to solve under 60 seconds		
100	10	Not possible to solve under 60 seconds		
20	20	160,9	135,0582	17,3588
50	20	Not possible to solve under 60 seconds		
100	20	Not possible to solve under 60 seconds		

Table 9: Overlapping Market 4 Client Benchmark Results

CRATI Benchmark Results – Overlapping Market – 5 Clients				
Item Size	Merchant Count	Maximum Memory Load (MB)	Auction Turnaround Time (S)	Optimizer Time (S)
20	2	71,9	8,1274	0,0623
50	2	89,9	8,2899	0,1297
100	2	Not possible to solve under 60 seconds		
20	5	75,6	8,3960	0,2326
50	5	115	81,5833	4,4807
100	5	Not possible to solve under 60 seconds		
20	10	100	28,1738	11,7493
50	10	Not possible to solve under 60 seconds		
100	10	Not possible to solve under 60 seconds		
20	20	175	142,1140	26,2769
50	20	Not possible to solve under 60 seconds		
100	20	Not possible to solve under 60 seconds		

Notes: Solvability of 5 Merchant, 50 Item configuration is 45% and 20 Merchant, 20 Item configuration is 77.3% . Solvability is the ratio of problems that can be solved under 1 minute time limit to all problems ran in the benchmark. Time limit is the aforementioned tine limit introduced to the Choco solver. If the ratio of solvability is under 40% configuration is considered infeasible (i.e. not practical to solve and not worth the extra time required). Also, if a problem is infeasible in any client count, it is considered infeasible for all client counts. The spikes seen in any table is the result of inclusion of rogue combinations that cause solver to timeout and give-up.

The benchmark figures have been calculated as follows. The Maximum memory load have is the value that the system have been reported after system solved at least 10 problems. Auction turnaround time and optimizer time has been calculated from the mean of five solved problems. Otherwise noted, all problems have been solved successfully for a given merchant-item configuration. Its important to note that the increased auction turnaround time is the result of using a single optimizer. When the optimizer doesn't crash in multiple optimizer configurations, the auction turnaround time is stable and can be roughly calculated as $(optimizer\,time + 9)\,seconds$ but, as aforementioned, this configuration exhibits a memory-leak like behavior and crashes the platform.

Latency, another important aspect of such systems was consistent throughout testing, even under high load. JADE framework didn't show any signs of slow down. Stability of the JADE framework was consistent regardless of the number of Merchant or Client agents which impose the biggest load after the optimizer.

Memory and CPU usage of the JADE platform was also negligible and increased minimally with added agents and this leaved the free space much desired by the optimizer. Similarly, the reported memory problems on the optimizer agent was tracked back to the Choco solver, not JADE.

As a result, when coupled with informed design decisions and identification of the potentially time consuming processes inside the CRATI platform and optimizing them before implementation resulted in a very robust, scalable and high performance platform. Unfortunately, most important component of the whole platform, the solver was performed sub-par when integrated to this complex and sophisticated framework because of the unforeseen reasons.

CHAPTER 5

CONCLUSIONS & FUTURE WORK

In this thesis, a method of solving combinatorial reverse auctions by converting the problem to weighted set covering problem has been investigated and introduced. Also, a combinatorial reverse auction platform which builds upon this foundation with real life applicability and fast trading capability has been investigated, designed, implemented and introduced. The product, as named CRATI, has been implemented using completely free, open source agent-based technology which leverages parallel and distributed computing, multi-agent system and mobility which are used broadly and enable many features that are important, needed and taken granted today. The developed platform tested against quite reasonable and feasible market configurations and performed in a satisfactory way. As aforementioned the benchmarks showed the point that agent-based computing has come to and the power of agents which can achieve many complex tasks in a simple way quite reliably and without the hurdles of thinking all of the lower level details. This thesis also exposed some of the difficulties for today's software development process. Especially the behavior of Choco solver proved that the existing software should be designed with multithreaded environments in mind whenever possible otherwise it may behave outside the intention. Another conclusion of this thesis is the feasibility of attacking mathematically hard problems up to a certain point of complexity. As the results shown here are obtained with an outdated desktop PC, today's server processors and server hardware is much more powerful and more capable in these tasks. On the other hand, when the Choco solver has been pushed to its limits with benchmarks with accepted difficulties [29], it was not possible to obtain any solutions in a practical time. This finding again shows us the hardness of solving NP-complete problems.

