An Agent-based Grid Services Exchange
Thesis @ IMM.DTU , March 2004

Ola Mortensen (c973684) and
Martin Andersen (c973449)

March 12, 2004

IMM-THESIS-2004-17

Technical University of Denmark
Lyngby

Preface

This report is authored by Martin Andersen and Ola Mortensen and is the result of the *An Agent-based Grid Services Exchange* master thesis project at the Technical University of Denmark (DTU), at which both authors are students. The project has been carried out in the period between October 7th 2003 and March 12th 2004 and fulfils the requirements for obtaining a Master of Science in Engineering degree at DTU. This thesis was made at the institute of Informatics and Mathematical Modelling (IMM) at DTU under the guidance of Robin Sharp.

Lyngby, March 12th, 2004.

Ola Mortensen, s973684

Martin Andersen, s973449

Acknowledgements

Our thanks to our supervisor Robin Sharp for his guidance, time and support. Your input and experience has been highly appreciated.

Ola
I would like to thank my family, Gun, Björn and Johan for their support through my studies. To my girlfriend, Johanna, thanks for your love and for always being considerate. And to Martin, as always, it has been a pleasure working with you.

Martin
I would like to thank my family for supporting me, morally, emotionally and financially. Also, Kamilla, thank you for your love and time. And also thanks to the Swedish cook, Ola, for being my partner in so many projects, and for being a good friend.

Abstract

Grid systems are utilised for performing computational and storage tasks, on a greater scale. However, most Grid systems are comprised of big cluster systems, and are used privately within organisations or in the scientific research community. The Grids provide services as a pool of resources available to the user of the Grid.

This thesis proposes a system where commercial and non-profit Grids can combine their services by exchanging them and selling them to each other, through a network comprising of these Grids.

The Grid owners and buyers of services will negotiate prices and resources through the use of electronic agents, meeting at the local marketplace system. The marketplace systems form a hierarchical network of hubs where the users of the system meet and exchange or sell their resources locally.

The thesis will address the negotiation, using argumentative negotiation techniques to form a framework. Suggestions on the formation of utility functions and how to price the resources are given, as well as the model for the system, and a proposed method of measuring trust within the system.

Keywords: Grid Services, Electronic Agents, Negotiation, Economy, Utility functions, Pricing, Marketplaces.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The scientific community has had a need for great computational power for a long time, and has been a major purchaser of big homogeneous cluster systems, to achieve the processing power needed for simulations and calculations. However, the community has recently put more and more effort in going for the relatively more inexpensive heterogeneous Grid systems.

These efforts has been mostly concentrated within the scientific community, but the commercial sector is beginning to launch projects involving these technologies. As the need for performing big calculations rise, we need more of these Grid systems available, and to motivate these systems, we imagine that companies *provide* services for a fee, to other companies. We suggest the *Agent-based Grid Services Exchange*, which we name *WEGAS* (Worldwide Exchanging Grids Agent-based System).

The fundamentals of WEGAS is the Grids, which provide computational, storage and other services for other users. These services can be anything from disk storage to application licence leases. We introduce the concept of *payment* for these services, so the Grid owners can sell their services. Also, Grid owners are often companies who use their Grid for computations themselves, but in times of inactivity sell the unused available services.

The payment can proceed in many different ways, with variable prices or even with services in return. The whole idea is to create an *exchange*, somewhat like the stock exchange, where services are posted by the sellers, so that they can be bought by other users.

For propagating this exchange structure, we will use *electronic agents*, as they have many desirable properties, and can act upon the wishes of the users who launch them. The agents handle the negotiation, which should imitate how normal people would negotiate when bargaining for groceries on the town square.

The project involves many additional areas, apart from the previous mentioned negotiation, electronic agents and Grid technology. The proposed exchange will also touch areas like *electronic contracts*, *conflict management*, *trust* and *security*.

In this report, we will focus on the negotiation and pricing areas. We will discuss the other mentioned fields, but not go into very high detail about them. We feel we need to mention them for completeness reasons, though.

We will try to implement prototypes of parts of WEGAS, and test some of the concepts of the system.

To illustrate how this system could be used in a real life scenario, lets explore an example:

> A biotechnology research company is exploring new ways to fight the cancer, and for this, they need to discover proteins with certain properties. The method of finding these proteins require a form of brute force method, as every possible molecule has to be generated, and then analysed according to some well-defined mathematical and biological equations.
>
> The method require a lot of processing power, but the company does not have the funds to buy big cluster systems to compute the analysis of the the proteins. Therefore, they would like to have these computations made on a external system.

Considering this example, it would be very handy for such a company to be able to buy access to certain *Grid services*, for the limited time they need the computations. This example also stresses the fact that the transactions made within WEGAS must be very discreet and secure, as the results of the calculations could be subject to patent, and it would be disastrous if they were to leak before patenting.

1.1 Related work

This project involves in many technological areas, which are presented in different forms in the literature.

Grids have been the focus of much research the later years. Especially the proposal of the Open Grid Services Architecture (OGSA) [13] is interesting for this project, as Grid resources are treated as services, which harmonise nicely with our proposal.

The concept of Grids also comes from the electrical grids we have today, and an example of grids where services (power) is exchanged for money is Nord Pool (The Nordic Power Exchange). This exchange is interesting, because it operates with variable prices, and is a big real life production system, and many parallels can be drawn between the systems.

Several efforts have been made with electronic negotiation using agents [34, 23, 27, 36]. Agents have many nice properties, like autonomy, the setting of goals, acquiring of goals, and intelligence. The use of agents also makes the negotiation more human-like.

The only research effort we found to be similar to our proposed system, is the research made by Giddy, Abramson and Buyya [3]. The paper explores some of the concepts connected with *Economy Grids* as they are named. The paper suggest an architecture called GRACE (GRid Architecture for Computational Economy) which can be used for posting resources, charging for them and applying access policies.

The paper also lists a lot of ideas for negotiation, pricing, accounting, protocols and payment mechanisms. It ends with some performance simulations and experimentations regarding scheduling and trading. The paper is very brief about the subjects, and clearly indicates that the fields needing research are vast.

1.2 Report structure

The report is divided into 5 main chapters, 2-6. The second chapter, **Model**, will address the modelling of the WEGAS system. This includes the definitions of the various concepts used throughout the report, as well as how the actual system are organised.

The third chapter, **Grids**, will explain some specific elements related to the actual Grids. This includes how the services portfolio can be kept up to date by monitoring the resources, and trying to predict the availability of them.

The fourth chapter, **Negotiation**, is the main focus of our project. This chapter will explain the whole negotiation process in detail. It will discuss the ontology of the negotiation, and how the agents should devise a strategy for the negotiation. The strategy is deeply bonded to the *utility functions* of the users, which will define how the users evaluate the given circumstances, resulting in a price they are willing to pay, either in form of resources or currency. The chapter will finish with discussing the result of a negotiation, the *contract*, and what happens when the contract is breached, and a conflict occurs.

The fifth chapter, **Security**, will discuss the security aspects of the proposed system. A security analysis will is made and evaluated, and suggestions for remedies to the security issues are made. Also, the concept of *trust* is discussed, and a trust model is devised.

The sixth chapter, **Implementation & test**, will describe what we have chosen to implement, and for what reasons. The implementation will be the basis of testing, and testing is performed to see if the proposed models and methods do actually result in a usable outcome.

Chapter 7 presents the conclusions, as well as presenting an overview of what we think can be done in future work in this field, and with this project.

Appendix A will present conflict scenarios, while B will describe some of the scenarios when rejecting proposals in the negotiation. Appendix C will present an BNF of the proposed negotiation ontology. Appendix D presents a negotiation example, to clarify the elements in the negotiation. Appendix E contains a security analysis of the system, which the discussion in the first part of the fifth chapter is based on. Appendix F will display the class diagrams of the prototype implementation, and appendix G will present some tests of the prototype, and the results of these tests. Finally, appendix H will layout the contents of the included CD.

The bibliography can be found at the very back of the report.

Chapter 2

Model

This chapter will introduce the system modelling, and many definitions and concepts. Firstly, we will discuss our definition of resources in section 2.1. Then, we will draw up three possible overall system models in section 2.2. It is important that we define the concept of agents, in section 2.3, before going on and choosing a system model in section 2.4, which is further explained herein. Then we will elaborate on the economic aspects of the system, and try to characterise the different elements, in section 2.5. Then we will go into more detail about the system model and the various interactions of the players, by describing the various processes that exists, in section 2.6. We will end the chapter by summarising our findings in section 2.7.

2.1 Resources

Our definition of a resource is that it is something that can be valued as a commodity, and therefore has a value. The value is often depending on several external or internal factors, and is determined individually by the various actors. This value is set in *currency*, to make things easier to understand. We will use the specific unit of Danish crowns, *kr.* This is however irrelevant, as it could be any currency or an abstract unit for that matter.

Within Grid systems, we identify the following items as being resources:

CPU This resource is processing cycles in some defined unit.

RAM Memory storage.

DISK Fixed disk storage.

NET Network bandwidth.

APP Application usage, ie. the lease of a licence for the use of a specific application.

Furthermore, these resources are dependent on each other. For instance, you *cannot* order only RAM, as RAM is not the preferred storage media. You have to order DISK instead. Also, all applications use CPU as well as RAM (physical or virtual), and the order of CPU must come with a RAM side dish.

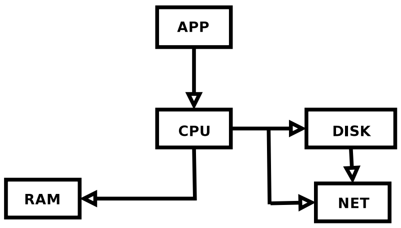

Figure 2.1: Resource dependence graph

These interrelations are graphed in figure 2.1. This figure shows a directed graph, which displays the resource requirement for a particular resource request/order. If a buyer wants CPU, he has to order the other underlying nodes in the graph as well. This relationship should be enforced by the service provider, and maybe somewhat hidden from the user. The provider can offer an arbitrary amount of CPU and the underlying resources in a form of *basis package*. Also, the resources in the graph that have no outgoing arrows, are not possible to order by themselves.

The branching from CPU to NET and DISK means that if you order CPU, you have to order DISK *or* NET. This is because the data that has to be used in the computation can be stored locally, using DISK, but also shared across a network connection, using NET.

The resource relations are normally bound with a factor. For instance, if one wants 100 CPU units, the system might require 10 RAM units for this to happen. However, we have not chosen to incorporate this into our model for simplification reasons, thus making it possible to order 1000 CPU units and 1 RAM unit. We expect these limitations to be enforced locally.

Currency can be considered a resource, but we have chosen not to include it in our regular definition of resources. However, whilst every resource can be valued in currency, so can currency. It has a 1:1 relationship with itself, obviously.

Furthermore, *data* can be regarded as a resource [4]. However, this can be seen as a specialisation of the *application* resource, we have in our model. An application licence can thus be a permit to query the data services provided by the system.

2.1.1 Resource units

The resources should all have defined units, like megabytes for RAM and gigabytes for DISK. Also, this can be a composite unit, because maybe the speed of the storage devices are important, so it can be evaluated as speed times capacity. But for the sake of simplicity, we will avoid such composites as much as possible, unless we deem them important.

We propose the units to be used with our resources, as given in table 2.1.

Table 2.1: Resource units

Resource	**Unit**
RAM	MegaBytes
DISK	GigaBytes
CPU	Benchmark
NET	Mbit/s
APP	[Application,Licences]

The CPU unit is somewhat different from the others, as it is harder to determine. Programs are different in nature, and thus has different operations they need to execute. Processors today have many auxiliary functions that aid the calculations[1], and mathematical microprocessors. Although Intel processors scale linearly according to the MHz [6], other processors do not, and taking into consideration our heterogeneous nature of systems, we must use another unit.

However, the MHz unit is used in major production grids as NorduGrid [20], but they are not concerned by the actual performance details, as they are not aiming to sell any of the resources.

Most of the Internet community agrees on that the only real way of determining CPU speed and capacity, is by benchmarking [24][6], producing a relative number (or several). And even this approach can be deceiving [7][1]. Widely spread and accepted tools exist for benchmarking [8], although costly. In our project, the need for an exact number reference is not so great, and our selection of benchmarking tool is secondary, as long as it gives some kind of hint of the performance.

[1]Like Intel's MMX and AMD's 3DNow

We need benchmarks for all the systems in the Grid, to be able to provide a total of the resources. Tools are in the making to do this more structured [9], but not complete in any way. So we will focus on choosing a single system benchmark tool, and then add the system numbers together. Also, all participating grids must use the same benchmarking application, for the numbers to be of any value.

The RAM and DISK resources have the MB and GB units respectively.

The NET bandwidth will be a number which indicates the speed which the Grid system can provide. This speed will be the speed to its ISP. Here we will assume that the line is not asynchronous, i.e. that upstream and downstream speeds are equivalent. Of course, the actual transfer speeds that is reached depends on many other factors, like congestion on the local network.

The application licences are of course bound to a specific application, so the data type for this are a tuple containing the (unique) application name and the number of licences available (or wanted) for this application.

The currency unit has already been discussed, and set to Danish crowns (DKK).

2.2 Model analysis

This section briefly outlines three different possible system models, from which the final model will be selected. The most important flaws and advantages that the models have is considered, to express the necessity for a good model. Agents will be mentioned, a definition of them will be provided in section 2.3.

2.2.1 Model I

This system model was introduced by [34], and it focuses on the negotiation between Internet stores and potential buyers.

In this model there are a buyer agents and stationary seller agents. The user that wishes to purchase an item, initiates the process by loading the buyer agent with the necessary information about the product he wants to buy. Through a knowledge base server the buyer agent collects information about the whereabouts of potential sellers.

The buyer agent now creates a roaming buyer agent (a mobile agent) that jumps to the first relevant seller system on the list. The roaming buyer agent queries the seller agent for a price of the wanted product and the seller returns an offer and the two agents start to negotiate. After the negotiation is done the roaming buyer agent continues to the next seller on the list and repeats this procedure, while sending status messages to the buyer agent at home. After meeting all relevant seller agents, the roaming buyer agent renegotiates the most promising collected quotes. This procedure is repeated until the sellers are no longer willing to negotiate. The buyer agent returns to the user with the collected quotes and presents them for the user that can choose to accept one of them.

Controlling the negotiation algorithm used in the agents can be an advantage, since all control befalls the user. Another advantage is that the mobile agents can communicate directly on the seller system, thus localising the negotiation.

An disadvantage is that if there are many sellers that it needs to visit, it can be very time consuming, since the buyer agent has to physically move to each seller. Also, the buyer agents need to visit the sellers, thus making them easier to compromise.

2.2.2 Model II

The second model has a hierarchy of dedicated marketplaces (servers), where broker agents exist to handle negotiation. The buyer initiates connection to its local broker, telling it what it is after. The broker, which have a knowledge of local sellers, queries this information and starts contacting sellers. These sellers spawn mobile seller agents to the marketplace, so the broker can negotiate with them. The broker represents the buyer in the negotiation, and it is therefore important that the buyer has total trust in the broker's negotiation skills.

When the negotiation is done, the broker presents the buyer agent with the results to select from. If the buyer is not satisfied with the results, he can ask the broker to inquire in other markets than the local.

The hierarchy system is an advantage, since the buyers can choose to stay on local markets or foreign ones. Also, if the local market puts too much stress on the local marketplace, new nodes can be created in the hierarchy to distribute the workload.

Also, the idea of having a broker ensures that buyer and seller can keep more of their anonymity in the negotiation phase.

2.2.3 Model III

Model III is similar to model II, and what distinguishes the two models, is the role of the broker in the system, as well as the infrastructure. In this model the buyer agent, containing all necessary information, contacts the broker (like in model II). The broker then spawns a new mobile broker agent that has all information about suitable sellers that can be of interest for the buyer. The buyer and broker agent then together visit each seller, to negotiate on the seller systems.

Just as in the second model, the broker represents the buyer in the negotiation. However, this model has no marketplaces, and therefore have to "steal" resources from the seller systems for the execution of negotiation. And it is certainly not scalable.

2.3 Agents

In our system, we will use *agents*. An agent is defined in many ways in the literature, and cannot really be unambiguously defined([10]), and in some cases definitions may be confining. A common definition is ([15]):

> An [autonomous] agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

Different classes of agents can exist. A distinction between 3 types of agents is made in [10], *Neutral Objects*, *Server Agents* and *Autonomous Agents*. The difference lies in what the *goals* and *motivations* are. A neutral object doesn't have either, and is therefore not an agent at all. Although, it can be given goals by users or other agents, and become a server agent, by the process of *goal adoption*. A server agent has goals, but no motivations for having them, as it has been given them. An autonomous agent has motivations that spawn goals, because it constantly evaluates its environment, and chooses on its own.

Most agents in our systems act upon an agenda which is laid out by the initiator or user, and are therefore server agents. Although, we will also see neutral objects and some kind of autonomy. Some will have a relatively short lifespan, accomplishing their missions fairly fast, and some will be more persevering.

We define two main types of agents in our system, namely *Commerce agents* and *Resource prediction agents*.

2.3.1 Commerce agents

The commerce agents are the ones that do the negotiation and interaction with agents from other origins. We distinguish between *three* subtypes of agents in this main group of agents. The *buyer agent*, the *provider agent* and the *broker agent*. These agents are displayed throughout the report with the symbols shown in figure 2.2.

Figure 2.2: Agent symbols

The *buyer agent*, is the agent that facilitates the buyer of services. This agent is sometimes also called a requester agent, as it is launched by the initiator of a transaction. The term *buyer* is used here, as it is easier to understand, although a transaction in this system is more an exchange of resources (including currency) rather than a purchase. This agent is a server agent, as it has goals laid out by a user, and therefore no self-constructed motivations for having them. The main goal of the agent is to purchase the wanted resources for the lowest price, and this goal is not changed throughout its lifetime. Although intermediate goals might change, for instance if the local market cannot offer the resources to the right price, and another market has to be queried.

The *provider agent* acts upon the wishes of the service provider. Another term for this agent are *seller agent*, as it is launched by the seller of services it provides. This agent is somewhat an autonomous agent, as it reacts to an incoming request, and tries to sell the sought services for as high a price as possible. One could although argue that the provider agent always has the goal of maximising the profit of the available resources, and is therefore a server agent.

The reason why we are reluctant to call the provider agent for seller agent, is that a provider of services can act as a buyer also. The symbol for the agent remains the same though. It will be evident in the text, when we refer to the agent as a buyer or a seller.

The *broker agent* is the connecting link between the two former agents. The broker steps in and negotiates using the information about the needs of the buyer. This is a neutral object, as it has no goals, until it receives a request, and gets assigned goal, making it a server object. Although, it cannot be told to do things outside its responsibilities, like spy on the other party.

The *mobility* of the agents are a property that supports the models very well, and a thing that makes agents very useful indeed.

The roles of the different agents are described in section 2.4.1, as the choice of system model is crucial for how the agents must behave.

2.3.2 Resource prediction agents

These agents lie on the grid systems and reports the actual resource consumption and the predicted load in the future. This calculation must take a number of factors into consideration, and can be relatively complex. The actual method of calculation is described in detail in section 3.2.2. The interaction of these agents are with the resource providers, or sellers, only.

To monitor resource consumption, we need methods of checking the current load of many machines, all of the grid participants. This requires some kind of collaboration among the agents to calculate the total resources available at a Grid system. This has to be done with scalability in mind [32].

The stationary resource prediction agents on every machine in the Grid, reports back to the selected front-end of the Grid[2]. The front-end resource prediction agent then gathers these statements, and provide a total for the Grid. How this is carried out, is up to the implementor, and is sometimes provided by the Grid software.

2.4 System model

It may be evident in the text that we prefer model II. Although this model adds complexity in form of the hierarchical marketplaces, this feature is very handy. It can increase security, as things happen in a central zone, for all the local societies. Also, this is the most scalable solution, in our opinion.

The agents are deployed to the marketplace, which is central to the societies. An overview of the model can be seen in figure 2.3. In this figure, we can also see the Certification Authority, which is further explained in section 5.4.

The figure only displays a very small hierarchy, but there could be several levels between the master broker and the local broker. Here, the brokers also refer to the marketplaces, although at a local marketplace, the local broker will spawn several brokers to aid the incoming users.

[2]see section 3.1, about Grid systems.

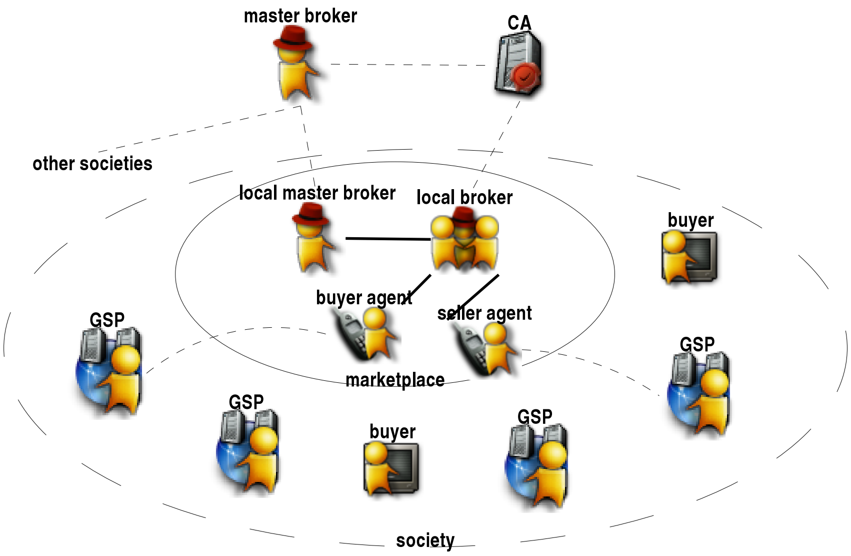

Figure 2.3: Model II

The hierarchy consists of marketplaces, all with local users attached forming a *society*. The maximum number of underlying users or nodes, is set for every marketplace. There can be 3 types of users attached, Grid Service Providers (GSPs), buyers, and other marketplace systems. Often, the GSPs will also act as buyers, but buyers can never be GSPs, as they are defined as participants without resources.

The marketplaces are sort of *agent playgrounds*, so in order for them to play fair, and not compromise the marketplace, the agents are executed within a *sandbox*, not letting them at sensitive system information. For details, see section 5.1.

The root of the hierarchy tree should not have other nodes than marketplaces, as it has coordination assignments, such as relaying requests and agents from one subtree to another. Also, the marketplaces should be laid out according to their geographical distance from each other, i.e. Swedish and Danish users or marketplaces should not be far apart.

When the maximum number of underlying nodes is reached, a dividing process has to start, which must result in underlying marketplaces. This process can be done somewhat automatically, but the hardware part must be present. So an administrator has to set up some new marketplace systems, to facilitate the division.

When these new marketplace systems are created, the old underlying users have to be divided between them. This can be organised in a number of ways, with varying levels of automation. We propose a solution where the marketplace can assign its users to new marketplaces, namely underlying marketplaces. The users will then migrate to the new marketplace.

To justify the existence of the marketplaces, they can charge a fee for every deal they facilitate. This fee can be calculated by the time the broker needs to finalise a deal, or some fixed percentage of the transferred funds. Normally, the former will be preferable, and also our choice for WEGAS.

2.4.1 Commerce agent roles

With a broker in the system, the roles of the buyer, seller and broker have to be clearly defined and enforced.

The buyer has, as the initiator, the final say in acceptance of a deal. He is the one who initiates contact with a broker to express his requirements. These requirements have to be well-defined and complete enough for the broker to handle the negotiation properly.

The buyer agents are mobile in the system, normally residing on the local marketplace, with a the assigned broker. The buyer has a limited negotiation role, as it leaves this to the broker, but it should act upon suggestions and questions from the broker.

The broker has many responsibilities, which would otherwise befall the buyer (or seller). First of all, it has to be loyal to the buyer and seller (or in other words, neutral), and do the best it can to get an optimal deal for the buyer, while respecting the boundaries of the seller. Also, full discretion must be ensured, so that the buyer and seller will be willing to trust the broker, and security in general is an important thing to consider when designing the broker.

The seller is mainly passive, as its usual responsibilities are to advertise the available system resources, which is deemed for sale. Also, when a broker contacts the seller, a mobile agent must be born and sent to the marketplace system. This mobile version has to negotiate with the broker and take care of the interests of the home Grid system.

One of the bases for defining the responsibilities of the agents, are the economic model, as the agents, and the rest of the system, has to support it. This is the subject of the next section.

2.5 Economic model

This section provides our view upon the economic market of our system. We will firstly try to characterise some important aspects of the market, and then try to establish our economic model.

2.5.1 Characteristics of the market

The market supply and demand are varying sizes, but a trend must be identified to be able to make assumptions, which is basis for our further work in this area. This is mainly based on our conception of how the users of the system behave, and who they are.

Characteristic of the players

The users of the system are different in nature, and the need for computational resources as well as storage resources will emerge in different commercial areas. However, we will characterise the different users here from the market of today, to give an overview of the usage potential.

The typical sellers of services will at first be existing Grid systems, which is mainly found at various research institutes, or collaborations of such[3]. Also, some commercial Grid systems can choose to post their resources for sale at times of little utilisation. This can be major medical corporations or biotechnology companies. These types of companies can also appear as buyers of services, and hereby exchanging services with each other.

Later, we will maybe see companies specialising in supplying Grid services for buyers. They will be a form of Grid Service Providers (GSP), like the Internet Service Providers (ISP) we have today.

Buyers of these services can be among the before-mentioned sellers, who themselves have Grid applications they want run. They can buy resources to reduce the time delay for their job to finish, in comparison with only using their own facilities. Also, we will see buyers without infrastructures, who need a few Grid applications run, but don't want to invest a lot of money into hardware.

Finally, the marketplace system has a big role to play, as it tries to ensure that everything flows in an orderly manner. Also, it dispatches brokers to be actively part of the negotiation process.

[3] Like NorduGrid [20]

Job characteristics

Given the resource dependency rules described in section 2.1, we can define the most commonly used distribution of resources that is reserved in a transaction. In table 2.2, some typical jobs are depicted. These examples are products of our imagination and best judgement, and should not be considered as facts.

Table 2.2: Typical job layouts

Job	**CPU**	**RAM**	**DISK**	**NET**	**APP**
1	●●●●●	●●	●●	●	○
2	●●	●●●●	●●●●	●●	○
3	●●	●●●●	●	●●●●●	○
4	○	○	●●●●	●●●●	○
5	●●●	●●●	●●	●●	●●●

1. This job is computational intensive, but doesn't require much data to execute. This type of job is therefore easy on the other resources, and can be quickly dispatched and retracted.

2. This job is more data intensive, and therefore uses minor amounts of processing power, but large amounts of CPU and RAM[4]. However, the CPU usage may be greater, considering how dependent the bus is on the processing power.

3. This job is like nr. 2, but retrieves the data from a network connection, and is therefore very network intensive.

4. This job is only utilising storage space and a network connection to transport the data. Therefore, no other resources are used.

5. This job utilises a special licensed application. The rest of the resource usage are kept at a medium here, but can practically be any of the above.

These jobs are chosen to illustrate the differences and correlations between the resources. For instance, if a lot of storage space is needed, the data must be transported, thus using somewhat large amounts of bandwidth.

[4]As data need to be loaded into memory for processing.

An important factor that aren't included, is *time*. Some jobs can book network bandwidth at the start of the task lifetime, and at the end[5]. So the overall bandwidth usage can seem small, but is maximised at these intervals.

These jobs are merely examples of what, we presume, will be typical tasks that have to be executed. Tasks differ extensively in nature, and could vary from these profiles.

2.5.2 Market model

The market model of our system consists of many buyers and sellers. Although, we assume that it is a buyers market, which means that there are mostly a larger resource supply, than demand. However, this need not always be, as resources are transient commodities.

There are many different market models, like an auction based model where bids are accepted to an item (resource). Also, the classic demand and supply equilibrium theory [28] offers a framework for the market model.

However, we choose to implement various elements from the models. The market is based on individual pricing, as every player decides on his own how he sets the price on his goods. Also, this allows sellers to differentiate their offers to the individual sellers. Although the customisation of the price cannot be particularly individualised, as the information about the buyer, which the seller has access to, is very limited. Almost only the trust level is accessible. So this won't create monopolistic competition [2].

Also, we want to create a tool to aid the seller or buyer in evaluation and pricing of resources. To do this, we will collect information about the previous deals, and from this extract a *market price* for every resource, see section 4.4.2 for details on market prices.

The market price is meant as a helping tool, and not a fixed price. Although, the market price can be somewhat deceiving, as the motives for signing a deal can differ. Like if two parties has a tight bond of trust established by previous experience with each other, they can choose to sign a unusually favourable deal.

The supply is advertised from the sellers in the marketplace system, as described in section 2.4.1. The demand is calculated from the deals made, which will be further explained in section 4.4.2.

The demand and supply numbers resulting in a market price, will hopefully also help to regulate the market, so that congestion is easier avoided, as the incentive for buying services diminishes with a rising market price.

[5]To transport data to the system, and results home.

2.6 Processes

This section will be about the different processes in the system. We have several processes in our system, all of which will be explained. Some processes are segmented into phases to easier describe what happens.

2.6.1 Transaction process

We have the following structure of the different phases in an transaction, as seen in figure 2.4. Also, the agents involved can be seen below the boxes.

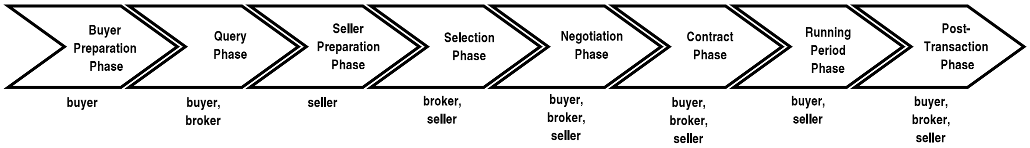

Figure 2.4: The transaction phases

These phases will be explained beneath.

Buyer Preparation Phase (buyer) This phase is the beginning of the process, where the buyer decides that he need some services. This phase includes the calculation of the magnitude of the task to be performed, so the buyer knows what he's after. Also, a deadline or an estimate on time consumption must be made. These things are loaded into a newly created mobile buyer agent, which migrates to the marketplace system, and moves into the next phase.

Query Phase (buyer,broker) At this stage, the mobile buyer agent has migrated to the local marketplace, and contacts the *local master broker* (LMB), which spawns a *local transaction broker* (LTB) to handle the request. LTB queries the local repository for seller advertisements and compiles a preliminary list of relevant sellers. These sellers are then asked to provide an agent for negotiation.

Seller Preparation Phase (seller) The seller must also package its mobile agent with the relevant information, so it loads it with information like the predicted future load of the system, so it can negotiate on the right terms. Also, a minimum price is decided, or the amount of resources wanted in return. Any other constraints the system has, must be known to the agent. It then migrates to the marketplace system.

Selection Phase **(broker,seller)** LTB waits for the sellers to report in, and handles them in a First In, First Served (ie. FIFO) queue. For every seller, LTB initiates negotiation, which will be further described in chapter 4. When all sellers have been handled, the most promising sellers goes on to the next phase, while the others are sent home.

Negotiation Phase **(buyer,broker,seller)** The remaining seller agents must send a message to their home systems, and lock the resources in the time slot they are needed. This ensures that no other seller agents for this system is negotiating for the same resources. More detailed negotiation is then undergone, with more involvement of the buyer in the decisions than the previous phase. This negotiation will also be further described in chapter 4. If the buyer agent is satisfied, the next phase is entered. If he is not satisfied, LTB will ask the buyer agent if LTB should try external markets for a fee, see the details in section 2.6.4.

Contract Phase **(buyer,broker,seller)** At this stage, the buyer has found a seller to exchange services with, and a contract must be signed. LTB compiles the contract, and gets the parties to sign it. This contract is stored in the local repository at the marketplace for possible later policing, and for statistical purposes. The buyer and seller must now handle the communication and exchanging from now on, which is the next phase.

Running Period Phase **(buyer,seller)** Here, the buyer and seller exchange services as contractually agreed. If any money are involved, they are also exchanged. When the exchange is complete, the next phase is entered.

Post-Transaction Phase **(buyer,broker,seller)** Both systems (buyer and seller) must send evaluation messages to the local master broker, evaluating the other party regarding the finished transaction. This step is to accurately recalculate trust vectors for the parties, and to see how successful the transaction was.

Every transaction contains these phases in more or less extent. The negotiation is a big part of the phases, but are explained later in chapter 4.

2.6.2 Registration process

To be able to use the system, a buyer or seller has to be a member of the local community. To be this, he must register with a local master broker at a marketplace (preferably nearby).

The user have to have a valid certificate issued by a certified authority. The marketplaces and brokers should decide on the validity of these authorities, or provide the authorities themselves.

At first, the registrant sends a message to the local master broker (LMB), about wanting to sign up. The LMB then does a couple of checks:

1. Is the registrant really a new user, or can he be traced back to an existing user?

2. Can the registrant be authenticated at a certified authentication authority, to validate his identity?

3. Is the registrant in a nearby system, or should he be transferred to a LMB more local to him?

If these checks do not set off any alarms, and the registrant provides a location at which he can be contacted, the registrant is accepted into the society. He is then given a trust record, set to "unknown" or a specified start value, see section 5.3 for details. He is then allowed to make transactions with other users.

The first check should be made across the whole agent system, on all hierarchy levels, because a banned user from another system could try to register at a different system. This can be done by letting the root marketplace in the hierarchy have a list of banned users, so the check is only made with this entity. The list should not be very large, and the process involves very little communication, so the scalability of the system is not affected.

2.6.3 Seller advertisement process

The sellers need to advertise their services for the local master broker, so that he can match the buyers with the sellers. This must of course be as up to date as possible, and should then be kept simple.

A seller system has to submit advertisements regularly, as the advertisements are set to time out, if no updates are received. This ensures that the broker are less likely to return bad results from searches in the local repository.

The seller should send the advertisements to the local master broker, who will update the repository. This is reached by simple message passing, to keep the communication at a minimum. It is not possible for the local master broker to do a validity check on the resources offered, so the seller can offer services he can't provide. But the trust model should root out these bad seeds.

The advertisements should contain a list of resources, their quantities and their availability dates. These advertisements is, despite what the name might imply, only available for the eyes of the broker, as he is the one doing the search queries.

2.6.4 External markets

Given the hierarchical marketplace system, we have local markets and external markets. The markets are the same in nature, but when the buyer cannot get a satisfying deal, he might try external markets. This is the responsibility of the broker, as he needs to facilitate these queries. The question is, should it be done by messages through the system or by the broker becoming mobile.

To avoid too much message passing, we would recommend that he should travel to the systems himself. Of course, he should not travel too far away, so we suggest he traverses the tree in a breadth first manner, so he minimises the number of nodes between the external and the local marketplace. This should ensure that the real world distance would also be minimised.

Firstly, he should search the underlying marketplaces, and if nothing is found, he should go further up in the hierarchy tree. The broker queries for sellers at the external marketplace he arrives at, and the process continues with selection phase, just as it would at the local marketplace.

2.7 Summary

In this chapter, we have defined what a resource is, which resources we are dealing with, and how they should be considered in the system. We identified that these resources can have relationships, and identified what they were in figure 2.1. Also, they all have translated values, as well as different units, as given in table 2.1.

We examined some different system models, which we considered for our system, before establishing our final system model as illustrated in figure 2.3. This model consists of several *marketplaces* connected in a *hierarchy*.

This model uses *agents*, which communicate, negotiate and interact. We have two main types of agents, commerce agents and resource prediction agents. The commerce agents can be either *brokers*, *sellers* or *buyers*.

We have identified some processes that exist in the system associated with making transactions, registrations and advertisements. A transaction can be split into 8 distinct phases, that the agents must go through to sign a deal.

Chapter 3

Grids

This section will talk about Grids related to WEGAS. We will discuss how a Grid should be defined, what we need a Grid system to incorporate to be able to function with WEGAS, and how WEGAS itself can be characterised.

Prediction of resource availability and how to monitor the resources across a Grid system, will also be discussed, and we will try to outline the possibilities of this.

Lastly, scheduling of tasks for a Grid system will be discussed.

3.1 Grid systems

Grids are an integral part of our project, obviously. A Grid is best defined as dealing with [14]:

> coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.

A *virtual organisation* is defined as a set of individuals and/or institutions. Grids bare relations to parallel computing in clusters, but Grids are much more heterogeneous in nature, and the availability of resources are more unpredictable.

Our proposal is concerned with a system that can connect Grid systems, enabling them to communicate and sell or buy resources from each other. Resources are paid in other resources, or currency. This project differs from the others, as almost all other Grid systems only address sharing of resources, without considering currency transactions being part of the sharing process/policy.

We need a Grid middle-ware system that can provide the various services that we are in need of, which are:

- Distributed resource allocation.
- Information services so we can retrieve information about current resource usage of a submitted job, see section 3.2.
- Security policies.
- A usable interface for interaction, job status information and job submittal.
- A defined front-end for the Grid system.

Apart from the basic requirements of the Grid system, our system is not concerned about the underlying architecture. Of course, the job submittal procedure are different for each Grid system, but ideally this could be interfaced by WEGAS to create a general Grid API, that can be used for distributed computation of tasks[1], or other Grid services. Many commercially supported working groups exist for creating Grid standards[2], but no standards are currently in production.

To keep more variety, we could also let the Grid middle-ware architecture be a factor in the negotiation, so several architectures can be used. This would also help users to a more painless adoption into the WEGAS system.

Whatever systems are chosen for possible implementation candidates, the architecture is important. In recent development, a service-based Grid architecture (OGSA) have been developed [13], and implemented[3]. This architecture incorporates Web Services to make a more seamless distributed systems integration. This idea aligns well with our project, as we are using the term *Grid services* as a collection of commercially available resources. Also, this technology is open and uses a wide range of standards to easier enable usage and integration with applications.

But is our system of hierarchical marketplaces a Grid itself? Foster provides us with a three point checklist that enables us to check at least his definition [12].

According to the paper, a Grid is a system that:

1. *coordinates resources that are not subject to centralised control...*
   This fits our system, because the negotiation performed is controlled locally, by local marketplaces. This is the coordination of resources in our system.

[1] This of course only applies to processing power usage.
[2] joined in the Global Grid Forum (`http://www.gridforum.org`)
[3] Globus Toolkit version 3.0

2. *...using standard, open, general-purpose protocols and interfaces...*
   We strive to use as many open standards in our system, as well as free or open software, where available and appropriate. Our proposed system is not subject to any restrictions[4] and can be used freely by anyone.

3. *...to deliver nontrivial qualities of service.*
   We provide room for many possible constraints and demands in our negotiation and contract structure to allow the user to ensure quality of service.

Given these descriptions and explanations, we can postulate that our system is indeed a Grid itself. A Grid that consists of individual Grids[5].

3.2 Prediction and monitoring

Monitoring the resources in a Grid system is essential in determining how much we can spare of resources at any given moment. Also, to predict future resource availability, collecting empirical information can be helpful.

3.2.1 Monitoring resources

Most of all, the resource monitoring is important for the task scheduler, so it can assign tasks in a distributed manner, taking full advantage of the Grid nature [31]. This is however not the main focus in this section.

To monitor the resources, some kind of monitor must be installed on all the systems in the grid, and in general, the monitoring must be done locally as much as possible to avoid stress on the communication channels. Scalability is also an important matter when deciding on such a system.

Our agent based strategy fits well in this context, as the network load can be substantially reduced if analysis of raw data is undergone locally. Use of mobile agents for active monitoring is described in [33]. This approach is sensible when the resource updates are needed from different sites at different times. Our model is that the Grid status information has to be updated somewhat frequently, so the mobile agent strategy is not suitable. Also because of the resource overhead involved in creating, migrating and running mobile agents.

[4] Other than the obvious technical demands

[5] Actually, any cluster or normal computer can participate, given the required interface, so it need not be Grids. This is not the intention, though.

Another research effort is done in [31], which is supported by the Global Grid Forum. They use the producer/consumer model to monitor performance. This approach is however not particularly scalable in itself, so a *compound producer/consumer* is created. This component acts as both a consumer and producer and can therefore collect the data, see figure 3.1. This way, a hierarchy can be created within the grid, and that can be scalable. These producers and consumers acts as agents for the systems, although stationary.

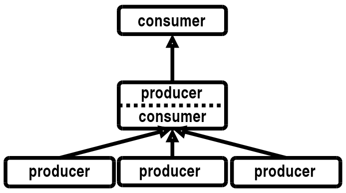

Figure 3.1: Compound producer/consumer

This strategy is however somewhat demanding for these intermediary systems, but this is necessary for not overburdening the consumer that collects all the data. Also, network traffic is reduced, as the intermediary consumer/producer can be closer to its producers.

The load introduced on the systems by the monitoring services are a problem, but normally, monitoring systems are designed with non-intrusiveness in mind [35, 31]. This overhead is a necessary evil for accomplishing our goals.

3.2.2 Predicting resources

Predicting future load is highly usable, as the scheduler can see how it should distribute the tasks given. For the WEGAS system, it is important to have an idea of what can be sold in the future.

When performance prediction is mentioned or treated in the literature, it is always short-term [35]. This makes perfect sense, as longer term prediction can be impossible with too many factors involved. However, this is what we need. We need a way to predict the availability of resources in the future. Our aim is to sell resources at scheduled times in the future, and there is no feasible way to predict long-term performance, based on empirical performance data. Also, in Grid systems, short-term prediction is somewhat unusable[6], due to their heterogeneous nature.

If we consider a Grid containing many nodes within an organisation, including desktop computers, we cannot ensure that all machines are turned on at all times, especially the desktop computers. Typically, such machines are turned on and off at regular intervals (when employees are working and not), so this pattern will surely be identified by time series analysis. This can be done for each node or for the Grid system in its entirety.

The previous generates various problems when we examine the demands of the specific resources. For instance, if an amount of DISK is needed and assigned across a selection of available nodes, and one of those nodes breaks down, we have no foolproof way of getting the data that was on this node. Of course, some method of caching or replication of data can recreate or supply the temporarily lost data. This, however, is a demanding task, and quite inefficient, if the amount of DISK needed are several TeraBytes.

Time series analysis prediction is useful for a long-term mean of the resource availability, but we can never guarantee that resources are available at the time they are needed. Therefore, we have to define some *minimum availability* of resources at certain times, as well as defining a buffer region with resources that within all probability is available, but not guaranteed.

We identify three approaches to this problem.

1. The user can define some kind of internal risk factor, defining how willing he are to sacrifice trust level for selling the predicted available resources, that cannot be guaranteed.

2. The buffer region is used as a factor in the negotiation, and is defined to be more risky for the buyer, and therefore sold at a lesser price than resources contained within the minimum available limit.

3. Only the minimum available resources are sold, and the rest of the resources are used to finalise tasks quicker, thus releasing resources earlier, so they can be resold.

These approaches vary in complexity, and all have pros and cons. To keep WEGAS simple, we choose the third approach. Although the two other approaches could also be used, with small changes. What the third approach lack in optimality, it makes up for in quality assurance. This approach will be the safest for ensuring Quality of Service (QoS), and it spawns the least changes in the rest of the system.

[6] For REAL Grid systems, and not cluster management systems [12]

Our proposed model assigns Grid resources to various buyers. The usage of these resources are scheduled to start and stop at certain defined times, as laid out in a contract. This schedule is the most helpful tool in determining resource availability of the Grid system. Provided that the resources are distributed fairly, this *task schedule* is the main component in our prediction system. The task schedule is like a calendar, it contains information of when a particular resource is booked, and the quantity of it. It will be further elaborated upon in section 3.3.

With these elements, we can provide basic prediction services to the user.

3.3 Scheduling

When a task has to be scheduled to start at a certain point in time, the Grid owner has to consult the schedule to see where it will fit in. But how do we fit the task in the schedule?

Every resource has its own schedule, containing bookings for that resource at different times. In the graph in figure 3.2, an example of a schedule for a DISK resource is given. The booking boxes are denoted with A, B, C and D. The scale on the x-axis is hours, and 0 refers to now.