While the infrastructure performs well, it is not free of shortcomings. For example, Choco's behavior under multi instance situations is less than desirable however, stability of the solver in the single instance scenario combined with the performance of the solver in the feasible configurations mostly compensates the situation and keep the platform usable and functioning. Other improvements can be implementation of the mobile client alongside with an advanced, AI backed decision engine for Merchants that can decide whether to enter an auction or decide to the prices of the bids per auction basis which will make CRATI much of a real trading platform than a simulation sandbox as it currently seems. Finally, to improve scalability even more, small glitches that prevent CRATI from messaging between two physical containers can be cleaned up and CRATI can become a totally distributed, first class trade platform.

APPENDIX

CONVERSION OF COMBINATORIAL REVERSE AUCTION TO WEIGHTED SET COVERING PROBLEM

As aforementioned in the thesis, CRATI determines the winners by converting the auction problems to weighted set covering problems and solving converted problems. In this appendix, the conversion process will be explained step by step and in great detail using an example.

Consider that the client wanted to buy the following items:

- 5 bottles of Uludağ Gazoz with an estimated price of 3.0 price units,
- 3 packs of ETİ Çubuk Kraker with an estimated price of 0,75 price units,
- 2 packs of Lay's potato chips with an estimated price of 2,38 price units,
- 4 packs of Balocco Wafers with an estimated price of 13 price units,
- 6 cans of Red Bull energy drink for 20,94 price units.

Let us define the items as i_1, i_2, i_3, i_4, i_5 respectively and define a universal set I where $\forall i \in I$. On the other hand let us assume that there are three merchants in the marketplace at that specific instance and define them as m_1, m_2, m_3 . The stocks of the merchants are as follows:

- m_1 has the items . i_1, i_3, i_4 ,
- m_2 has the items . i_2, i_3, i_5 ,

- m_3 has the items . i_1, i_2, i_4, i_5 .

When the auction has been announced, merchants have generated the following bids. It is worth reminding that every merchant will generate single bids for every item requested by client and can be supplied by merchant, alongside with a combinatorial bid that spans all items requested and the merchant can supply. Also it is worth reminding that single bids can vary up to 1% in any direction (positive or negative) and the combinatorial bids can have discounts up to 40%. Consider that the bids are given by the merchants as follows:

- Merchant m_1
  - Bid for item i_1 with a price of 2,98 price units (-0,77%),
  - Bid for item i_3 with a price of 2,56 price units (+0,77%),
  - Bid for item i_4 with a price of 14,21 price units (+0,9%),
  - Bid for items i_1, i_3, i_4 with a price of 12,77 price units (-17%).
- Merchant m_2
  - Bid for item i_2 with a price of 0,74 price units (-0,25%),
  - Bid for item i_3 with a price of 2,37 price units (-0,01%),
  - Bid for item i_5 with a price of 20,73 price units (-0,98%),
  - Bid for items i_2, i_3, i_5 with a price of 20,94 price units (-13%).
- Merchant m_3
  - Bid for item i_1 with a price of 3,198 price units (+0,66%),
  - Bid for item i_2 with a price of 0,78 price units (+0,41%),
  - Bid for item i_4 with a price of 14,53 price units (+0,38%),
  - Bid for item i_5 with a price of 21,30 price units (+0,17%),

- Bid for items i_1, i_2, i_4, i_5 with a price of 19,11 price units (-31%).