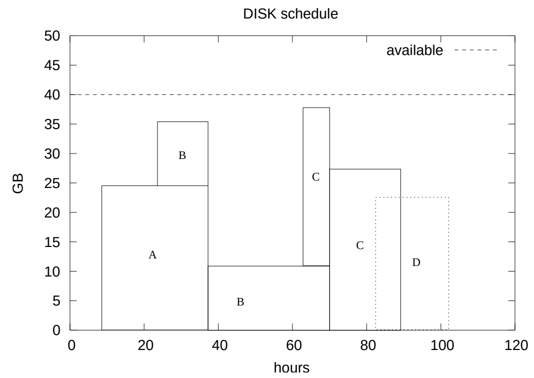

Figure 3.2: Example schedule for DISK

A, B and C are tasks that are already booked, whereas D is a task that the buyer wants to put in a slot indicated by a dotted box. Clearly, that cannot be done, so we need an algorithm to be able to fit it in the schedule. We have several possibilities. We can choose to fit it in as early as possible, in the example, that would be at the gap at about $37 - 63$ hours. Or, we could try to fit it in at the time slot closest to the slot the buyer wishes. That would be right after the C task has finished, i.e. at about 88.

To our mind, the algorithm that fits the needs of the seller the most, would be the first one, as it tries to maximise his resource usage. The seller can use this algorithm when he has to propose a deal. Also, the buyer can reject the earlier time slot, forcing the task to be put at the later time slot, with this algorithm.

However, the seller can also try to appease the buyer, as this will often give him a greater price, because the closer the buyer can get the date, the more satisfied he is, and the more he will pay for the services.

The first-fit scheduler algorithm can be described in pseudo-code like follows:

FIRST-FIT(*task*)

```
s ⇐ the schedule
a ⇐ the amount of available resource
hours ⇐ hours in future to check
for i = 1 to hours do
   space ⇐ true
   for j = i to i + task.length do
      if (a − s[j].booked) < task.size then
         space ⇐ false
      end if
   end for
   if space = true then
      return i
   end if
end for
return NIL
```

And the best-fit algorithm somewhat similar in pseudo-code, $preferred$ is the buyers preferred start time:

BEST-FIT(*task*,*preferred*)

```
s ⇐ the schedule
a ⇐ the amount of available resource
hours ⇐ hours to check
k ⇐ preferred
for i = preferred to preferred + (hours ∗ 2) do
   k ⇐ (i mod 2 == 0?k − i : k + i)
   space ⇐ true
```

```
    for j = k to k + task.length do
        if (a − s[j].booked) < task.size then
            space ⇐ false
        end if
    end for
    if space = true then
        return k
    end if
end for
return NIL
```

The best-fit algorithm tries the periods that surround the preferred start date, trying an hour later, an hour earlier, two hours later, two hours earlier, etc.

The algorithms assume that the available resource, a, is constant in the *hours* period. However, the algorithm can easily be altered if the schedule were to contain the amount of available resource at a given time. The algorithm returns an hour the task should have as start date, or NIL.

3.4 Summary

The Grid systems has to have certain elements to function properly in WEGAS, namely such things as information services, distributed resource allocation, security policies, usable interfaces and a front-end.

The resources the Grid provides, must be monitored to know what the capabilities are, as well as for billing. Prediction of resources is just as important, and we presented 3 different approaches to determine what resources can be sold. We settled on using the simplest solution where only the minimum available are for sale.

The scheduling of tasks are made with a first-fit or best-fit approach, maximising the sellers utilisation of his resource schedule, and maximising the buyers wishes respectively. This has been compiled into two algorithms depicted in section 3.3.

Finally, we establish that our proposed system is indeed a Grid itself, as it has the characteristics of one.

Chapter 4

Negotiation

Negotiation is an very important part of the system, since it is the process where the actual deals are made and our main focus. This process consists of agents that affect the outcome of the deal by using the information that they possess, in a effective manner. It can be defined as in [17],

> Negotiation is the process by which a group of agents communicate with one another to try to reach agreement on some matter of common interest.

In section 4.1, we will talk about different negotiation models, and try to adapt them to our needs. We will then explain our negotiation ontology in section 4.2, and explain how the strategy should be determined as well as how we represent the data structures, in section 4.3. We will explain the pricing aspects of the negotiation, mainly how the users should value the services given some parameters, but also market prices, demand and supply are treated, in section 4.4. After all negotiation is completed, a contract is created, and the contents of that is explained in section 4.5. Whatever conflicts that can occur, are treated in section 4.6, before a summary of the chapter is given in section 4.7.

4.1 Models

There are many different areas within negotiation and therefore many different types of negotiation models. To find a suitable negotiation model for the system, we need to identify the two basic components of our negotiation system, the negotiation protocol and the negotiation strategy. The protocol describes how the agents interact with each other, while the strategy describes the actions taken during the negotiation. This section will first investigate different negotiation models and, based on this investigation, find a suitable model for the system.

4.1.1 Bazaar model

Automated negotiation is very useful in electronic commerce. In [21], a model called the *original Bazaar model* is suggested. In this model, the buyer uses a learning based strategy to decide on good counteroffers by learning the seller's reservation price[1]. The reservation price is learnt by using a proposed mathematical algorithm iteratively. Also, the model considers the risk willingness of the buyer. The more risk a buyer is willing to take, the lower the price he will offer.

The fact that the model can handle risk, which could be used to represent the trust among agents, is good. However, using this model would simplify the negotiation too much, since it doesn't take all negotiation parameters into consideration. A modified version, implementing the variables we need, would also be very communication intensive, because of the iterative nature of the algorithm, and the fact that the steps taken are small.

4.1.2 Game theory

Many negotiation models are constructed using the concepts of game theory. In [19], these concepts are used when constructing a one-on-one negotiation model. The paper proves that using this model, the negotiation is finished in one step, which has to be considered very efficient. The two most important variables in this negotiation are the disagreement payoff, how much the user loses on not closing the deal, and the power of the player. The solution to this one-step negotiation is a point between the initial offers that is *Pareto optimal*. Pareto optimal is when none of the players can have a better solution without making the other player worse off. Striving for Pareto optimality is very common with negotiation models based on game theory, and a fair principle.

A seller or buyer that is highly trusted in our system, is similar to a player that has great power in the negotiation strategy in this paper. In addition, the utility functions of the agents and a player's disagreement payoff is, with some modification, comparable. However, the agents are individuals and are not interested in the welfare of the other agents (in the negotiation at least), whilst Pareto optimality is more fair. To be able to finish in one step,

[1]The reservation price is the minimum acceptable offer for the seller.

also means that the broker needs all information from the parties, and that prevents any hiding of information.

The paper proves that this model always has a unique solution, which is not necessarily true in our case, where a negotiation does not necessarily end with success. If the agents don't believe that they can come to an agreement, they can withdraw from the negotiation. In conclusion, this model cannot be used due to conflicting goals, and lack of information hiding.

4.1.3 Argumentative negotiation

Considering the previous two models, the agents need a more complex negotiation model for making use of the information that agents have access to. Accordingly, the agents need to send arguments to the other party to convince them of some issue. Such an argumentative negotiation scheme was suggested in [23] and in [27]. Using this protocol, the agent receiving an offer can answer with a suggestion on an alternative solution. To exemplify, a negotiation between a customer feeling peckish for coffee and a waitress at a Starbucks™could look like this:

Customer: I want a small decaffeinated Colombian cafe latte.

Waitress: I don't have that, but you can get a small decaffeinated Kenyan cafe latte.

Customer: Sounds good, I'll take that.

This kind of argumentative reasoning speeds up the negotiation process, and makes it possible for the parties to explore alternative solutions, suggested by the other party.

4.1.4 Model adaptation

With the last negotiation model and protocol suggested by [23], all information elements that exist in our system can be used as arguments in the negotiation. Also, the agents have the possibility to justify their negotiation stance. Therefore, we will adapt this negotiation protocol.

However, the suggested protocol is constructed for general use and cannot be directly applied to our system. The arguments need to be formalised for the agents to understand each other, and apply the appropriate countermeasures. In order to do this we need to define an ontology for our system.

4.2 Ontology

In this section, we define the negotiation ontology for our system. As mentioned, the protocol is adapted from [23] and our ontology is therefore partly based on this, with adjustments for the system in hand.

4.2.1 A formal language

It is assumed that an agent can, in some way, be identified. All agents identifiers that participate in an negotiation are collected in a set called *Agent*. All agents in the system are socially equal, no agent has greater authority than any other during the negotiation[2].

The agents need a way to talk to each other. For this, we will now create a language, L, in which different propositions can be expressed. As the user can choose to pay with resources and/or currency, which we want to collect into one price data type, L need to be a higher-order logic.

A proposal, or offer, contains all elements used in negotiation. Elements can be e.g. resources or constraints, and they can have different values. Constraints that are nonnegotiable, *hard constraints*, are not expressed with the use of L, since participants with mismatching hard constraints have already been sorted out, by the time they reach negotiation.

A negotiation normally concerns several resources. To represent a resource in L, a function for each resource is provided, for instance, RAM is indicated by the function $RAM(variables)$. A resource function takes one or several variables from the first 4 rows of table 4.1 as arguments. Instead of having one common start date and time period for all resource needs, one can express individual start dates and time periods for each resource, with the *startdate* and *length* variables.

Table 4.1: Variables in L

Variable	Explanation	Data type
size	Amount of resource needed.	Integer
price	The price of a resource, the total price, or an expression containing resources.	Integer or Logical Expression

Continued on next page

[2]Which means that the agents can't force each other to do something.

Table 4.1 – concluded from previous page

Variable	Explanation	Data type
startdate	The start date for the booking of the resource.	Date
length	The time period the resource is booked.	Date
trustlevel	The minimum level of trust for a user.	Integer
advance	Express the payment advance wanted.	Percentage

With the use of the variables in table 4.1, it is not possible to express complete negotiation offers or the value of the variables. To do this, we introduce the functions *equal*, for expressing equality, *and* for conjunction, and *or* for disjunction.

Here are some examples that can be expressed in L, so far:

10MB RAM costs 10 kr. and 2GB DISK costs 15 kr. $\Rightarrow$

$$\begin{aligned} and(\quad & RAM(equal(price, 10), equal(size, 10)) \\ & DISK(equal(price, 15), equal(size, 2)) \quad) \end{aligned}$$

Moreover, since the data type for the price variable can either be an integer or L expression, one can state the price as a compound value:

20MB RAM costs 10GB DISK and 15 kr. $\Rightarrow$

$$RAM(equal(price, and(DISK(equal(size, 10)), 15)), equal(size, 20))$$

The resource dependency graph given in section 2.1, is not implemented directly in L. Such restrictions would create more confusion than aid in understanding, in our opinion. The resource dependency graph has to be enforced in an implementation however.

4.2.2 Communication language

L can be used to construct complex negotiation proposals and requests, which the agent can interpret. However, so far, it is not possible to express the context of an expression in L. A communication language, CL, is suggested in [23] to represent the different actions that can be taken during a negotiation. CL is represented by the set I_{nego} as denoted here:

$$I_{nego} = \{proposal, request, accept, reject, withdraw\}.$$

The syntax for CL is described in table 4.2.

Table 4.2: CL command overview

Function	**Comments**
$request(a, b, \delta, t)$ $(a, b) \in Agents, t \in Time, \delta \in L$	δ represents the requested services.
$reject(a, b, \delta, \mu, t)$ $(a, b) \in Agents, t \in Time, \delta \in L$ $\mu \in R$	δ represents the offer that is rejected and μ represent the reasons for rejecting it.
$proposal(a, b, \delta, t)$ $(a, b) \in Agents, t \in Time, \delta \in L$	δ represents an offer.
$accept(a, b, \delta, t)$ $(a, b) \in Agents, t \in Time, \delta \in L$	δ describes the offer that is accepted.
$withdraw(a, b, t)$ $(a, b) \in Agents, t \in Time$	Indicating that agent a wants to withdraw from the negotiation.

The $reject$ function in CL contains a variable, μ, which represent the reasons for rejecting a proposal. It is important for the agents to be able to express reasons, since it helps the other agents to understand their position and come with a more suitable proposal. We have chosen to construct a language R for expressing reasons. A rejection always comes after receiving a proposal, and therefore it is important that R can reflect all relevant elements in a proposal. There are three main categories of reject reasons an agent can give:

- Resource reasons
- Constraint reasons
- Payment reasons

There are an infinite amount of reasons for rejecting a proposal, while only a minority of them are relevant for this system. In appendix B, all relevant reasons, and how to express them in R, are listed. The different reasons will have different effects in the formulation of the next proposal. However, these effects will be discussed together with negotiation strategy, see section 4.3.

Using the above concepts we can define DF, as done in Definition 1 below. A full BNF of the *digital framework*, DF, can be found in appendix C.

Definition 1.
The digital framework is a tuple $DF =< Agents, L, CL, R, Time >$

1. $Agents$ is a set of agent identifiers.

2. L is a higher-order logic language, constructed as previously explained.

3. R is a language for expressing rejections regarding resources, constraints and price.

4. CL is a communication language defining five message types. Given that a,$b \in Agents$, $\delta \in L, \mu \in R$ and $t \in Time$ it is defined as:

   (a) $request(a, b, \delta, t)$
   (b) $proposal(a, b, \delta, t)$
   (c) $accept(a, b, \delta, t)$
   (d) $reject(a, b, \delta, \mu, t)$
   (e) $withdraw(a, b, t)$

5. $Time$ is a discrete totally ordered set of instants

4.2.3 The protocol

Using DF, we can express offers and rejections and the agents can now implement the following protocol, illustrated in figure 4.1, with the states explained below:

1. The negotiation starts with the broker, a, sending a request to the seller, b.

2. If the seller immediately sees that he cannot provide the services requested, he withdraws ($\Rightarrow$ 7), if he can provide them, he sends a proposal to a ($\Rightarrow$ 3). If b has received a rejection from a earlier in the negotiation, the new proposal should be composed using this information.

3. a evaluates the proposal and if considered beneficial, a accepts it ($\Rightarrow$ 6). However, if a believes he has a better suggestion, he sends a new

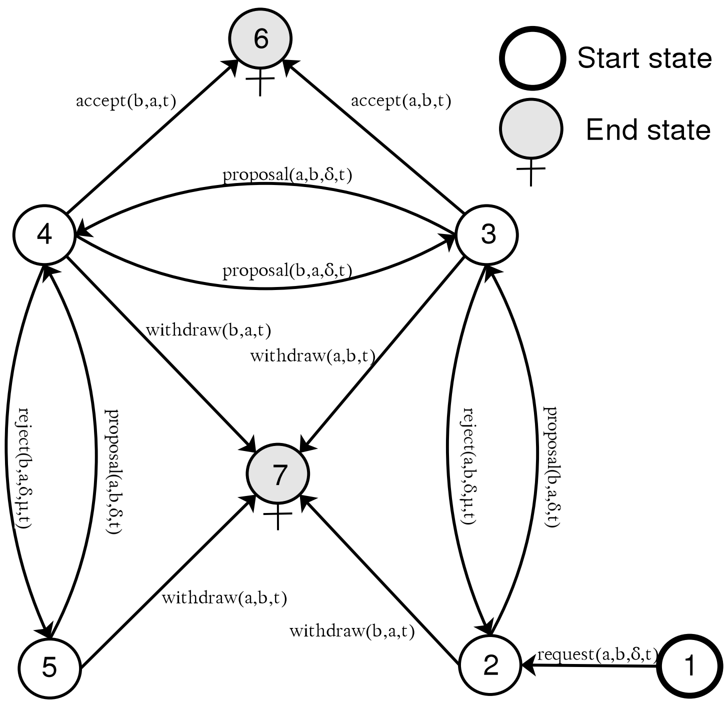

Figure 4.1: Negotiation protocol

proposal to b ($\Rightarrow$ 4), or if a for some reason cannot accept the proposal sent from b, a sends a rejection containing the reason for disliking the proposal ($\Rightarrow$ 2).

4. Same procedure as in 3, but it is now b that should make a decision. He can either accept the deal ($\Rightarrow$ 6), reject it with a reason ($\Rightarrow$ 5) or send a new proposal to a ($\Rightarrow$ 3).

5. a has received a rejection of a proposal. After interpreting the reason for rejection, a withdraws ($\Rightarrow$ 7) or sends a new and better proposal ($\Rightarrow$ 4).

6. The proposal is accepted, the negotiation ended successfully.

7. One of the agents has withdrawn, the negotiation ended unsuccessfully.

With the described protocol, we believe that the agents can communicate and describe their wishes in a flexible and efficient manner. We have created a protocol with the possibility to reject proposals and express reasons for rejecting them. Having the possibility to explain their negotiation stances increases the likelihood of the two parties finding a common negotiation ground.

4.3 Strategy

The previous section described how the agents can formulate negotiation sentences, so that they can communicate in a satisfying manner. However, how the agents decide what the contents of the sentences should be, is not yet discussed. We call this the *negotiation strategy*.

To start with, we should define what the goal of the negotiation is. The main goal is of course to close a good deal with the other party, but when should the agent stop negotiating and accept a given proposal?

We have identified two main types of strategies in a one-on-one negotiation, the first-fit approach and the best-fit approach. In the first-fit approach the agent accepts any proposal if it evaluates to be equal or better than his reservation price[3], where in the best-fit approach, the agent explores all possible ways of combining parameters to get various proposals, and then selects the best one.

We have adopted a mixture of the two strategies, since the outcome of the first-fit approach can be rather random and using the best-fit approach could result in a great number of possibilities to explore and could exceed a reasonable time span. The model is divided into two parts. In the first part, the parties negotiate on the elements in the deal such as start date and advance payment. This part of the negotiation is continued for a predetermined time[4] and all acceptable proposals are saved. The second part is the refinement part. The broker has now selected one of the offers found in the last stage and starts to put pressure on the price by, for example, comparing the offer with offers from other sellers.

4.3.1 Preferences

Our strategy is based on the preferences that the user gives the agent. The preferences describe what the user's priorities are in the negotiation. These preferences can also be viewed upon as constraints, as the agents must follow these preferences. The preferences include the following elements:

[3] The minimum acceptable price

[4] This should be determined experimentally, or explicitly by the user

Advance A tuple, <*preferred, limit*>, containing the preferred payment advance in percentage of the price, and the minimum or maximum limit, for sellers and buyers respectively. The limit can be dynamically calculated from the trust level of the other party, to make the deals more differentiated.

Payment The prioritised list of different payment methods, where each element in the list is represented by a tuple, containing four elements. The first element can either be `fixed` or `variable`, indicating variable or fixed pricing. The next element can be `individual`, for individual pricing for each resource, or `total`, for total pricing. The third element can be described in L, containing the wanted resources, to indicate that resources are to be used as payment, or be set to `currency`, to indicate that currency is wanted. The fourth value contain a factor of how this method will affect the price, i.e. 1.2 if the price should be raised 20% as a consequence of the appropriate method. It is used in the utility function, see section 4.4.5.

Dates Contains a time period, in which the resources can be reserved. These dates are hard limits, and indicate the time domain the negotiation can involve. If the dates in the request are nonnegotiable, this element is empty. The seller will seldom use these dates, as his limits are from now to eternity (in theory). The availability are more dynamically determined using the schedule described in section 3.3.

Reservation Price Contains the total reservation price. This is the minimum for the seller, and the maximum for the buyer. The buyer sets this value explicitly when initiating a transaction negotiation, while the seller calculates it on the basis of certain factors, see more in section 4.4.4.

Minimum trust levels This is a tuple of minimum trust levels, <*hard trust,soft trust*>. The first element is the hard constraint, which cannot be violated. The last value indicates at what value the price is affected by trust. The user will be willing to make a deal with the other user if he has a trustlevel lower than the soft constraint, but it will be at inferior conditions.

These elements contain much of the information used to negotiate properly. For example, a list of this information for a buyer could look like this:

Advance:	`<preferred,limit> = <15%,50%>`
Payment:	`[(fixed,individual,null,0.9), (variable,null,null,1.1), (fixed,total,currency,3.0)]`
Dates:	`2004.2.10 12:00 - 2004.3.30 12:00`
Reservation price:	2000
Min trust levels:	`<hard,soft> = <15,25>`

This example indicates that the buyer would only like to pay 15% of the total price in advance, but can accept as high an advance as 50%. Also, we can see that individual pricing is best for him, but he can be paid with some resources also. Everything must take place within March 10^{th} and 30^{th}, he wants to pay at maximum 2000 kr. in total for the deal, and he wont negotiate with sellers with trust levels under 15.

4.3.2 CL and trees

CL describes how we should represent the various message types, given in I_{nego}. However, the parsing of CL can be made more easy and versatile if we represent it as a *tree*. Also, the preferred standard form of communication when creating new systems, is *XML*. The XML *Document Object Model* (DOM[5]) can be represented in tree form. So we will use this data structure to represent CL.

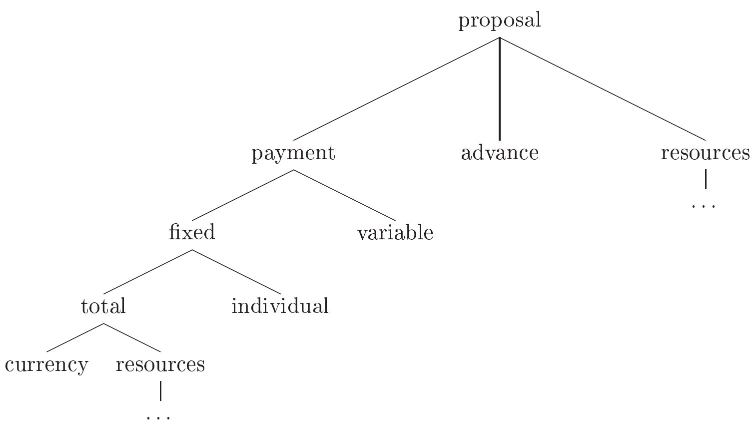

Figure 4.2: A proposal tree

We can also join the various types of messages in the same kind of tree, as

[5]`http://www.w3.org/DOM/`

most message types incorporate the same L structure. The root node of the tree describes what message type it is. For example, we can see a proposal in figure 4.2. The data values are not included though, but all payment type possibilities are shown.

The *resources* node has the various resources in question as children. The two *resources* nodes are similar in their content, but very different in their meaning. The node with the root node as parent contains the resources the buyer wants, while the node with the *total* node as parent, in the payment branch, is resources used for payment, and thus wanted by the seller. An example of a *resources* node is shown in figure 4.3.

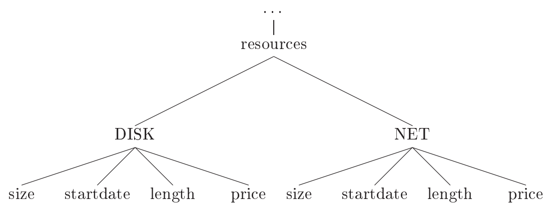

Figure 4.3: The *resources* node

The *price* nodes on the resources are only present when using variable or fixed individual pricing, and the nodes are *not* present in the *payment* branch.

A request is made in the same manner, but without any *payment* or *advance* branch, as the seller must suggest the contents of these elements.

We propose that the rejection and proposal type are joined in the same tree structure. This will further minimise the communication quantity, as parts of the proposal and the rejection can be sent in one tree, one message. To demonstrate, imagine that the buyer thinks the advance payment the seller suggested is too high, as well as the payment method is not desired. The root element will be named *rejection*, as the tree contains rejection nodes. The rejection details are added as new nodes below the relevant nodes, like depicted in figure 4.4. This is done when parsing the tree, which will be explained in section 4.3.4.

The nodes can have attributes as well as values. Every element in the *payment* sub-tree have an attribute called *flag*. This flag must be used to help the parties remember what the past proposals and rejections has determined.

When an element is rejected entirely, the *flag* attribute of that element is set to "REJECTED" for the remainder of the negotiation. When a proposal is accepted by one of the parties, and he thinks he can make a better proposal,

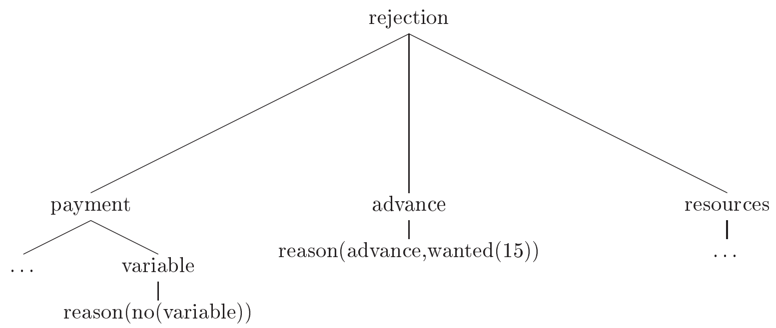

Figure 4.4: A rejection tree

he must save this accepted proposal, and, in the newly created proposal, set the flag of the previously accepted payment method node to "PROPOSED". This ensures that both parties know that this element has been explored.

Also, the flag value "CURRENT" are used to indicate what the current payment node is. The value "FIXED" is used for elements that are non-negotiable.

As can be seen on the trees, the start dates and length elements are bound to the individual resources. Apart from the resource dependencies mentioned in section 2.1, there are dependencies regarding dates. For instance, the RAM and CPU resources *must* be booked at the same time slots, but the model allows booking them at separate time slots. For this, we suggest that a resource coupling or *packaging* is used, enveloping the resources in a package, so they have the same start date and length, while sizes and prices can be separate. We suggest this as a solution to the problem, but we wont incorporate it into our model due to time limitations.

4.3.3 When to reject and when to propose

An agent receiving a proposal does not necessarily need to respond with a rejection. Instead he can choose to send a counteroffer in form of a new proposal.

An agent is obligated to send a rejection containing disapproved elements in the preceding proposal. At a point in the negotiation where all elements in the proposal are acceptable, the agent starts to consider alternative solutions. To find these solutions, the agent searches the tree and tries to create a new proposal. The possibility of sending counterproposals gives both parties an opportunity to explore different negotiation stances.

4.3.4 Parsing the trees

When receiving a message from the other party in the negotiation, the agent needs to parse the included tree in order to interpret its meaning, and to send an appropriate reply.

We have created an algorithm for parsing the trees and creating suitable answers. The algorithm will be explained in two different parts, handling rejections, and handling proposals and requests.

Handling incoming rejections

When a rejection is received, a new proposal p is created with updated elements. The elements of a rejection not containing rejections, is copied to the resulting proposal. This algorithm is the part handling rejections:

1. If the payment branch does not contain a rejection, the information from the payment branch is copied to p. If it does, the payment branch is searched to see if the payment methods from the preferences has been proposed or rejected earlier. It tries the first element in the preferences that hasn't been proposed or rejected earlier. Should there be no acceptable methods, the algorithm will exit with an error.

2. If the advance branch contains a rejection, the payment advance value will be adjusted, and stored in p. If not, the advance branch is copied to p.

3. The length, size and type of resource is non-negotiable, so only the start date can be discussed in the resource branch at this stage. If a rejection is detected on the start date node, the date can be set to something more preferable.

4. Finally, the price is calculated. The utility function is used to determine the price, described further in section 4.4.5. The price is attached to p where appropriate according to the payment method.

Handling incoming requests and proposals

The nature of a request and a proposal is similar and can therefore be treated, more or less, the same way.

If a proposal is received, the algorithm CREATE-REJECTION is used to check if there are any elements that are unacceptable. If no rejection is returned, the proposal is treated as acceptable, which means that a new proposal p should be created. The advance payment will be calculated, if not accepted. The rest of the algorithm is the same as step 3 and 4 above.

We provide pseudo-code of the algorithm for handling all types of trees below. It takes the user's preferences and the received tree as argument.

PARSE-CL(*preferences*, *cl*)

```
p ⇐ new proposal
pref ⇐ preferences
if cl =REJECTION then
  if HAS-REJECTION(cl.payment) then
    p.payment ⇐ NEXT-PAYMENT(cl.payment, pref)
    if p.payment = NIL then
      WITHDRAW()
    end if
  else
    p.payment ⇐ cl.payment
  end if
  if HAS-REJECTION(cl.advance) then
    p.advance ⇐ ADJUST-ADVANCE(cl.advance)
  else
    p.advance ⇐ cl.advance
  end if
  if HAS-REJECTION(cl.resources) then
    p.resources ⇐ FIT-DATE(cl.resources, pref)
    if p.resources = NIL then
      WITHDRAW()
    end if
  else
    p.resourcess ⇐ cl.resources
  end if
else if cl =REQUEST or cl =PROPOSAL then
  if cl =PROPOSAL then
    p ⇐ cl
    r ⇐ CREATE-REJECTION(pref, p)
    if r != p then
      return r
    end if
  end if
  p.payment ⇐ NEXT-PAYMENT(cl.payment, pref
  p.advance ⇐ pref.advance.preferable
```

```
    p.resources ⇐ FIT-DATE(cl.resources, perf)
    if p.payment = NIL or p.resources = NIL then
        WITHDRAW()
    end if
end if
The utility function is run to determine the price and update p
return p
```

The pseudo-code uses some help functions. The first one is the boolean HAS-REJECTION function, which takes a branch as input and searches it for rejections, returning *true* if a rejection is found.

Second function is NEXT-PAYMENT which uses the payment branch from the tree and the preferences of the user to decide the most preferable payment method. It returns a payment branch which is altered according to these preferences. Information which is not altered, is copied. If a appropriate payment method cannot be found, NIL is returned.

The ADJUST-ADVANCE function is a small function that regulates the payment advance towards the user's preferred value, to try and accommodate the other user's wishes.

The FIT-DATE function takes the *resources* branch of the input tree and the preferences as input and return the next possible date the task can be performed. For sellers, the SCHEDULER algorithm, explained in section 3.3, is used. All other information that exists in the input branch will be copied to the new resources branch. However, if a new date cannot be found, NIL is returned.

The last help function is CREATE-REJECTION, which searches the tree for elements that are not acceptable. If none is found, it returns the input tree, if it finds some, a rejection tree is returned. We will describe the generation of a rejection in pseudo-code below.

CREATE-REJECTION$(preferences, cl)$

```
pref ⇐ preferences
r ⇐new rejection
copy all elements of cl to r
branch ⇐ cl.payment
for all node in branch do
    if node.flag != REJECTED and node.flag != PROPOSED then
        if MATCH(pref.payment, node) = false then
            attach rejection for node in r
            exit for
        end if
```

```
      branch ⇐ node
    end if
  end for
  new_advance = ADJUST-ADVANCE(cl.advance)
  if new_advance better than cl.advance then
    attach advance rejection to r with new_advance
  end if
  if FIT-DATE(cl.resources, preferences) = NIL then
    attach date rejection to r
  end if
  return r
```

The MATCH function matches the current node with the preference payment list. If the list contains an element with the current node name, the function returns true, and false otherwise.

4.3.5 Deal refinement

After both parties find all the elements in the deal acceptable, they move on to the next step in the negotiation, where the deal is refined by regulating the price.

At this stage, the broker has negotiated with several agents and can compare the achieved deals with other deals to try to push the seller to lower his price to the same level (or better). Also, the broker can argue that if the buyer is a regular customer, he should have a discount. The reasons that can be used in this step of the negotiation is called *betterOffer*, *regular* and *tooHigh*. It goes without saying that *betterOffer* cannot be used if a better offer does not really exist. This is one of the reasons why a trusted broker is handy, as the buyer could fabricate the other offers easily, where a broker must be truthful and loyal to both parties.

The *regular* reason indicates that the buyer want a rebate for being a regular customer. The seller can choose to grant this, or not. The *tooHigh* reason is used when the buyer is unsatisfied with the price. The seller can react on this as he pleases.

The broker's argument is sent to the seller with a rejection containing the matter of interest. The seller evaluates the rejection to see if he thinks the reason is valid for lowering the price. The seller answers the argument by either sending a new proposal with a lower price, or by sending the same proposal again, indicating that the reason is not satisfactory for lowering the price. This procedure continues until all reasons are void.

4.3.6 Recap

Here, we will try to recap the negotiation phase in more compact form, with the aid of a figure.

The communication within the processes is shown as a flowchart diagram in figure 4.5. At the beginning of the flowchart, the agents have their utility function, preferences and a minimum/maximum price.

The seller receives a request and constructs a proposal. The broker can reject the proposal and the seller will then construct new ones, until the broker finds all elements acceptable on behalf of the buyer. The price and all other elements in the proposal has to be stated in the preferences for the proposal to be acceptable.

If the broker can construct a new proposal based on his preferences, one is created and sent to the seller. The same procedure is then started again, with the seller sending rejections and the broker constructing new proposals, until the seller is satisfied.

Alternatively, the parties can access the next step in the negotiation, by the negotiation timing out. This is illustrated by a dotted arrow in figure 4.5. Also, the agents can at any time withdraw from the negotiation. If no proposals has been accepted, the negotiation is treated as a failure[6].

The second part of the negotiation is entered when all the details, apart from the price, are decided. However, a prerequisite for entering this phase, is that at least one proposal must have been accepted. The broker then selects one of the proposals, using the utility function, and as long has he has arguments for lowering the price, he presents them for the seller. The argument is evaluated by the seller and if it is accepted, the seller sends a new price to the buyer. Situations where an argument is acceptable but the new price is lower than the sellers minimum price, the minimum price is sent in return. During the second part of negotiation the broker can fetch proposals from other sources and use this as reasons for lowering the price.

When the broker has run out of arguments, a contract is created and delivered, by the agents, to the service provider and the buyer. Both parties then have to manually sign the contract, using their certificates. Only if both parties return to the broker with a signed contract, the negotiation is a success.

How the agents rate the various deals, are treated in the next section about pricing.

[6]Due to illustration simplicity the withdraw state is not shown in figure 4.5

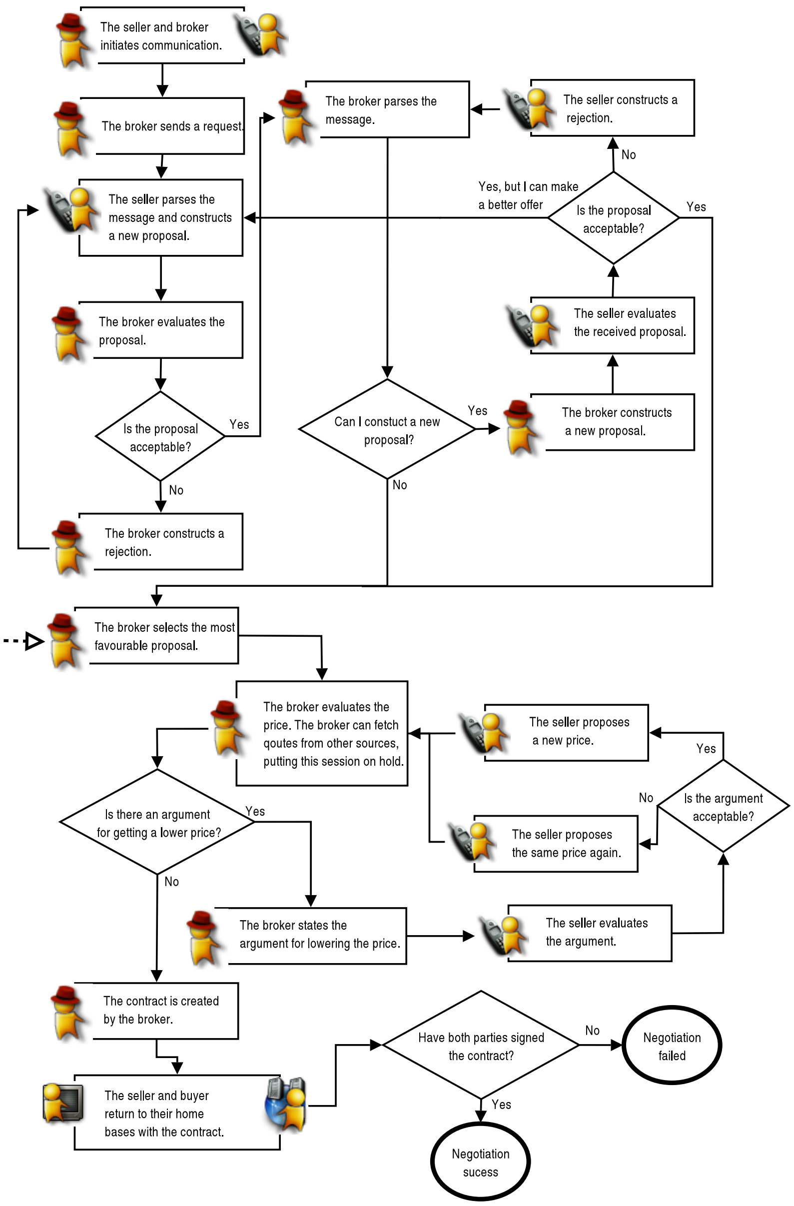

Figure 4.5: A flowchart of the negotiation process.

4.4 Pricing

A cornerstone in our project is pricing. Namely, how the various parties in the system value the resources and services. This is individual for every party in the system, and our task is to define a common way they can value the services, given their specific criteria.

As mentioned in section 2.1, every resource has a translated value in currency relative to every users preferences at the given time.

4.4.1 Variable vs. fixed prices

Prices are never static, as the market forces always push the prices in certain directions, all dependent on supply and demand[7]. Likewise, the market prices have a dynamic nature in our system.

We have two choices of pricing resources, *variable prices* or *fixed prices*. The variable price equals the market price, and the fixed price is agreed upon before a transaction is initiated. This choice can be compared to how the interest loan market works. You can loan money at a fixed interest, or a variable interest. Depending on market conditions, you can benefit or lose.

If a fixed price approach is selected, we can choose to either specify a price for each resource wanted, or a total price for all wanted resources. This is up to the seller or buyer. A total price can help security, as it hides the way that the seller/buyer prices the individual resources. Individual pricing provides transparency, and makes it easier to negotiate on individual elements.

Although it gets complicated if the price is set in resources instead of only currency. Therefore, we choose that you can only give a total price, when paying with resources. Also, both currency and resources can be used for payment, because, as noted in section 2.1, currency can be considered a resource.

If a variable price approach is selected, the market price for each individual resource is used, or any specified percentage over or under this price. Therefore, it becomes too complicated to set a total price for resources, if using the variable price approach. For every wanted resource, you have to specify at what rate you want the resources. For instance, say you want DISK at a variable price, then you can agree to buy/sell DISK for 90% of the market price (or any other percentage).

Also, WEGAS won't allow that resources can be paid for with the variable price approach, in other resources. This would mean that the market prices should be paired to find the appropriate relation. It is possible, but would be too complicated.

When using the variable price approach, it can be useful to have a mechanism for aborting a task, if the market price gets too steep. Therefore, the

[7]In the majority of market models.

buyer should be able to specify a *maximum market price* for the resources, he wants to pay for. A seller can then choose to keep processing a job[8] at a lower price within the limit, although the market price is higher.

To outline our limitations on payment methods, refer to table 4.3. The table explains which payment method can be used in which situation. *Individual* means that the resources are priced individually, and *Total* means that a total price is given.

Table 4.3: Limitations on payment methods. (C=currency, R=resources)

	Fixed price	**Variable price**
Total	R+C	-
Individual	C	C

When choosing a variable price approach, the market value of the given resource is important. The market price obviously determines what the user must pay for his resource needs. To make things more simple, we define that if you buy/reserve 30 GB storage, and you only use 10 of them, you still pay for the full 30, because they are reserved for you and cannot be used by others in the reservation period. Consider the scenario graphs depicted in figure 4.6. The market price is shown on the left and the disk usage on the right. Although only 32% of the capacity is used (on average), full price has to be paid, which is the reserved capacity times the market price (times the agreed percentage of the market price that is to be paid).

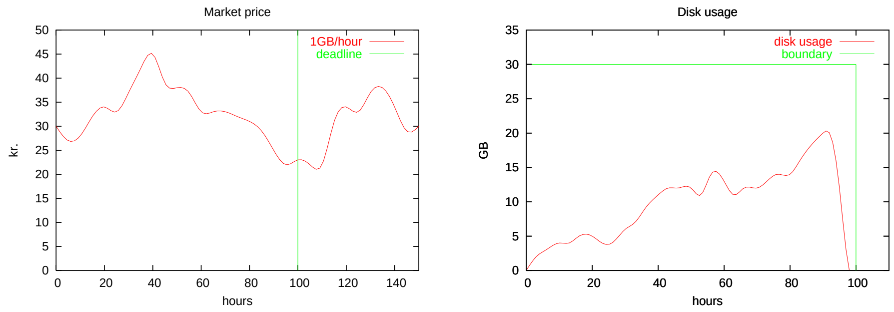

Figure 4.6: Variable price market and usage

These graphs also reveal that our granularity is *hours*. This is of course a somewhat arbitrary choice, as we could use any time unit, but it seems

[8]The jobs are assumed to be sanely distributed, so they can be started and stopped arbitrarily within reasonable time.

reasonable. The total price is calculated as the area in the left graph below the line, and before the deadline, times the reserved amount of resource. This totals to about 10,000 kr. in this example[9].

The other resources are billed in the same manner as in this example.

4.4.2 Determining market prices

Market prices are maintained on every local marketplace in the WEGAS system. These prices are based on the previous deals.

Normally, we will have deals that span multiple resources, and this can cause a problem because the marketplace is not necessarily aware of how the buyers/sellers value the resources. The marketplace could try to figure it out from the composition of resources and the price. This is not always possible, however, and surely not efficient.

Therefore, we suggest that the buyer tells the broker for every deal, how he values the different resources. This ensures that the agents won't publicise their motives, i.e. how much they value the different resources, which gives the marketplace the ability to adjust the market price accordingly.

In a fixed individual price scenario, the process value of the resources are given explicitly, and the process is therefore trivial.

When resources are used as payment, the buyer still have to tell the broker how much he values the bought resources. The resources used for payment are not considered in the market price, as it is used as payment, and therefore not considered bought goods.

In a variable price scenario, remember that the prices are given as percentages of the market price. Therefore, it is trivial to calculate the momentary values, and adjust the market price accordingly.

The variable price scenario creates some problems though. The market price increases when a deal of, for instance, 110% of the market price is signed. This will affect the resulting price the buyer has to pay for the resources. We think, however, that this adjustment is necessary for giving a truthful indication of the market price.

Another problem is when a grid services provider tries to buy his own resources at a greater price to invoke a higher market price. This can however easily be made impossible by the broker system, by not including the grid provider in the search results. But two grid providers can choose to sell resources to each other, thereby raising the market price. This is harder to control, but tools do exist which can identify deviant behaviour, like if the

[9]This seems rather steep, and might very well be, as it is a hypothetical example.

two providers exchange resources very often, and always to a higher price than the market price. This is considered out of scope, though.

Calculating market prices

There are several methods of updating the market price, when a new deal is signed. It can be based on *Moving-Average Forecasting Procedure*[16], with a window of w deals, which would take the last $n - w$ deals, to calculate the new market price, with the formula given in 4.1. n is the total number of deals, n being the latest deal. $p_{r,i}$ is the price for resource r in deal i.

$$M_r = \frac{1}{w} \sum_{i=n-w}^{w} p_{r,i} \tag{4.1}$$

The problem is that this method puts as much weight on the w^{th} deal, as with the latest deal. Therefore, we have chosen to use the *Exponential Smoothing Forecasting Procedure*[16], where the latest deal is weighed the highest. This procedure uses a factor α, which can be set according to formula 4.2.

$$w = \frac{2 - \alpha}{\alpha} \tag{4.2}$$

For instance, a α of 0.2 would involve the last 9 deals. We can now use this factor for calculating the next market price, based on the previous, in function 4.3. This is also easier to do, as we wont have to keep track of the previous w deals, just the last market price.

$$M_{r,t+1} = \alpha p_{r,t} + (1 - \alpha) M_{r,t} \tag{4.3}$$

The specific value of the window w, should be set experimentally, and after how dynamic and fluent the market prices should be.

4.4.3 Determining demand and supply

To determine demand and supply, an approach similar to the method of determining market prices is adopted.

The supply is determined on the grounds of the resource advertisements given by the sellers, mentioned in section 2.6.3. The supply number is updated at regular intervals to keep it up to date.