When the CRATI's Optimizer agent receives the bids, analyzes them and generates some variables and constraints. First let us consider the variables.

Optimizer generates a integer variable with an interval of $[0,1]$. These variables act as boolean bid selectors. In this case these variables will be $b_1, b_2, \ldots, b_{13}$, one for every bid given. The set of all bid selectors is denoted by B where $b_i \in B$

Optimizer will also generate an integer variable for every bid price and a price variable, which is again integer. Remember that the prices are multiplied by 100 in order to obtain their integer counterparts. The variables will be as follows:

- $w_1 = 298$,
- $w_2 = 256$,
- $w_3 = 1421$,
- $w_4 = 1277$,
- $w_5 = 74$,
- $w_6 = 237$,
- $w_7 = 2073$,
- $w_8 = 2094$,
- $w_9 = 3198$,
- $w_{10} = 78$,
- $w_{11} = 1453$,
- $w_{12} = 2130$,
- $w_{13} = 1911$.

Last but not least, optimizer generates another integer variable called cost with an interval $[0, \sum_{i=1} w_i]$. This is the optimization variable for the optimizer and tries to minimize it.

After generating all variables, optimizer will go ahead with constraint generation. Optimizer will generate two types of constraints. One for item coverage and other one for the price minimization.

The item coverage constraints are inequalities created in a one per item fashion. They force the solver to obtain every item at least from one bid. The previously generated integer variables will be utilized here. For this problem, optimizer will generate 5 constraints as follows:

- Since i_1 can be obtained from bids b_1, b_4, b_9, b_{13} the constraint is $1 \leqslant b_1 + b_4 + b_9 + b_{13}$,
- Since i_2 can be obtained from bids b_5, b_8, b_{10}, b_{13} the constraint is $1 \leqslant b_5 + b_8 + b_{10} + b_{13}$,
- Since i_3 can be obtained from bids b_2, b_4, b_6, b_8 the constraint is $1 \leqslant b_2 + b_4 + b_6 + b_8$,
- Since i_4 can be obtained from bids b_3, b_4, b_{11}, b_{13} the constraint is $1 \leqslant b_3 + b_4 + b_{11} + b_{13}$,
- Finally, since i_2 can be obtained from bids b_5, b_8, b_{10}, b_{13} the constraint is $1 \leqslant b_5 + b_8 + b_{10} + b_{13}$.

After generating coverage constraints, optimizer generates the cost formula using the bid and weight variables generated. Optimizer also equates this formula to the cost variable generated before. The closed form of the formula is

$$cost = \sum_{n=1}^{|B|} w_i \cdot b_i$$

The explicit form is given below

$$cost = (298 \cdot b_1) + (256 \cdot b_2) + (1421 \cdot b_3) + (1277 \cdot b_4) + (74 \cdot b_5) + (237 \cdot b_6) + (2073 \cdot b_7) + (2094 \cdot b_8) + (3198 \cdot b_9) + (78 \cdot b_{10}) + (1453 \cdot b_{11}) + (2130 \cdot b_{12}) + (1911 \cdot b_{13})$$

When all variables and constraints are generated, they are sent to the Choco solver within a container called Model. After sending the model to Choco, minimization of the *cost* variable is requested. Then, Choco minimizes the *cost* variable and returns the model back with all variables set. Winning bids are determined by reading all bid selector variables and getting the ones with value of 1. Auction closing price is determined by the *cost* variable. Later these variables are further analyzed and the owners of the bids are obtained. As the last step, this information is sent back to the auction coordinator.

REFERENCES

[1] V. Krishna: Auction Theory, Elsevier, 2003.

[2] Imandi: The Expert Marketplace, http://www.imandi.com (Last access: July 2010)

[3] Sorcity: Sourcing and Higher Velocity, http://www.sorcity.com (Last access: July 2010)

[4] R. Epstein, L. Henríquez, J. Catalán, G. Y. Weintraub, C. Martínez, F. Espejo: A combinatorial auction improves school meals in Chile: a case of OR in developing countries. International Transactions in Operations Research, 11 (2004) pp. 593–612 (2004).