Whenever a deal is made, the reserved resources is mapped as demand for the period they are to be used. The demand is registered for that period, but when demand is queried upon from the users, it is always the current demand that is given. Otherwise, demand would be zero if queried for most future dates.

These figures should be public at the marketplace.

4.4.4 Seller expense analysis

To help a seller decide on price setting, its helpful to know where he breaks even[10]. He needs to perform an analysis of his income and expenses. In this section, we will briefly discuss how such an analysis can be performed.

To be able to calculate a break-even price, it is important to know the fixed costs, C_f, and variable costs, C_v. After investing in the equipment[11], the only variable cost that exists, is electricity.

The fixed costs is a constant expense over a time period e.g. wages, rent and write-off. In this calculation, it is assumed that all the fixed costs for a year, except write-off, can be collected in one variable, o.

According to Danish law, one should use the balancing method to calculate the write-off for machinery [18]. To calculate the yearly write-off percentage, p, for machinery that is bought for K kr., equation 4.4 is used.

$$p = 1 - \sqrt[n]{\frac{S}{K}} \tag{4.4}$$

In equation 4.4, n represents the number of years the write-off should be performed over. If you reduce the value of the machinery by the same percentage each time, it will never be worth zero. Therefore, machinery has a minimum value called the "scrap value", S.

The yearly fixed cost can now be calculated as,

$$C_f = o + (N \cdot p) \tag{4.5}$$

where N is the current value of the machinery. The sellers total cost, C_{tot}, can then be defined as,

$$C_{tot} = C_f + C_v \tag{4.6}$$

For a seller to break even, his total income from selling resources, I, must be equivalent to his total expenses, see equation 4.7. However, total cost is calculated on a yearly basis and since we use hours as our granularity, we need to transform C_{tot} from year to hour. It is not possible for a seller to

[10] Where the seller's profit is as big as his expenses [18]

[11] Servers, network connection, licences, etc.

use the equation to determine a final price based on this calculation, but it can be used to regulate the price to achieve some profit margin.

$$\begin{aligned} I > C_{tot} &\Rightarrow \text{Profit} \\ I = C_{tot} &\Rightarrow \text{Break-even} \\ I < C_{tot} &\Rightarrow \text{Loss} \end{aligned} \tag{4.7}$$

The point where he breaks even, $I = C_{tot}$, will therefore equal his reservation price, as he is probably not interested in deals with loss. However, this value can be adjusted downwards if he benefits from the deal in other ways. Also, because we are dealing with transient services, he could choose to make a deal with loss, just to minimise his loss.

Seller profit

A seller can state a profit margin in percentage for each resource, m_r, that he wishes each deal to have. To regulate the price to obtain the profit margin, we can multiply the price with a factor, which we will call δ_r here. Since it is possible to buy different amounts of resources in different time periods, you have to split up the total cost on each resource, $C_{tot,r}$, which can be done in many ways. One should define the fixed costs and the write-off for every resource, and do the calculations in equations 4.5 and 4.6 resource-wise.

One can argue that if a deal occupies a percentage L_r of a resource, it should also cover, at least, the same percentage of the resource cost. It is now possible to calculate the expected turnover, T_r for a resource in a deal (Q_r is the quantity of a resource, M_r is the market price of the resource and l_r is the reserved time):

$$T_r = (Q_r \cdot M_r \cdot l_r) - (C_{tot,r} \cdot L_r \cdot m_r) \tag{4.8}$$

Of course, if the expected turnover exceeds the profit margin, there are no need to regulate the price and δ_r should therefore be set to 1. If the turnover is lower than the profit margin, we need to increase the price. The price should be regulated with the percentile ratio between the wanted income and the expected income. Given the above assumption it is now possible to define δ_r.

$$\delta_r(t_r) = \begin{cases} 1 & \text{if } T_r \geq 0 \\ \frac{C_{tot,r} \cdot L_r \cdot m_r}{Q_r \cdot M_r \cdot l_r} & \text{if } T_r < 0 \end{cases} \tag{4.9}$$

This factor can be used to calculate the price, so the profit element remains. Another big factor is the utility functions, which will be described below.

4.4.5 Utility functions

Every player in the market has to have a utility function, to be able to evaluate the resources according to his current needs. The complexity of this function is completely up to the user. The only requirements for this function is that it takes a combination of resources, and outputs an estimated (personal) value.

Many elements can enter in to this function, such as demand and supply, trust levels, or meta-variables such as *convenience*. These things must be incorporated into the utility function, and customised to fit the users needs. Also, the utility function for the seller differs from that of the buyer, as he has different criteria.

4.4.6 Seller utility function

Our utility function should contain the factors that can affect the resulting price. These factors are joined in the utility function, which can be seen in equation 4.10. Actually, the result of the function $U_{s,r}$ is an indication of how the pricing formula must be regulated to provide discount or increased prices. Therefore, the seller is not necessarily trying to maximise his utility function, as normally found in the literature [36].

This utility function is for the seller, as denoted with a subscript s. This function depends heavily on how the seller has set his preferences, stating how willing he is to vary the price on certain factors.

$$U_{s,r} = f_1(d_r, s_r) \cdot f_2(t_{min}, t) \cdot f_3(D_{s,r}, l_r) \cdot f_4(R) \tag{4.10}$$

The factors used in equation 4.10 are explained in table 4.4. The utility function must be resource-wise, as the demand, supply and start date depends on the resource context.

Table 4.4: Utility parameters

Var	Name	Explanation
d_r	Demand	Demand of the resource r, as stated at the marketplace.
s_r	Supply	Supply of the resource r, as stated at the marketplace.
t / t_{min}	Trust	t is the trust of the other party, while t_{min} is the minimum level of trust, before it affects the price negatively, i.e. the soft trust limit.
$D_{s,r}, l_r$	Start and length	This is the start date and length parameters of the resource. Also, see the scheduling section 3.3 for more information.
R	Payment method	A factor dependent on the payment method used. Varying according to the preferred method.
Q_r	Quantity	Quantity ordered of resource r
M_r	Marketprice	The current marketprice of the resource r

As the reader may have noticed, everything is enveloped in separate functions, f_1 thru f_4. These functions are the responsibility of the seller to provide, as these functions evaluate the given variables and outputs the factor that should influence the value of the utility function.

The first variables are the demand d_r and supply s_r. The seller should ensure that the price is adjusted accordingly to these factors. The next ones are the buyers trust level t, and the sellers defined *soft minimum trust* t_{min}. The minimum trust is meant as a minimum *before* it has any effect on the price, like stated in section 4.3.1.

$D_{s,r}$ is the start date and l_r is the length of the resource reservation. The function f_3 is the supposed to rate the convenience of when the resources are booked. R is the payment method used.

How the functions should behave, or look, is as mentioned up to the seller. However, we will test various possibilities in chapter 6, where we will try to experimentally define some of the functions, so we can have a generic utility function.

When we have a value for $U_{s,r}$, we can calculate the price of every resource as seen in function 4.11:

$$P_{s,r} = Q_r \cdot M_r \cdot l_r \cdot U_{s,r} \cdot \delta_r(T_r) \tag{4.11}$$

And then the total price, if this is wanted, trivially as in function 4.12:

$$P_s = \sum_{r=1}^{n} P_{s,r} \tag{4.12}$$

However, the seller can set a minimum price $P_{s,min}$ (or $P_{s,r,min}$) he wants for his resources[12]. This price cannot be violated, and overrides P_s (or $P_{s,r}$). This mechanism can be employed whenever some factors or market conditions force the seller to do it.

4.4.7 Buyer utility function

The premises for the buyer are a little different, and somewhat simpler. This function is more reliant on the input of the user, as he doesn't (necessarily) have to return his investment in some equipment. Therefore, we are stuck with the question: "How much do you really want this? Put it in integers, please."

Normally, the buyer would have some general idea of what the price should be, maybe based on the marketprice also. The buyer can also define a upper spending limit, as the purchaser of services normally are governed by his senior economy manager, who grants him the money for projects. On a wider scale, economic feasibility studies can be made, like a cost-benefit analysis, to determine the preferred and maximum price.

Also, the buyer can take into consideration what the wanted task would cost in time and money, if he were to do it himself[13], or get it done outside of WEGAS context. These factors are again subjective and hard to determine.

Like the Cobb-Douglas buyer utility function, mentioned in [36], our buyer utility function incorporates the notion of *importance*. This factor is a number between 0 and 1, which expresses how willing the buyer is to spend more than expected. The more important the problem, the higher the willingness, obviously.

However, we can also define several other factors that influence the function, such as the start date and a list of preferred sellers.

To make things easier to rate for the user, we suggest that some scheme is used to abstract the values between 0 and 100 to commonly used rating words, such as "Very much" or "To a small degree". This can be a tool for helping the user when he has to decide on the many elements.

The buyer needs to define which start date he prefers[14], $D_{s,r}$, the *hard deadline*, $D_{d,r}$, and the *earliest start date*, $D_{e,r}$. The two latter is given in

[12] This equals the reservation price

[13] Especially relevant for grid providers that wants to buy extra resources

[14] The amount of time used by each resource is determined beforehand.

the user preferences as mentioned in section 4.3. To calculate the *latest start date*, $D_{l,r}$, we just need to subtract the amount of time the resource is to be reserved from the hard deadline, $D_{l,r} = D_{d,r} - l_r$.

The graph in figure 4.7 shows how a function can value the starting date. The closer to the preferred date, the more value it has for the buyer.

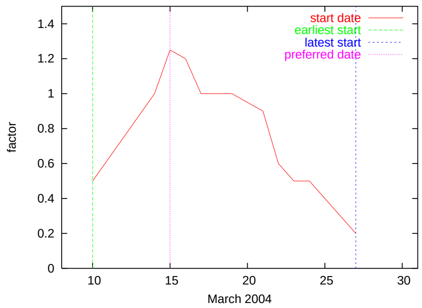

Figure 4.7: Startdate function

We can now suggest an utility function for the buyer, stated in equation 4.13.

$$U_{b,r} = f_1(D_{s,r}, l_r) \cdot f_2(J) \cdot f_3(\{S_1, S_2, \ldots\}) \cdot f_4(t, t_{min}) \cdot f_5(R) \qquad (4.13)$$

The list, $\{S_1, S_2, \ldots\}$, is the preferred sellers list mentioned earlier. It can be a set of preferred sellers, all increasing the utility with the same factor, or a set of tuples, where every seller has his own increasing factor. It might as well also contain sellers that is not preferred, thus yielding a decreasing factor. This list can also be classified as an internal trust register. t is the official trust level of the seller, which is supplied by the broker.

J is importance of the task, i.e. how critical it is for the buyer to have completed. This affects his willingness to spend, naturally. This is one of the values that could benefit from having an abstraction scheme, with values mapped to descriptions.

With the utility function in place, we can calculate the price that the buyer is willing to pay for the required services, $P_{b,r}$, in 4.14. This resembles the seller function (4.11) somewhat.

$$P_{b,r} = Q_r \cdot M_r \cdot l_r \cdot U_{b,r} \cdot f_6(d_r, s_r) \qquad (4.14)$$

This function is different in some respects, mainly on its usage. The buyer version is used for comparing the given price by the seller to the preferences given by the buyer. Therefore, we have to have demand and supply included in the formula. This is however up to the buyer how the function, f_6, let's these factors affect the price.

Like the seller, the buyer has a limit on the price, a maximum price $P_{b,max}$, which cannot be violated. The total price P_b is calculated the same way as for the seller in equation 4.12.

All the functions, f_1 through f_6, and the utility function is shipped to the broker along with the needed values. The buyer has agreed on giving the broker the power to act on his behalf. Although, depending on the implementation, the functions could be invoked remotely on the buyer by the broker to preserve more information hiding. However, the broker is acting on the buyer's behalf, and has no immediate reason to doubt the integrity of the broker.

4.5 Contracts

After the agents have finished the negotiation and agreed upon the details of the transaction, they need to make the other part liable to the agreement. For this, we suggest an electronic contract. In this section we explain how such a contract should be designed. Finally, there will be a brief discussion about the legal status of electronic contracts signed by agents.

4.5.1 The contract

As mentioned, the purpose of a contract is to keep the two parties responsible for what they have agreed upon. Since a contract is often used to settle a dispute between parties, it is important that the contract includes all elements of the agreement. Our suggestion for an electronic contract contains the following elements:

Parties It should be specified whom the contract concerns. This is stated by name and contact information of the two parties.

Services This element describes which services are expected to be delivered and their quantity. For each service it should be clear who is the producer respectively consumer.

Price The total price, if any, of the services mentioned in the **Services** element.

Payment This element describes the payment method. The advance payment is also included here. Also, the recipient and payer are specified.

Dates The start and end date. The start date indicates from when the services are accessible for the buying party, while the end date is the date where the task should be finished, or access is revoked, and the seller are no longer obligated to provide services for the buyer.

Jurisdiction Jurisdiction states which country's laws that should be applied in case of conflict. Jurisdiction is one of the uncertainties of contracts over the Internet according to [22], since:

> [...] the Internet is a form of communication that rises above spatial boundaries. Its domain flows indiscriminately across international boundaries as easily as it flows across the street.

This creates a problem, since it is harder to determine where the contract was formed and which laws that apply. Therefore, we suggest that it is written in the contract that Danish law is used. This element is therefore non-negotiable.

Dispute This element explains which authority that should settle a conflict. It assumes that the users of this system are mainly companies or organisations, which often appreciates discretion in case of a conflict. Thus, all conflicts are settled in an arbitration where the record of the proceedings are not public, compared to a normal court where all records are public. We suggest that the International Chamber of Commerce (ICC) arbitration service is used, since it is the world's foremost institution in the resolution of international business disputes [5].

Transaction details For each service described in the **Services** element, it is clarified how the service are expected to be used, such as platform information or other requirements.

Signature Both parties need to sign the contract electronically. This is done with certificates, the same certificates that identify the agents.

The above contract should not be considered as an complete contract, since our legal knowledge is very slim. A final product should contain a standard contract issued by a lawyer. The mentioned contract elements are the ones we have identified, in collaboration with [25], to be relevant for this project.

4.5.2 Legal liability of agents

Whether a person is allowed to sign a contract or not, is clarified in civil law. However, the field of contracts entered by software agents is more ambiguous. The issue is discussed in [11] and it is pointed out that:

> The legal problems surrounding the making of contracts via computers have been examined systematically. There is now no doubt in stating that a contract can be validly made via computer technology.

Moreover, [11] argues that a software agent has the same legal status as a software program, which according to the law is a "thing" and a "thing" does not have any legal status. Therefore, it is important to explain what role the software agents can play in the negotiation. Since an agent is not a legal person, it is not possible to issue a power of attorney to an agent and therefore not possible for an agent to enter a contract on behalf of someone. However, all members should accept the agents as negotiators. But, due to current regulations, the agents should return to its home base and have the contract manually signed by the user, to ensure its legal status.

4.6 Conflict management

Conflict management is the concept of identifying possible conflict scenarios within the system and taking appropriate action against them. A conflict scenario refers to a situation where the buyer and/or seller do not behave as expected, which could result in a possible system failure or transaction failure.

In this section, we will focus on conflict scenarios that can occur inside the *running period phase* of the transaction process, see section 2.6.1. This is where the buyer and seller interacts, and is therefore outside of the broker's domain. The conflicts that arise apart from these are all in the jurisdiction of the broker system, and will be treated inside this context.

4.6.1 Transaction scenarios

A transaction scenario represents different types of transaction possibilities. These scenarios are of importance since different types of transactions can result in different conflicts. We have identified the following scenarios in the model:

1. Buyer delivers a *full package* [15] to the seller. The results are delivered when the task is finished.

2. Buyer delivers a full package to the seller. The results are delivered gradually, as they are computed.

3. Buyer delivers executable code, and data needs to be gradually fetched from the buyer system, or some other data source.

4. The buyer acquires a service from the seller, and uses it without any interaction from the seller.

These transaction scenarios can now be used for devising the various conflict scenarios.

4.6.2 Conflict scenarios

The transaction scenarios mentioned in the last section has unique conflict scenarios that only exist for them. However, most of the conflict scenarios are valid for all transaction scenarios. In appendix A, all conflict scenarios we have identified are listed together with the transaction scenario it refers to. The last column of the table states the action taken if the conflict occur. The decision of what action to take, is based on the severity of the conflict and its consequences.

Although the table includes the most important conflict scenarios (to us), it is impossible to list all possible scenarios. Also, from a organisational point of view, it is difficult to get an overview of a system that manages each scenario rather individually. Therefore, we propose that the system has a small set of somewhat universal rules, which it can use to make a decision of which action to take, given a situation. Not only the system should use the rules, but also the users, in order to know how the system reacts in case of a conflict.

The first rule, 4.15, handles payment conflicts. Since the reason for such a conflict can be very unclear, these problems should be settled according to definitions in the contract.

[15] A collection of executable code and data

§1 Breach of payment agreement
The case that one of the parties does not pay the agreed price or one of the parties collects the advance payment and vanishes, is a severe breach of contract and handled according to the **Dispute** element in the contract. (4.15)

Rule number two, 4.16, handles cases where one of the parties does not behave as expected. The reason for that can vary, e.g. a system may be down temporarily. There have to exist a certain gentlemanliness between the parties, even if one of the parties does not behave as he is supposed to, the other party will always wait a certain time period before he cancels the transaction. Also, the parties should be interested in keeping things rolling to avoid loosing too much of the invested time and resources.

§2 Behaviour issues
If one of the parties disappear or do not act as expected, one should wait until a reasonable time has elapsed. The waiting party is entitled to receive compensation from the disappeared party. (4.16)

If one of the parties tries to harm the other party, it is a serious problem. In addition, appearing under false identity is also a serious issue. However, to discover that a party tries to harm another party can be difficult, and it should therefore be settled outside the system, also because the severity of the attack varies. The third rule (4.17) describes this situation, which could end with suspension of some of the inflicted parties. But until proven guilty, the accused party is still able to function within the system. This state should maybe be evident to other parties and result in temporarily lowered trust levels.

§3 Misconduct
In case of one of the parties behaving or appearing with malicious intent, this party is liable for damages to concerned parties. The conflict should be settled outside the system and the accused party should be, if found guilty, suspended from the system. (4.17)

For a user to have his task executed on another computer, both the host system and the task has to follow the agreed standards, e.g. the system is badly setup or the task package is not properly formatted. In the case that one of the parties break the standard, he is liable to compensate the other party. The damaged party can, however, be sympathetic and let the other party correct the error and continue the transaction. The latter will also create goodwill and is recommended. This matter is summarised into rule the fourth rule, 4.18.

§4 Compatibility issues If a system or a task does not follow the agreed standard, the liable party has to compensate the other party. It is up to the other party to decide if the transaction should continue or not after correction of the error.	(4.18)

Of course, the above rules does not cover all breaches of a contract and therefore there is a need to have a general rule for contract breaches, 4.19.

§5 Other contract breach If a contract breach is made, the liable party has to pay for damages to the concerned parties. It is up to the party that suffered the damage if the transaction should continue or not.	(4.19)

In the above rules, we have not specified the magnitude of the compensations. This is because each breach, even if it concerns the same rule, can be very individual, so we consider this as out of scope for this project. Nor can we ensure that all scenarios are covered. Nonetheless, we believe that the rules cover most conflicts scenarios and gives good pointers on which actions to take.

4.7 Summary

Our chosen negotiation model is the *argumentative negotiation* model, which lets us use arguments and reasons when proposing and rejecting. This model is adapted to suit our needs as written in our ontology.

The ontology defines a language, L, for describing resources and their variables. This language is a higher-order predicate logic, and can be used to combine all the resources in a single statement, like describing resources as the value of the price variable.

This language is complemented by the language CL, our communication language. This language wraps the L descriptions into a message type, which agents it concerns, etc. Also we have created a small language R for expressing reasons when rejecting a proposal.

CL can be expressed as a tree, thus making it easier to parse, and more structured. This also enables us to communicate using XML, and represent everything in a more uniform way.

Then, we have devised a strategy for the agents, enabling them to process the protocol described in the ontology. The strategy uses *preferences* for the user to parse a message, and send an appropriate response. Algorithms are given on how the agents should negotiate.

The resources have different values for the various users at given times. This evaluation of resources, and the way they are sold, is performed using *utility functions*. These utility functions incorporate the various factors that are important for the user, and can have implications on how much he is willing to pay for the resources, or sell them for.

The utility function factors can include such things as the sellers total cost of running the services, how important the buyer thinks his task is, how well the start date of the resources fit in with the schedule at the service provider, and so on.

We have defined that the demand and supply is calculated and held at the marketplace for easy reference. These factors are complemented with a current market price for the resources, which is influenced of the deals made in the community, using a running average.

When negotiation has ended, the contract is signed, containing the important and necessary elements for the transaction to proceed. If any conflicts should occur, we have devised 5 general rules for how they should be handled.

Chapter 5

Security

For the users to rely on WEGAS, it is crucial to have a certain level of security. The security level should ensure that the users can sell and buy resources in a secure manner and without interference of unauthorised entities.

Firstly, we analyse our system, and try to identify vulnerabilities. These vulnerabilities are then discussed and possible solutions are suggested, for the agents in section 5.1, and the data they transmit in section 5.2.

Our analysis is based on the model presented in [30]. This model is used, as it ensures a complete analysis of the system, and is the recommended model by the National Institute of Standards and Technology (NIST). In short, it is a risk analysis that identifies the vulnerabilities of the system and provides a method for rating them. The first two sections will discuss the necessary level of security, based on a security analysis we have made, see appendix E.

The trust model used in the system is then explained in section 5.3. The section explains what we define as trust in the system, how the evaluation of users should be carried out, and how the users should rate each other.

The certification authority is discussed in section 5.4. We will briefly discuss its role in the system, and how it should work in collaboration with the other elements.

5.1 Marketplace and agent security

According to the risk analysis, the vulnerability that causes the greatest threat is a badly setup firewall on the marketplace. The marketplace could be accessed by rather simple means which could cause serious damage. This vulnerability can be compared with other security issues on typical server environments, such as not updating software or using weak passwords. A gullible system administrator is also a security threat, since he would be open for social engineering attacks. These kinds of security issues are not discussed here, since they do not concern WEGAS directly, but are well-known security problems, and good methods for solving such problems have been presented in numerous books.

A more difficult security issue to handle is agents executing malicious code after being accepted to the marketplace. When the agents execute their code on the marketplace, the code is executed in a *sandbox*[1], which means that it can only cause a limited amount of harm on the marketplace. If two agents execute in the same sandbox, it is theoretically possible for one agent to read the other agent's memory. However, it is very difficult to do so, since they don't know at which addresses certain variables are stored. This issue is heavily depending on the chosen technology for the agent system, although the agent system should ensure that the agents can keep their privacy.

One could imagine that the marketplace could attack the agents, but this involves the marketplace being compromised first, so we wont consider such attacks.

An agent with malicious code can either originate from the user that created the agent with malicious intent, or the agent could have been tampered with after leaving its home base. The (technical) effects of the two cases are the same, and the security mechanism for handling it is described above. Nevertheless, it is a more serious security breach if the agent is tampered with, since confidential information may have been exposed and the goal of the agent could change. As suggested by [26], we consider a mixture of code mess-up and cryptography for preventing attackers from modifying the code of the agents.

The idea with code mess-up is to transform a normal piece of code into something less readable and incomprehensible. Creating mess-up code consists of three major parts:

Inserting dummy code To make the code more complex, dummy code in form of dummy classes or methods can be added, which have nothing to do with the functionality of the agents.

Altering values of numeric variables Changing the values of all numerical values by multiplying them with a random number that is kept in the home base and the marketplace.

[1]A safe environment on the server, with limited system access

Altering values of string variables The string is altered by multiplying the numeric value of every character in the string with a random number that is kept in the home base and the marketplace.

By using encryption, one can protect the code in a secure manner. The Advanced Encryption Standard (AES) is the most modern conventional encryption system at the time of writing and can use a key length up to 256 bits[2]. AES is powerful enough to secure the code of the agents. An AES key is generated by the agent's home base upon the agents creation and is delivered to the marketplace by public key encryption. It goes without saying that the keys have limited lifetimes.

5.2 Data security

Not only the agents need protection from intruders during their travel. The security analysis shows that a similar vulnerability exists when task data is sent back and forth between the seller and the buyer systems. To protect the task data from exposure, we can use the same security mechanisms as the ones for protecting the agent's code. The local broker agent generates a conventional session key when the contract is signed, which both agents bring back to their home bases, and use for encrypting and decrypting task data.

The last part of the system that require encryption to ensure integrity and confidentiality, is communication with the authentication authority. In figure 5.1, communication parts that has a need for encryption are illustrated.

Another area the security analysis brings focus upon, is authentication. Authentication is needed for the marketplace to authenticate the agents, and for the agents to authenticate the marketplace. How easy it is for a third party to obtain a valid authentication certificate depends on which technology that is used. Authentication is also shown in figure 5.1.

Replaying messages during the negotiation is not a problem in WEGAS, since each message has a timestamp attached to it, making it easy for the agents to control the validity of the message. To avoid that an agent can be captured and replayed at a later time, each agent should have a reasonably limited lifetime.

Nonrepudiation issues can occur in two areas in the system. If a user denies having sent an agent to the marketplace, and a user denying having

[2] `http://csrc.nist.gov/CryptoToolkit/aes`

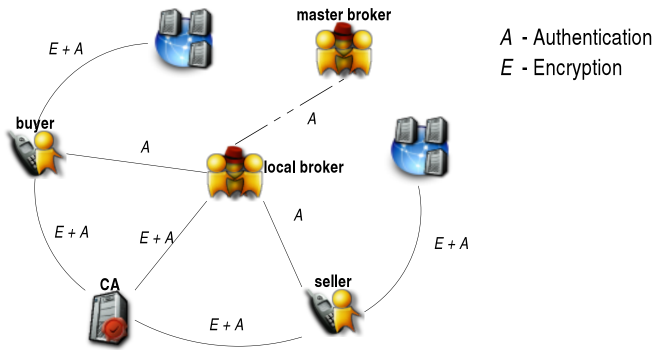

Figure 5.1: Communication security

entered a contract. These issues are solved with digital signatures, identifying the agents and the signature in the contract, without doubt.

With the above security mechanisms, we believe that WEGAS can provide a secure platform for negotiations and transactions and ensure confidentiality, authentication, integrity, nonrepudiation and access control [29].

5.3 Trust model

> *You may be deceived if you trust too much, but you will live in torment if you don't trust enough.*
>
> Crane, Frank

Even though it is hard for a user to decide how much he should trust another, it is important that the users of this system have a way of measuring trust. Lack of trust among the users will complicate the business relations. To add an element of trust, and thus some extra security, a trust model is required.

The trust model we suggest, is based on earlier behaviour of the users, rather than on recommendations as used in PGP[3]. Because of the wanted flexibility of the system, we require more detailed information for denoting trust.

As described in section 2.6, the users evaluate each other after completing their transaction. Partly based on these evaluations, the user is assigned a certain trust level, or *trust index*. This section will describe the trust model

[3]PGP Trust model, `http://www.pgpi.org/doc/pgpintro/\#p20`

by discussing the decision factors, extensions of the model and the basic rules.

5.3.1 Decision factors

The trust index represents the user's earlier behaviour. A user's behaviour is a collection of several sub-behaviours, which represent behaviour in different areas. The trust index is an weighted average of these sub behaviours, therefore they can be referred to as *decision factors*. The model has three decision factors:

Success rate(S) A ratio indicating how many of the performed transactions were successful.

User evaluation(U) The mean of the evaluations that the users deliver after completed transactions, ranging from 0 to 100.

Variance(V) The number of different users the user has interacted with.

Also, the term *successful transaction* needs to be defined clearly, as users must agree on this:

$$\boxed{\begin{array}{l}\text{A successful transaction is a completed transaction, in} \\ \text{which none of the elements in the contract are violated.}\end{array}} \tag{5.1}$$

Calculation

Seeing that the decision factors do not have equal importance, they should not have the same impact on the trust index. To compensate for this, they are assigned different weights in the calculation[4].

$$0.3S + 0.5U + 0.2V = \textit{trust index} \tag{5.2}$$

The letters indicate the decision factors, see description above, and the number before the letter is the weight in percent, e.g. the variance factor is weighted 20% of the total trust index.

[4] The weighting is done by the authors, and is only based on what we believe is relevant for the system and has no immediate scientific basis

Notice that the two decision factors, success rate and user evaluation, are indices while the variance factor is a number. For the equation 5.2 to make any sense, we need to convert this variance factor to a value between 0 and 100. Table 5.1 illustrates our chosen conversion from a variance number to a variance index.

Table 5.1: Conversion from variance number to index

Different Users	**Index**
1-4	20
5-9	40
10-14	60
15-19	80
20+	100

This conversion is of course based on our expectations about the size of the systems. A more dynamic solution can be chosen, also for the model to be more fair, as a user with specialised services, maybe occupying 80% of the market of that service, can still have an index value of 20 if the market only consists of 5 users. But now, we can see from equation 5.2 that the trust index spans from 0 to 100, where 100 is the best and 0 the worst.

The reason for weighting the user evaluation higher than the other two factors, is because user evaluation is the only one that measures the quality of a user, while the two other factors measure quantity. Thus, with the current weighting, the distribution between quantity and quality is dived equal. This could of course be revised, if this equal division creates problems.

5.3.2 Trust model details

The three elements of the trust model can be seen as different elements that offer different levels of useful trust indices. Instead of only having one trust index, one can split the trust level up and get a trust index for each decision factor. In addition, each decision factor can have its own factors, for which one can also get a trust index. This linked trust model can, in theory, evolve to infinite large trust tree containing very specific information of the users earlier transactions. However, the last element in a branch of a trust tree, a leaf, does not have a trust index, instead it contains some value of some data type. The value represent a direct fact that has not been set in relation to something else and can therefore be considered independently.

Figure 5.2 illustrates the trust tree for this system. The maximum depth in the tree is three, incorporating very specific transaction details. Again, the

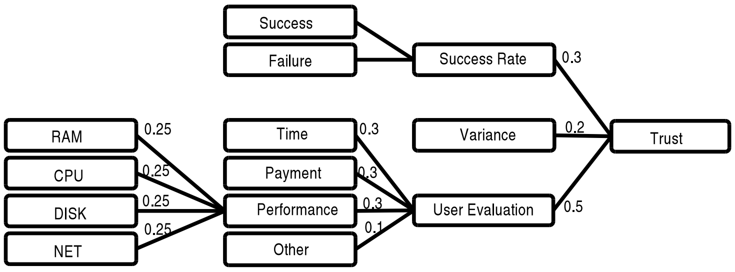

Figure 5.2: Trust tree

weighting in the tree is done by the authors and based our chosen priorities. However, an exception is made in success rate since it is not rated on a scale from 0-100, as the other elements. The success rate is calculated using equation 5.3.

$$\text{Success Rate} = 1 - \left(\frac{\text{Failures}}{\text{Successes} + \text{Failures}} \right) \cdot 0.8 \tag{5.3}$$

The reason why the weight of successes is 20% higher than failure, is because the failure of a transaction can be the fault of the other party. This way, we reduce the damage caused by bad transaction partners.

The detailed elements in the trust tree is described below, in table 5.2.

Table 5.2: Elements in trust tree

Depth 1	**Depth 2**	**Depth 3**	**Explanation**
Success Rate	Success		Number of successfully completed transactions.
	Failed		Number of transactions that has not been completed successfully. A transaction is treated as successful if the transaction are completed without complications.
			Continued on next page

Table 5.2 – concluded from previous page

Depth 1	Depth 2	Depth 3	Explanation
User Evaluation	Time		The user evaluates how he believes that the other party have respected the given deadlines.
	Payment		The user evaluates how well the payment procedure was handled.
	Performance	CPU	The level of processing performance, proportional to the amount of performance purchased.
		RAM	The amount of RAM made available, proportional to the amount promised.
		DISK	The amount of storage available, proportional to the amount promised.
		NET	The bandwidth available, compared to the amount promised.
	Other		If other complications or good things occur, this is the rating category for it.
Variance			Rating of the users variance, as described above.

The performance evaluations should be carried out automatically, using some kind of rating scheme. We will not investigate this further, due to time limitations.

General rules

We define some general rules concerning the usage of the model. Firstly, the users receive a start index for each decision factor the first time they register. This trust index is set to 25, which is a bit under the medium trust index (50). A user has to earn his trust, but in the same time, the rating is not so low so he doesn't have a chance to initiate transactions with anyone.

The second rule is that the initial trust index for success rate does not change until the user has fulfilled at least ten transactions. If he is below ten transactions, you cant make a fair appraisal of the situation.

The third rule is similar to the second rule. The decision factors, variance and user evaluation, does not change its initial trust index until after the user completed six transactions. We chose a lower boundary number, as these indices are more complex than the success rate, as they consist of more sub-elements, and are therefore more balanced in nature.

The proposed trust model does not exclude that a user might get an unfair trust index. Such a scenario can appear if two users make a transaction that ends unsuccessfully. Both users' trust index will be degraded even if it is only the fault of one of the users. It appeals to the users honesty when they evaluate each other, since both users can agree (in person) on reporting the transaction as successful, thus bettering their trust index. It is clearly defined what a successful transaction is, so this can be an issue for policing the execution of the transaction. Also, spot checks can be made to control this problem.

Another related problem is when the parties in a transaction evaluates the transaction to be successful and unsuccessful respectively. This is however resolvable, as the given definition is clear[5].

Even though the model has its flaws, it provides a good way for the users to extract information on multiple levels of complexity. However, this trust model exists to provide information about past behaviour, as noted. It is free for the various negotiation parties to use other internal trust models for deciding.

5.4 Certification Authority

The Certification Authority (CA) is responsible for authorising the users. Any potential user of WEGAS have to be identified by a CA to be able to register as a WEGAS user. The CA deals out certificates to identify the users.

The CA can be included within WEGAS or be an external trusted CA. The model will support both, just as long as the CAs used are trusted by the participating users.

The communication to and from the CA should of course be heavily encrypted, to avoid private keys being intercepted. And it should also be clearly defined in the certificates who the certificate is issued to, and where they can be reached.

5.5 Summary

A security analysis based on a model from the National Institute of Standards and Technology (NIST) have been made, in which the system's vulnerabilities have been identified.

[5] Which means that the contract must be clear and undisputable.

To protect the marketplace from malicious agents, the agent are executed in a sandbox, which is a safe environment with very limited system access. The agent's code and the messages sent from and to the agents home base, are protected by using encryption. The system needs a certification authority (CA), for identifying users of the system.

A trust model that can reflect a user's behaviour in the system, is proposed. The trust model consists of three decision factors, from which a trust index can be calculated. Using the trust model's different levels, a user can retrieve specific information about the other party that can be used as arguments in the negotiation.

Chapter 6

Implementation & test

This chapter presents the parts we have chosen to implement of WEGAS and which tests that are performed to support and prove our model. The main purpose of the chapter is not to illustrate how each part of the implementation works in detail, but rather to investigate the presented ideas, and see if they can be individually tested.

The chapter will start by briefly presenting the implementation of parts of WEGAS. The choice of technologies are outlined. Next, the implementation is tested and different setups are tried. Based on the things presented in the chapter, we will discuss the validity of the implementation.

6.1 Implementation

WEGAS is a rather large system to implement, with many features. Therefore, we will not aim for a full implementation. The idea with the implementation, is to illustrate how some of our assumptions on how WEGAS should be constructed, will work.

Because our focus is the negotiation and pricing, we have chosen to avoid the Grid aspect altogether. Such things as scheduling and available resources are simulated to provide an easy interface for the things needed in the negotiation, but the actual underlying Grid software is not implemented.

The certification authority interface is also simulated, as this is not our focus.

We have not created the hierarchy infrastructure of marketplaces, instead we chose to concentrate on a single community. This excludes the part of the negotiation where the buyer can ask the broker to seek a better offer at other marketplaces.

Before addressing the implementation aspects of WEGAS, the choices of technology is discussed.

6.1.1 Technology used

WEGAS should work across many architectures and platforms. Therefore, it would be convenient to use a multi-platform language as basis, like *Java*. This will ease some integration efforts. Besides Java, we introduce Aglets and JDOM.

Aglets

WEGAS uses electronic agents to perform negotiation, prediction, information collection, and many other things. We investigated some different packages that provides agent capability, such as *Concordia*[1] and IBM's *Aglet project*[2]. Both projects are built upon the Java framework. The Aglet project is available for general public, whereas Concordia is not. Therefore, we chose to use Aglets.

Aglets can be considered as Java objects which can move between different hosts, and, most importantly, preserve its internal states.

To create, monitor, dispatch and dispose of agents, a server application written using the Aglet framework, called *Tahiti*, is used. It is distributed with the Aglet distribution. Ideally, the server should be implemented from scratch, at least in the case of the marketplace. It could then provide services and interfaces, which we have chosen to put in the hands of the master broker.

Aglets are now listed as a project under the sourceforge community. The last version of the Aglet framework were released on February 2002, so further development of aglets are uncertain. However, the number of downloads and activity on its homepage show that the interest still exists.

JDOM

As described in chapter 4.3.2, the proposals, requests and rejections are constructed according to a certain tree structure. To describe this tree structure in a data structure, we have chosen to use XML, as advocated in the mentioned section. Also, we use XML to describe and store other data structures, such as the preferences of a user.

[1] Concordia2 project, `http://www.merl.com/projects/concordia2`

[2] `http://aglets.sourceforge.net`

To facilitate tree operations, we have chosen to use the JDOM package. JDOM is an open source Java API for parsing and performing operations on XML documents. w3c also has a Java integration package, however it is to extensive for our needs. JDOM is written in and for Java and is conformant with the Java code conventions and class libraries.

6.1.2 Proposed vs. implemented

As mentioned in the beginning of this section, WEGAS is not implemented fully. Therefore, we will briefly clarify which parts that have been implemented.

Both the seller and master broker is executed on the same Tahiti host, which means that the seller does not have to be mobile. However, to test mobile agents in the prototype, we have chosen to let the buyer start from another Tahiti host, and then dispatch to the marketplace Tahiti host, where the other agents reside.

A simple communication protocol is implemented to have the agents communicate properly, and know who sent the messages. The negotiation is performed as proposed, except from the deal refinement steps. The reason why we didn't implement the last negotiation phase, is because we believe that it does not show any interesting points. An agent will therefore not return to his home base with a contract as described, but with an collection of proposals which have passed through the main negotiation phase.

6.1.3 Overview of packages

Here, we will explain how we have chosen to design the WEGAS prototype. We hope that it will become clear how our vision of how a WEGAS negotiation would proceed. The prototype will be explained package by package.

WEGAS is built upon 7 packages to match the different components of the system:

agent Here, all main classes for the agents are located. They control the agents actions, as well as their inter-communication. Only commerce agents are implemented, as mentioned earlier. All agents extend the abstract Aglet class.

ca CA is short for Certification Authority, and the package includes classes for describing, registering and verifying the systems users. A user is described through the User object, which contains identification information about the user. Also, the trust model, described in 5.3, is the responsibility of this package.

gsp The Grid Service Provider package, which is solely used by the seller. It contains administrative interfaces to his schedule and for predicting service availability.

marketplace Simulates the marketplace interface for the master broker. Among its responsibilities is to administrate incoming advertisements from the seller and to calculate and update the market prices. Even though only the master broker can make an instance of the a marketplace, the other users use it to store marketplace information they have received from the master broker[3]. For standardising the appearance of a task, the marketplace package provides a class for this, which is used by all users.

negotiation This package contains all classes concerning the negotiation in WEGAS. The package has a class for each message in our communication language, CL. All negotiation are performed by the agents using and instantiating classes from this package.

negotiation.utility This package consist of two parts, a user functions abstract class and the utility functions. The user functions classes are declared abstract, since each user must have their own implementation of their user functions. The utility functions part provide functions for the users to calculate prices and utility as described in sections 4.4.5 through 4.4.7.

test The last package contains implementations of our suggested user functions.

For a more detailed overview of the packages, class diagrams of the above packages are shown in appendix F, and Java documentation are found on the CD, as well as on `http://wegas.mufaza.dk/javadoc/`.

6.1.4 Communication protocol

In the prototype, the protocol is primarily used to locate other users and send negotiation messages. However, in a full implementation, the protocol

[3]This is possible due to each agent (or Aglet) having its own executing environment.

would also be used to ensure security, such as verifying users, assigning them certificates, etc. A sequence diagram of a typical communication process using this protocol is illustrated in figure 6.1. SUser denotes the seller's user object, and BUser is the buyer's user object. All messages are packaged with information of the sender.

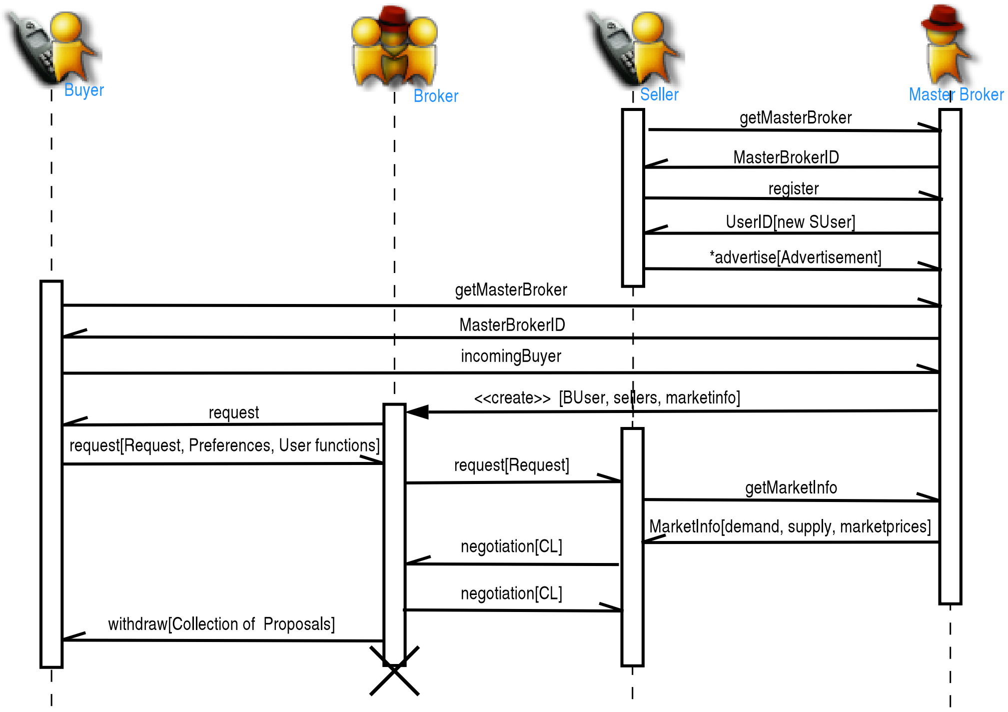

Figure 6.1: Sequence diagram utilising the communication protocol in WEGAS

The first time a seller enters a marketplace, he is not able to distinguish the master broker from the other agents. Therefore, he broadcasts the `getMasterBroker` message to all the agents on the marketplace. All agents ignore the message, except the master broker who sends his user object in return. The master broker registers the seller in WEGAS and the seller sends his advertisements, describing what he is selling.

A buyer identifies the master broker the same way as the seller. After the master broker has registered the incoming buyer, by the `incomingBuyer` message, he creates a broker given the current market information, the buyers user object and a collection of sellers denoting all active sellers[4]. However, if no active sellers are available, the buyer will get a `noAvailableSellers` message in return.

To initiate the negotiation, the broker obtains the buyer's request, preferences and user functions in the `request` message. The `Request` object is then passed to the seller by a `request` message. However, before the seller starts the negotiation, he updates his market information through the master broker, using the `getMarketInfo` message, receiving the `MarketInfo` message. The broker and seller then negotiates by sending `negotiation` messages back and forth[5], until a timeout occurs or no new proposals can be made.

In a final product, a user should not have to locate the master broker each time he enters the system, as the master broker information should be stored when he registers the first time. In addition, each message should be attached with a timestamp to ensure that the messages are received in the correct order.