[5] HedgeHog: E-Procurement Solutions, http://www.hedgehog.com (Last access: July 2010)

[6] P. Cramton, Y. Shoham, R. Steinberg: Combinatorial Auctions, The MIT Press, 2006.

[7] S. J. Rassenti, V. L. Smith, R. L. Bulfin: A Combinatorial Auction Mechanism for Airport Time Slot Allocation. The Bell Journal of Economics, Vol. 13, Issue 2, pp 402 – 417, (1982).

[8] eBay: New & used electronics, cars, apparel, collectibles, sporting goods & more at low prices. http://www.ebay.com (Last access: July 2010)

[9] GittiGidiyor: Türkiye'nin en işlek alışveriş merkezi http://www.gittigidiyor.com (Last access: July 2010)

[10] B. Hudson, T. Sandholm: Generalizing Preference Elicitation in Combinatorial Auctions, Second International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1014-1015 (2003)..

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms 2E, The MIT Press, 2001.

[12] R. Bar-Yehuda, S. Even: A Linear-Time Approximation Algorithm for the Weighted Vertex Cover Problem, Journal of Algorithms, Vol 2, pp 198-203 (1981).

[13] İ.Cereci, H. Kılıç: CAWP: A Combinatorial Auction Web Platform: ICE-B 2010 (International Conference on e-Business), Athens, Greece, 2010

[14] K.L. Brown,, Y. Fujishima, Y. Shoham: Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (1999).

[15] T. Sandholm, S. Suri, A. Gilpin, D. Levine: CABOB: A Fast Optimal Algorithm for Combinatorial Auctions: Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) (2001).

[16] M. Woolridge, N. R. Jennings: Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, Vol. 10:2, 1995, pp. 115-152 (1995).

[17] F. Bellifemine, G. Caire, D. Greenwood: Developing Multi-agent Systems with Jade, Wiley, 2007.

[18] H. V. D. Panunak: Industrial and Practical Applications of DAI: Multi Agent Systems (E.D. G. Weiss), pp. 377-421, The MIT Press, Cambridge, MA (1999).

[19] Choco Whitepaper, http://www.emn.fr/x-info/choco-solver/lib/exe/fetch.php?id=presentation&cache=cache&media=pdf:choco-presentation.pdf (Last access: July 2010)

[20] V. Conitzer, T. Sandholm, P. Santi: Combinatorial Auctions with k-wise Dependent Valuations: American Association for Artificial Intelligence, 2005, pp. 248-254 (2005).

[21] C. Lund, M. Yannakakis: On the Hardness of Approximating Minimization Problems: Journal of the Association for Computing Machinery, Vol. 41, No 5, pp 960-981 (1994).

[22] NASA: National Aeronautics and Space Administration http://www.nasa.gov (Last access: July 2010)

[23] Assembla: Bug tracking tools, project management & collaboration tools. http://www.assembla.com (Last access: July 2010)

[24] Apache Subversion: Enterprise-class centralized version control for the masses. http://subversion.apache.org (Last access: July 2010)

[25] Trac: Integrated SCM & Project Management http://trac.edgewall.org (Last access: July 2010)

[26] Eclipse: Universal IDE http://www.eclipse.org (Last access: July 2010)

[27] R. G. Smith: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver: IEEE Transactions on Computers, Vol c-29, No. 12, pp. 1104-1113 (1980).

[28] Kangurum: An internet supermarket. http://www.kangurum.com.tr (Last access: July 2010)

[29] Benchmarks with Hidden Optimum Solutions for Set Covering, Set Packing and Winner Determination by Ke Xu. http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/set-benchmarks.htm (Last access: July 2010)