This prototype has been tested for the described capabilities, and we have found it to conduct a satisfying negotiation process. The test can be read in appendix G.

6.2 Utility function test

This section will deal with the utility function, and mainly the user functions used in the utility function in equation 4.10, as they govern the behaviour of the utility function.

6.2.1 User functions

The output of the functions should be limited within the vicinity of 1, as it is multiplied with the normal market price. Four or five factors are multiplied, so just slightly low outputs will result in a very low result. Here, we will try to create some recommendations for outputs of the functions f_1 through f_6.

We suggest that a function should react on its input, but only to a certain degree. We suggest that limitations are enforced within them.

If we know the maximum and minimum values of the input, x_{min} and x_{max}, where the function should vary, we can create a linear function, incorporating these extremities. The largest wanted output, o_{max}, and the

[4]Sellers with an active advertisement.

[5]`CL` here is proposals or rejections

lowest, o_{min}, can then be used to create this linear formula, as indicated in the graph in figure 6.2.

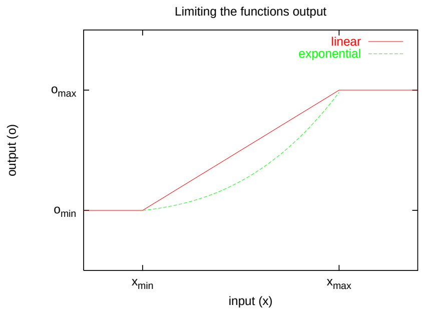

Figure 6.2: Linear or exponential

An exponential version of the graph is also displayed, just to indicate that it could be any form of formula, not just linear.

This approach can be used in almost all functions, as intervals $[x_{min}, x_{max}]$ can be identified for most.

What should be the optimal choices for o_{min} and o_{max}? We can devise a method of determining this based on the amount of impact wanted on the price. If the worst case scenario occurs, where input to all function are x_{min} or lower, all outputs would be ${o_{min}}^n$ where n is the number of user functions in the utility function ($n = 4$ for the seller and $n = 5$ for the buyer).

So if the seller would at most halve the price, he would use:

$$\frac{1}{2} = {o_{min}}^4 \quad \Leftrightarrow \quad o_{min} \simeq 0.84 \tag{6.1}$$

And equivalently, if he would at most double the price, he would use $o_{max} \simeq 1.19$. Same method can be applied to the buyer, naturally. Many recommendations for the buyer user functions can be found in section 4.4.7.

If we consider the sellers utility function, recall that is was defined like this:

$$U_{s,r} \quad = \quad f_1(d_r, s_r) \cdot f_2(t_{min}, t) \cdot f_3(D_{s,r}, l_r) \cdot f_4(R) \tag{6.2}$$

We can use the linear function approach in function f_1 through f_3. f_1, regarding demand and supply, would look at the ratio between the demand and supply, and if that ratio exceeds x_{min} or x_{max}, f_1 would return the appropriate limit values.

With f_2, regarding trust, we can use the defined hard and soft trust limits. The function can return 1 when the trust exceeds the soft limit, but between the soft and the hard limit, the linear function approach can be used. This model doesn't reward the buyer for having a great trust level, though.

f_3, regarding start date, can best be expressed by a graph, illustrated in figure 6.3. This figure shows how the choice of a start date influences the price. If the start date is close, a 20% discount is awarded (factor 0.8)[6], as the resources at this date are not likely to be sold anyway. No discount is awarded the next 13 days, but then a 10% discount is introduced, to acknowledge that the buyer books well-ahead.

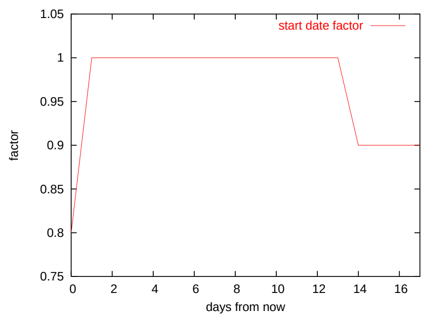

Figure 6.3: Start date function suggestion

Of course, if the buyer requires, for instance, 20 days of computation, awarding him 20% for the whole period might seem to go overboard. One give him the discount on the first 24 hours, and then no discount. And for that matter, the 10% discount after 14 days, if that is appropriate. This must of course manifest itself in an aggregate price, as the discount cannot be given directly, given our system model.

We suggest that if the payment method, R, should affect the price, it should be done by predefined values. The user can set his preferences to, for instance, award 10% discount for *fixed total currency*, 10% penalty to the price for choosing a *variable* payment approach, and ∞ for payment in *resources*, indicating that he is not willing to receive any resources as payment.

The utility function may be altered freely, and factors may be added or

[6]Or even lower. In this period, the seller would choose to accept losses, as his services would otherwise just be wasted.

removed as desired. This is just our proposal for such a function, with the variables (functions) for the user to decide.

6.2.2 Utility function tests

Here, we will try to test the utility function as a whole, given the user functions above. We will in these tests use the same request for 20GB DISK resources in a period of 48 hours. Also, we use these values as premises for the tests:

1. $x_{min} = o_{min} = 0.9$
2. $x_{max} = o_{max} = 1.1$
3. Initial trust values of 30 for both seller and buyer
4. A supply of 100 and a demand of 85
5. The buyer has set an importance of 10
6. The market price starts at 3 kr./GB/h
7. Both buyer and seller have a soft trust limit at 25 and a hard limit at 15

We used the implementation for these tests, initialising the objects we needed to calculate the price and use the utility function, without the agent aspect involved.

The buyers price setting is used, and we will do 20 iterations, recalculating the market price for every iteration. This is to get an idea of how the different factors influence the buyer's willingness to pay more (or less).

We have chosen six parameters to adjust, which are:

- The start date, starting at the preferred date
- The demand, starting at 85
- The buyer's trust and the seller's trust
- Adding the seller to the preferred list
- Importance of the deal

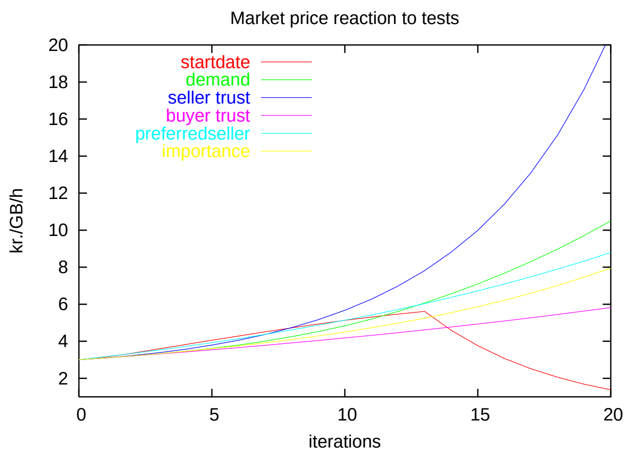

Figure 6.4: Utility function parameters

These parameters are adjusted from a value normally yielding the o_{min} result in the utility function, to the o_{max} value. This test just indicates how quickly the factor can change the utility. The results can be seen in figure 6.4.

The parameter buyer trust does not have any effect on the buyer's utility function, and this line should then be considered the bottom boundary, as it doesn't affect the price. The market price rises due to the buyer evaluating the deal to be more favourable than the market price.

The seller trust is the upper boundary, as his trust is always above the soft trust limit, thus always yielding a high output from the user function.

We can see that importance has a lower impact than the demand for instance, as a result of the user functions. The preferred sellers list is not exponential, but linear, again because the seller is added to the list throughout all iterations.

The most interesting curve to notice here, is when the start date is varied. Suddenly at the 13^{th} iteration, the buyer sets his price to 0. This is because his deadline is surpassed, and he wants no part of the deal. This would by all probability be caught by the seller's reservation price, and thus never go through as a deal and affect the market price.

6.3 Market prices

Here, we will try to illustrate how the suggested market price updating formula in section 4.4.2 updates the market price, and how the method we didn't choose handles the task.

We have fabricated an environment with 30 deals, fluctuating freely. We have then used both the chosen forecasting procedure with exponential smoothing, and the one without. The window is set to 10 deals in the past. The result can be seen in figure 6.5.

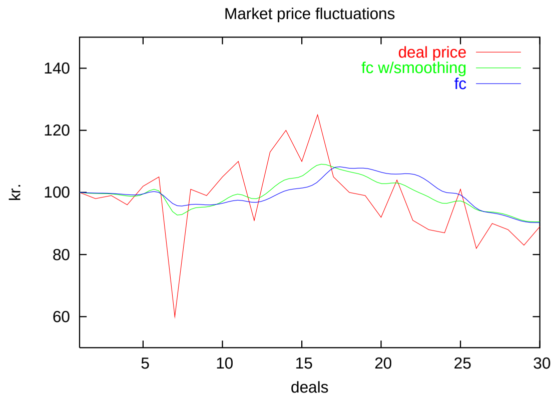

Figure 6.5: Market prices

We can see that both methods update the market price efficiently, not reacting too much to the irregularity at the 7^{th} deal. The smoothing formula seems to be more reactive, as expected. If we try to calculate the difference from the actual data for this period, we find that the exponentially smoothed method is closer to the actual data.

6.4 Summary

We have presented an overview of the implementation, which were written using Java, Aglets and JDOM technologies. A communication protocol has been devised, making it possible to negotiate properly.

Recommendations regarding user functions has been made. The user is advised to limit his user functions so they wont impact the utility function excessively, and we have shown a method for ensuring this. Then we have tested the proposed user functions and utility functions given some prerequisites, and shown how the different factors can impact the prices.

A test of the chosen method of updating the market price has also been made, which illustrated that the method is responsive and more dynamic than the method we didn't choose.

Chapter 7

Conclusion

This section will provide the conclusions to our work done in this thesis.

An exchange of services in Grid context involves many areas, so effort has been put in many different subjects.

We have devised a system model consisting of three types of players, the buyer, grid service provider (GSP) and the marketplace. The GSP are the seller of services in the system, we have denoted as WEGAS. Each of these players has one or several agents to assists them in different tasks.

WEGAS comprises of a hierarchy of marketplaces that binds the users of the system. The marketplace is the centre of commerce, to which the buyer and GSP are connected. They utilise the marketplace to negotiate deals in a safe and trusted environment.

The marketplace harbours the broker agent, who has the responsibility of performing the negotiation in cooperation with the GSP and buyer.

The negotiation are performed according to a protocol, which is based on argumentative negotiation techniques. The players sends proposals and rejections to each other, to arrive at an agreement.

These messages are described in a language we have devised, which formalises the contents of what can be considered in such a negotiation. They can express resources, payment and reasons for rejecting proposals. The language properly formatted, so messages made with it can be parsed correctly, and responded.

The resources are priced using utility functions, and using helpful tools such as a market prices, demand and supply, which the marketplace provides. When reaching an agreement, an electronic contract containing specific elements, are signed by the participants in the negotiation.

Supply is governed by advertisements, which contains the available resources at the different GSPs. The GSPs use prediction and their schedule to determine their available resources.

Vulnerabilities in the system has been explored, and a trust model has been made. The trust model supports the negotiation model in penalising bad behaviour, and rewarding good.

A small prototype has been implemented, incorporating the core negotiation features. This prototype can show how the negotiation proceeds in WEGAS.

We believe that the idea has great potential, and given more time and effort, it can evolve into a widely used concept, which can ultimately change the way we view computer services.

The first steps has been made towards a system that can connect Grids locally, and globally, so they can exchange their resources with each other, thus creating a global community pool of resources to the benefit of all.

7.1 Future work

WEGAS has great potential, but much work remains before it can be considered for any commercial purposes. We would like to end this report by discussing possible improvements.

The Grid aspect has been addressed, but not implemented. This is of course of great importance, as they are the building blocks of the system. Having WEGAS deployed on numerous grids is the only way to truly analyse how the market will evolve and how market prices for the different resources will be set.

Our assumptions also have to be met, such as determining the resource usage of a given task, before going into the negotiation about the task.

A simple scheduler is introduced to help the GSP organise tasks. To better optimise the usage of a grid, a more advanced scheduler is an absolute must. It must be able to move tasks around more dynamically. One could imagine that such a scheduler would be able prioritise tasks. Giving tasks higher importance to others, would allow the GSPs to prioritise their customers, which can be very valuable for creating a stable clientele.

An important issue is standardising tasks, which have to be developed or adopted. This would make it possible to deploy the tasks multiple types of platforms and/or Grid middle-ware.

The exchange of currency is not addressed in this thesis, and would be a dire necessity for a final product. The payment should be handled by a third party, like the marketplace since it is trusted by all parties.

Appendix A

Conflict scenarios

This appendix describes the conflict scenarios we have imagined. Table A.1 contains a column named **Conflict Scenario**, where the scenario is outlined. The next column, **Action**, describes what appropriate action should be taken. **TS** is the Transaction Scenarios that can be any of the following:

1. Buyer delivers a full package to the seller. The results are delivered when the task is finished.
2. Buyer delivers a full package to the seller. The results are delivered gradually, as they are computed.
3. Buyer delivers executable code, and data needs to be gradually fetched from the buyer system, or some other data source.
4. The buyer acquires a service from the seller, and uses it without any interaction from the seller.

Table A.1: Conflict scenario table

Nr	Conflict Scenario	TS	Action
1	Seller collects the advance payment, and vanishes.	1,2,3,4	The system should not handle this, external help is needed to resolve the conflict.
			Continued on next page

Table A.1 – continued from previous page

Nr	Conflict Scenario	TS	Action
2	Buyer never pays for the delivered services.	1,2,3,4	The system should not handle this, external help is needed to resolve the conflict, as defined in the contract.
3	Buyer vanishes, there are no one to deliver results to.	1,2,3	If the buyer does not reappear inside some time frame, the seller have to delete the data.
4	Seller breaks down, data is lost. Task must be restarted.	1,2,3,4	The buyer is contacted and asked to resend the task. The seller has to compensate the buyer for any delay.
5	Seller breaks down, executing time is lost. Task is delayed.	1,2,3,4	If the seller can not make up for the lost time within the deadline, the buyer is contacted and is compensated for lost time.
6	Seller breaches contract by not providing the agreed services.	1,2,3,4	The buyer (is compensated and) can decide if the transaction should continue or not.
7	Buyer breaches contract by damaging part of the sellers systems by a malicious program or agent.	1,2,3,4	Cannot be handled by the system. External party has to decide the consequences, as defined in the contract.
8	Buyer or seller is not truthful about their identity.	1,2,3,4	Cannot be handled by the system. External party has to decide the consequences, as defined in the contract.
9	Buyer delivers a faulty program, and seller therefore cannot deliver the services.	1,2,3,4	The seller has to be compensated for lost time, and the seller can decide if the transaction should continue or not.

Continued on next page

Table A.1 – concluded from previous page

Nr	Conflict Scenario	TS	Action
10	Seller has a badly setup system, on which the buyer program can't execute properly.	1,2,3,4	The buyer has to be compensated for lost time, and the buyer can decide if the transaction should continue or not.
11	Seller disappears.	1,2,3,4	The buyer have to find another provider, and the buyer has to be compensated for lost time.
12	Seller steals the buyer's results, without the buyer noticing it (at first).	1,2,3,4	There have to be instituted legal proceedings against the seller, according to the contract, and the buyer has to be compensated.
13	The buyer is temporarily down and the seller can't deliver results.	2	The results are kept and packaged with the next gradual result delivery.
14	The buyer system data needed is for some reason not available, and the seller can not perform the task as agreed.	3	The seller checks at intervals, if the data is available. If the booking period is exceeded, the transaction is considered done, and resources are freed.
15	The buyer is still using the resources at deadline.	4	It is the buyers responsibility to alert the seller that he needs more time to finish his task and compensate the seller for any overtime. Otherwise, the seller considers the task done, and the resources are freed.

Appendix B

Reason scenarios

Table B.1 lists all those reasons an agent can give during a negotiation. The first column describes what reason element the scenario belongs to. The second column explains the scenario in words and the last column illustrates the notation for the scenario in R.

Table B.1: Reason scenarios

Reason	**Scenario**	R
Resource	I cannot provide the requested services.	$reason(resourceDescription, cannotProvide)$
	I require a minimum of x units of this resource.	$reason(resourceDescription, minRequired(x))$
	I want at most x units of this resource.	$reason(resourceDescription, maxWanted(x))$
	I am not interested in that resource.	$reason(resourceDescription, notDesirable)$
	The posted start date is too soon, earliest acceptable date is d.	$reason(resource, tooSoon(d))$

Continued on next page

Table B.1 – concluded from previous page

Reason	Scenario	R
	The posted start date is too late, latest acceptable date is d.	$reason(resourceDescription, tooLate(d))$
	The date is not negotiable.	$reason(resourceDescription, notNegotiable)$
Constraint	Your trust level is to low.	$reason(trust, tooLow)$.
	Your trust level is to high.	$reason(trust, tooHigh)$
	I require at least the value x in the tx trust element.	$reason(trust, required(x, tx))$
	p percent should be paid in advance.	$reason(payment, wanted(p))$
Price	The price is to high.	$reason(price, tooHigh)$
	The price is too low.	$reason(price, tooLow)$
	Another participant gave me a better price, o, for the same deal.	$reason(price, betterOffer(o))$
	I have been doing business with you for a long time, I believe that I am entitled to a discount of x percent.	$reason(price, regular(x))$

Appendix C

BNF

Below is an description of the BNF to the logical language used in the negotiation.

$$
\begin{aligned}
<CL> &::= <proposal> | <request> | <widthdraw> | \\
&\quad\ <accept> | <reject> \\
<proposal> &::= \textbf{proposal(} <parties> \textbf{,} <L> \textbf{,} <time> \textbf{)} \\
<request> &::= \textbf{request(} <parties> \textbf{,} <L> \textbf{,} <time> \textbf{)} \\
<withdraw> &::= \textbf{withdraw(} <parties> \textbf{,} <time> \textbf{)} \\
<accept> &::= \textbf{accept(} <parties> \textbf{,} <L> \textbf{,} <time> \textbf{)} \\
<reject> &::= \textbf{reject(} <parties> \textbf{,} <L> \textbf{,} <reasons> \textbf{,} <time> \textbf{)} \\
\\
<parties> &::= (<agent> \textbf{,} <agent>) \\
<agent> &::= \textbf{a} \mid \textbf{b} \\
<L> &::= \boldsymbol{\{}[<price> \textbf{,}] <constraints> \textbf{,} <resources> \boldsymbol{\}} \\
<resources> &::= <and> | <or> | <resource> \\
<reasons> &::= \boldsymbol{\{} <reason> \{, <reason>\}\boldsymbol{\}} \\
<reason> &::= \textbf{reason(} <R> \textbf{)} \\
<and> &::= \textbf{and(} <resources> \textbf{,} <resources> \textbf{)} \\
<or> &::= \textbf{or(} <resources> \textbf{,} <resources> \textbf{)} \\
<resource> &::= <RAM> \ | <CPU> | <NET> | <DISK> | <APP> \\
<time> &::= Timestamp
\end{aligned}
$$

$$
\begin{array}{rcl}
< RAM > & ::= & \mathbf{RAM(}\ < variables >\ ) \\
< CPU > & ::= & \mathbf{CPU(}\ < variables >\ ) \\
< NET > & ::= & \mathbf{NET(}\ < variables >\ ) \\
< DISK > & ::= & \mathbf{DISK(}\ < variables >\ ) \\
< APP > & ::= & \mathbf{APP(}\ < appname >\ \mathbf{,}\ < variables >\ ) \\
 & & \\
< variables > & ::= & \mathbf{\{}\ < variable > \{, < variable >\}\mathbf{\}} \\
< variable > & ::= & \mathbf{equal(}\ < var >\ ) \\
< appname > & ::= & \mathbf{equal(}\ < application >\ ) \\
< constraints > & ::= & \mathbf{\{}\ < constraint > \{, < constraint >\}\mathbf{\}} \\
< constraint > & ::= & \mathbf{equal(}\ < constraintval >\ ) \\
< constraintval > & ::= & < advance > \\
< price > & ::= & \mathbf{equal(price,}\ < priceval >\ ) \\
< priceval > & ::= & < resources > \mid Integer \\
 & & \\
< var > & ::= & < startDate > \mid < length > \mid < size > \mid < indPrice > \\
< startdate > & ::= & \mathbf{startDate,}\ Date \\
< length > & ::= & \mathbf{length,}\ Date \\
< advance > & ::= & \mathbf{advance,}\ Percentage \\
< size > & ::= & \mathbf{size,}\ Integer \\
< application > & ::= & \mathbf{appname,}\ String \\
< trustlevel > & ::= & \mathbf{trustlevel,}\ Integer \\
< indPrice > & ::= & \mathbf{indPrice,}\ < type > \\
< type > & ::= & \mathbf{fixed}(Real) \mid \mathbf{variable}(Real, Real) \\
 & & \\
< R > & ::= & < resourcereason > \mid < constraintreason > \\
 & & \mid < paymentreason > \\
< resourcereason > & ::= & < resource >\ \mathbf{,}\ < rproblem > \\
< constraintreason > & ::= & < tproblem > \mid < advproblem > \\
< paymentreason > & ::= & < paymentproblem >
\end{array}
$$

$$
\begin{array}{rcl}
<\mathit{rproblem}> & ::= & \textbf{cannotProvide} \mid \textbf{minRequirement(} <q> \textbf{)} \\
 & & \mid \textbf{maxWanted(} <q> \textbf{)} \mid \textbf{notDesirable} \\
 & & \mid \textbf{tooSoon(} <\mathit{startdate}> \textbf{)} \mid \textbf{tooLate(} <\mathit{startdate}> \textbf{)} \\
 & & \mid \textbf{badDate(}\mathit{Date}\textbf{)} \mid \ \textbf{notNegotiable} \\
<\mathit{tproblem}> & ::= & \textbf{trust,} <\mathit{trustarg}> \\
<\mathit{advproblem}> & ::= & \textbf{advance, wanted(}\mathit{Integer}\textbf{)} \\
<\mathit{paymentproblem}> & ::= & \textbf{payment,} <\mathit{paymentarg}> \\
<\mathit{trustarg}> & ::= & \textbf{tooLow} \mid \textbf{tooHigh} \mid \textbf{required(}\mathit{Integer}, <\mathit{trustelement}> \textbf{)} \\
<\mathit{paymentarg}> & ::= & \textbf{tooLow} \mid \textbf{tooHigh} \mid \textbf{betterOffer(} <\mathit{offer}> \textbf{)} \\
 & & \mid \textbf{regular(} <\mathit{percent}> \textbf{)} \mid <\mathit{resourcereason}> \\
 & & \mid \textbf{no(} <\mathit{paymentmethod}> \textbf{)} \\
<\mathit{paymentmethod}> & ::= & \textbf{fixed} \mid \textbf{variable} \mid \textbf{individual} \\
 & & \mid \textbf{total} \mid \textbf{currency} \mid \textbf{resources}
\end{array}
$$

$$
\begin{array}{rcl}
<q> & ::= & \mathit{Integer} \\
<\mathit{percent}> & ::= & \mathit{Integer} \\
<\mathit{offer}> & ::= & \mathit{Integer} \\
<\mathit{trustelement}> & ::= & \textbf{trustid}
\end{array}
$$

Appendix D

Negotiation example

This example illustrates a typical negotiation, where the communication between the parties are shown, in the first round where the seller sends proposals and the buyer (broker) sends rejections.

In the examples, the $variable = value$ notation is used in place of $equal(variable, value)$.

The broker sends a request to a seller asking for DISK, along with a suggestion to a date period, and how much he is willing to pay in advance. The request tree is seen in D.1.

$$\begin{aligned} n_1 ::= request(\quad & (broker, seller), \{\{advance = 15\%\}, \\ & DISK(size = 10, startDate = 04.12.01\ 06{:}00, \\ & length = 192h\}, t_1) \end{aligned}$$

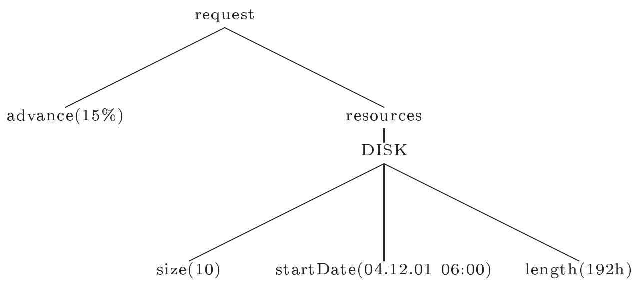

Figure D.1: The request, n_1

The seller parses the request and creates a proposal tree, seen in figure D.2. For this, he uses his preferences, and sees that he has a more preferable start date for him. He also picks the first element in his payment method list, which is variable pricing, and puts it in the proposal. Also, the advance is adjusted.

$$
\begin{aligned}
n_2 ::= proposal(\quad & (seller, broker), \{\{advance = 25\%\}, DISK(size = 10, \\
& startDate = 04.12.11\ 06{:}00, length = 192h, \\
& indPrice = variable(110.0))\}, t_2)
\end{aligned}
$$

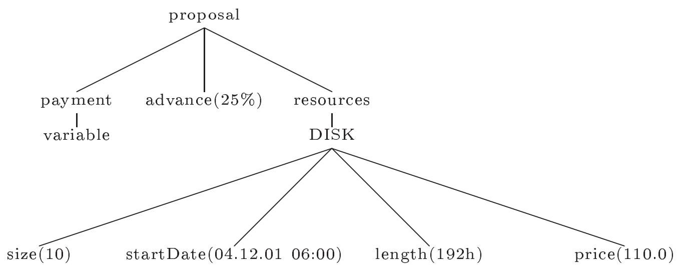

Figure D.2: Proposal, n_2

The number given in the *price* element, is the percentage of the market price wanted, in this example he wants 110% of the market price. The new start date is not acceptable for the broker on behalf of the buyer, therefore the broker sends a rejection stating that this is to late, as seen in figure D.3. Also, the advance element is rejected, and a new value proposed.

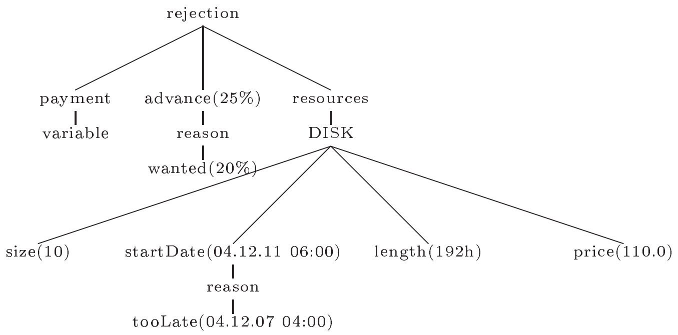

Figure D.3: Rejection, n_3

$$
\begin{aligned}
n_3 ::= reject(\quad & (broker, seller), n_2, \{reason(DISK(size = 10, \\
& \ldots, indPrice = variable(110.0)), \\
& tooLate(startDate, 04.12.11\ 06{:}00)), \\
& reason(advance, wanted(20\%))\}, t_3)
\end{aligned}
$$

The advance value given in the *wanted* node, is the new suggestion of an advance value for the seller to consider. The date value in the *tooLate* node indicates where the buyer at the very *latest* can fit the task in. This value is remembered by the seller, so he wont suggest dates later than that in the future proposals.

The payment method (*variable*) is also acceptable for the buyer, as he doesn't complain on it. Also, 110% of the current market price is below the buyers reservation price, so he didn't reject on that either.

The seller parses the rejection and creates a new proposal, n_4. He accepts the proposal of 20% in advance payment, and he moves the start date to within the buyers limits.

$$\begin{aligned} n_4 ::= proposal(\quad & (seller, broker), \{\{advance = 20\%\}, DISK(size = 10, \\ & startDate = 04.12.06\ 20{:}00, length = 192h, \\ & indPrice = variable(110.0))\}, t_4) \end{aligned}$$

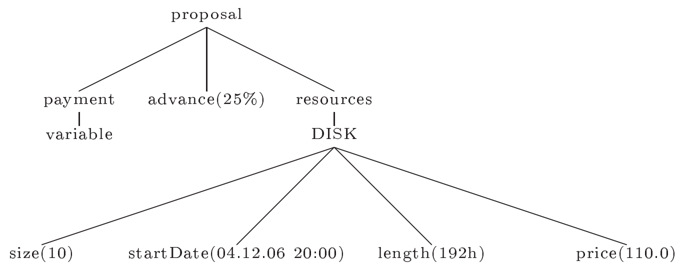

Figure D.4: Proposal, n_4

As this proposal is found acceptable, the broker saves this proposal for later discussion, namely the round where the price is further negotiated.

Appendix E

Security analysis

Our analysis is based on the nine step model suggested by [30]. The model is adopted since it ensures a complete analysis of the system and is recommended from the National Institute of Standards and Technology. The model includes the following steps:

1. System Characterisation
2. Threat Identification
3. Vulnerability Identification
4. Control Analysis
5. Likelihood Determination
6. Impact Analysis
7. Risk Determination
8. Control Recommendations
9. Result Documentation

Step 4, 8 and 9 of the model refer to areas that is out of scope for this thesis and therefore not considered.

E.1 System characterisation

To be able to make a risk assessment of WEGAS, it is necessary to understand the system process environment. Most of the characteristics of the system as been mentioned earlier, and will not be repeated, unless they are important in that context. As mentioned, the actors in the system are brokers, marketplaces, sellers and buyers. The actors can be considered, from a technical point of view, as having one of two different roles, agent or host.

The seller and buyer agents posses sensitive information, like reservation price, constraints, etc., which is important that they keep for themselves. The marketplace is a central part of the system, since it is here that all information is exchanged. Therefore it is important that the marketplace is trusted and have a robust security mechanism. Also, the marketplace has access to data stores that hold information of the system's users. It should be guaranteed that marketplaces are the only access points to this data store.

The agents in the system do not necessarily know each other and do not need to know each other either. Yet, all agents must be approved by the marketplace to be able to use its facilities. In addition, the agents must be able to authenticate that the marketplace is who or what it states to be. All authentication in the system is done by using an authentication server. The agents obtain their identification before entering the registration phase of the marketplace, see section 2.6.2, and is therefore considered to be out of scope for this analysis.

Since an agent cannot harbour at a marketplace until the authentication and registration process is performed, the availability of the system is dependent on the authentication servers to be accessible. Likewise, if the marketplace itself is not obtainable, no negotiation can be performed. Although some mechanism for delegating the load to another nearby marketplace is very possible to create due to the hierarchical nature of WEGAS.

E.2 Threat identification

Threat identification consists of analysing the source of the threat and the motivation of that source. This consideration is of importance when determining the likelihood of a certain threat, later on in the analysis.

There are three types of threat sources, natural[1], human[2] and environmental[3] [30]. For obvious reasons, only human threats are considered.

Even if the agents to some extent are intelligent, they cannot obtain malicious intentions on their own and are therefore treated as human threats. As mentioned, you can narrow the users down to only two types of users, agents and hosts. Aside from that, external threats exist. Table E.1 illustrates a all the source threats we have identified and their motivation.

[1]Floods, earthquakes, etc.
[2]events either enabled or caused by human beings
[3]long-term power failure, pollution etc.

Table E.1: Threat source analysis

Threat source	**Motivation**	**Threat actions**
Hacker	Challenge, Ego	Hacking, Social Engineering, System intrusion and unauthorised system access.
Competitors	Competitive advantages	Information destruction, information theft, unauthorised system access and social engineering.
Insiders(Sellers, buyers, marketplaces and program errors)	Negotiation advantages, Sabotage	Lack of error handling, system bugs, system sabotage, malicious code, fraud and theft.
Computer criminals	Destruction of information, influence outcome of deal, Monetary	Sale of information, sabotage, spoofing, system intrusion, system tampering.

E.3 Vulnerability identification

The purpose of this part is to produce a list of vulnerabilities that could be exploited by the identified threat sources. A vulnerability is considered to be a flaw or weakness that exist in the system.

The vulnerabilities identified are as general as possible in contrast to making them more specific, since we don't have an implemented and running system to explore and test on. The identified vulnerabilities is listed in table E.2.

Table E.2: Vulnerabilities

Nr	Vulnerability	Threat Source	Threat Action
1	Marketplace has a badly setup firewall	Hackers, industrial espionage, computer criminals	Breach of the firewall and access to confidential and sensitive data.
2	Gullible system administrator	Hackers, industrial espionage, computer criminals	Social engineering
3	Execution of malicious code	Insiders	After being accepted into the marketplace the agent executes code with malicious intent.
4	Trying to repeat an action, or mess things up, by re-sending a message	Insiders	An agent sends an old message, trying to obtain the same effect as the old message did.
5	An old agent is sent to the system again	Hackers, computer criminals	An agent is compromised and resent at a later occasion.
6	An agent tries to act on the behalf of someone else	Insiders	An agent impersonates another user for personal gain.
7	Changing the behaviour of an agent or analyse its intentions	Hackers, computer criminals	Capture and modify or analyse the code of an agent.

Continued on next page

Table E.2 – concluded from previous page

Nr	Vulnerability	Threat Source	Threat Action
8	A system obtains a real marketplace address	Hackers, computer criminals	A false marketplace obtains the address of a real marketplace and thereby disables it.
9	Task data is manipulated by a third party	Hackers, industrial espionage, computer criminals	Task data is obtained on its way to or from the seller and the information is tampered with, before it is sent to the proper recipient.
10	Communication between the mobile agents and their home agent is compromised	Hackers, industrial espionage, computer criminals	A man in the middle attack, the information between the agents is captured and tampered with, before sent back to the proper recipient.
11	The communication between the CA and the agent or marketplace is modified	Hackers, industrial espionage, computer criminals	Someone could play a false CA, by intercepting communication, and send false answers on the CA's behalf.
12	There are errors in the code	All threat sources	The users take advantage of an error, or security hole, in WEGAS to obtain unauthorised information.
13	A user claims that he has a contract with another user, but the other user denies	Insiders	A user creates a false contract, or the other user is lying about his participation.

E.4 Likelihood analysis

To determine the likelihood of a certain vulnerability is very difficult, as we have no statistical data to analyse. However, to be able to do a complete security analysis, we will try to determine the likelihood of the vulnerabilities based on the following factors:

- Threat source motivation
- Nature of the vulnerability
- Existence and effectiveness of available controls

The likelihood of an vulnerability is divided into three different levels, low, medium and high. The definitions, adopted from [30], of the different likelihood levels can be seen below.

High The threat-source is highly motivated and sufficiently capable, and controls to prevent the vulnerability being exercised are ineffective.

Medium The threat-source is motivated and capable, but controls are in place that may impede successful exercise of the vulnerability.

Low The threat-source lacks motivation or capability, or controls are in place to prevent, or at least significantly impede, the vulnerability from being exercised.

By controls we refer to those methods that exist to prevent a certain vulnerability. The likelihood for each vulnerability is shown in table E.3 below.

E.5 Impact analysis

Next step in the security analysis is to determine the impact of the different security threats. The impact of a certain security breach is a deciding element in how critical the information asset is for the company, which in return is decided based on the well-known six security factors, integrity, availability, authentication, nonrepudiation, access control and confidentiality [29]. Moreover, a few security breaches can be measured in lost income and repair costs. Impact of bad publicity, reputation and similar factors is harder to measure, although often more damaging. Therefore, we use the three level guideline as before, adopted from [30], to determine the impact where all above factors is taken into consideration.

High The vulnerability may: result in highly costly loss of major tangible assets of resources, may significantly violate, harm or impede an organisations mission, reputation or interest.

Medium The vulnerability may: result in costly loss of major tangible assets of resources, may violate, harm or impede an organisations mission, reputation or interest.

Low The vulnerability may: result in highly costly loss of some tangible assets of resources, may noticeably violate, harm or impede an organisations mission, reputation or interest.

The impact of each vulnerability is listed in table E.3. In addition to the likelihood analysis, it is difficult to determine the impact of a vulnerability, since the impact can vary over time.

Table E.3: Likelihood - Impact - Risk Level overview

Nr	Likelihood	Impact	Risk Level
1	High	High	High
2	High	High	High
3	Medium	High	Medium
4	Low	Medium	Low
5	Low	Medium	Low
6	Medium	Medium	Medium
7	Medium	High	Medium
8	Low	Medium	Low
9	High	Medium	Medium
10	Medium	Medium	Medium
11	Medium	High	Medium
12	High	Medium	Medium
13	Medium	Low	Low

E.6 Risk determination

The last step in our down-scaled version of the model from [30], is the risk determination that summarises the earlier steps. Risk determination is calculated by multiplying the likelihood of a threat and the impact of the threat. However, since both of these factors are given by either high, medium or low, and they do not have the same weight in the computation, there is need for some conversion. We have adopted the conversion suggested by [30].

- The probability assigned for each threat likelihood level is 1.0 for High, 0.5 for Medium and 0.1 for low
- Impact level is assigned with the probability 100 for High, 50 for Medium and 10 for Low.

The result of the risk level computation can be seen in table E.3. We have rated the number as high, medium or low risk. The risk scale provides a good help for management, when they consider new security strategies and what actions that must be taken to strengthen the security.

Appendix F

Class diagrams

Here follows the class diagrams for the 7 packages of the implementation described in section 6.1.

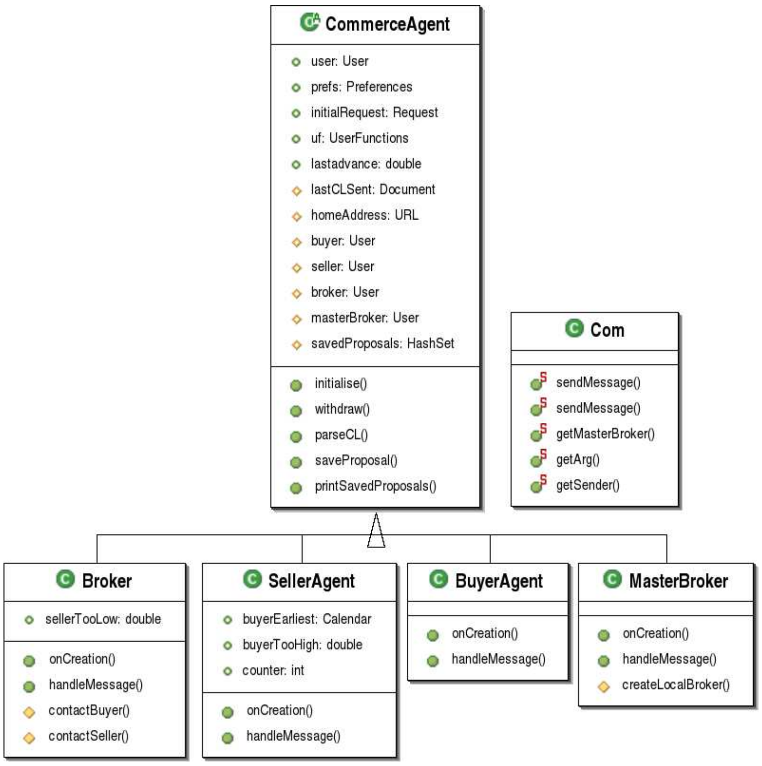

Figure F.1: agent package

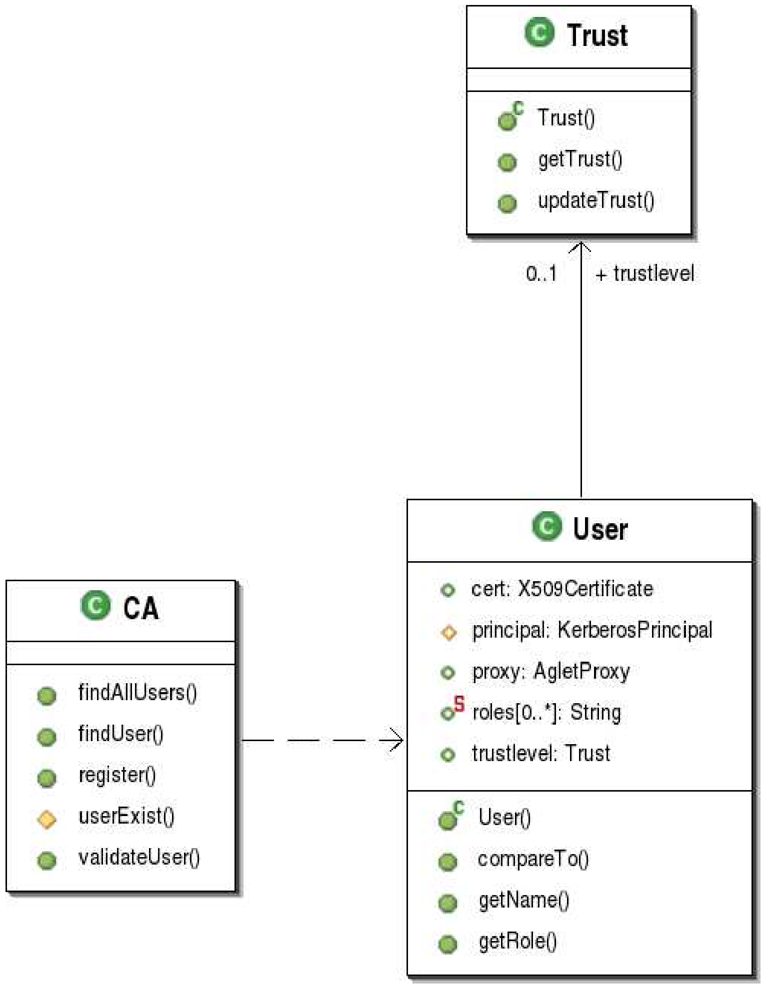

Figure F.2: ca package

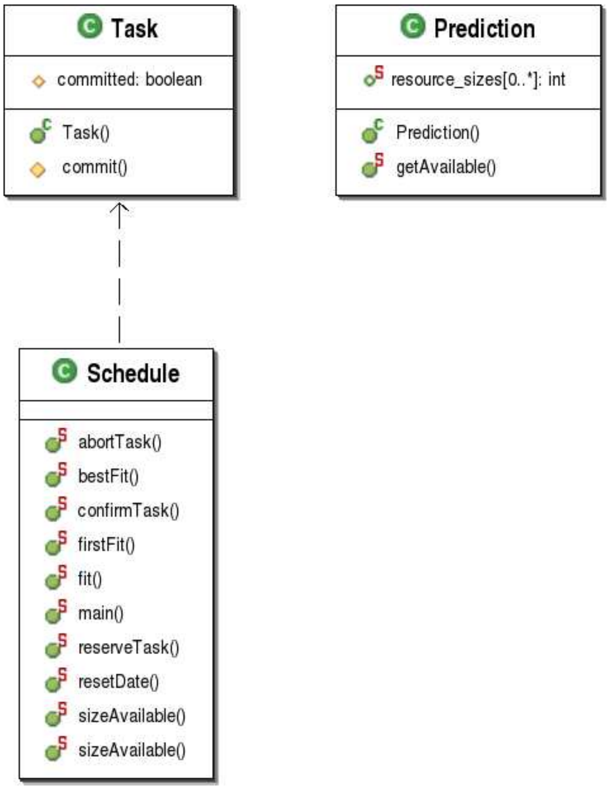

Figure F.3: gsp package

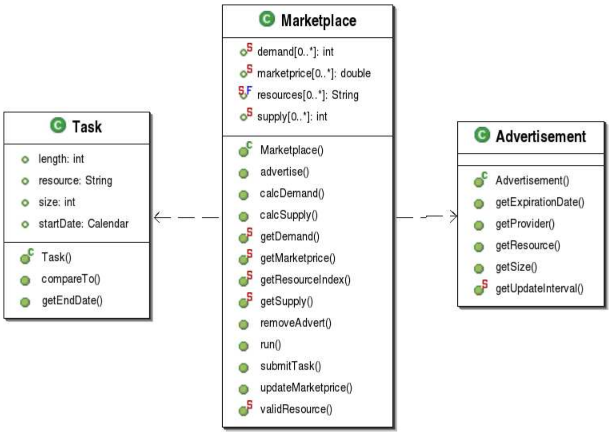

Figure F.4: marketplace package

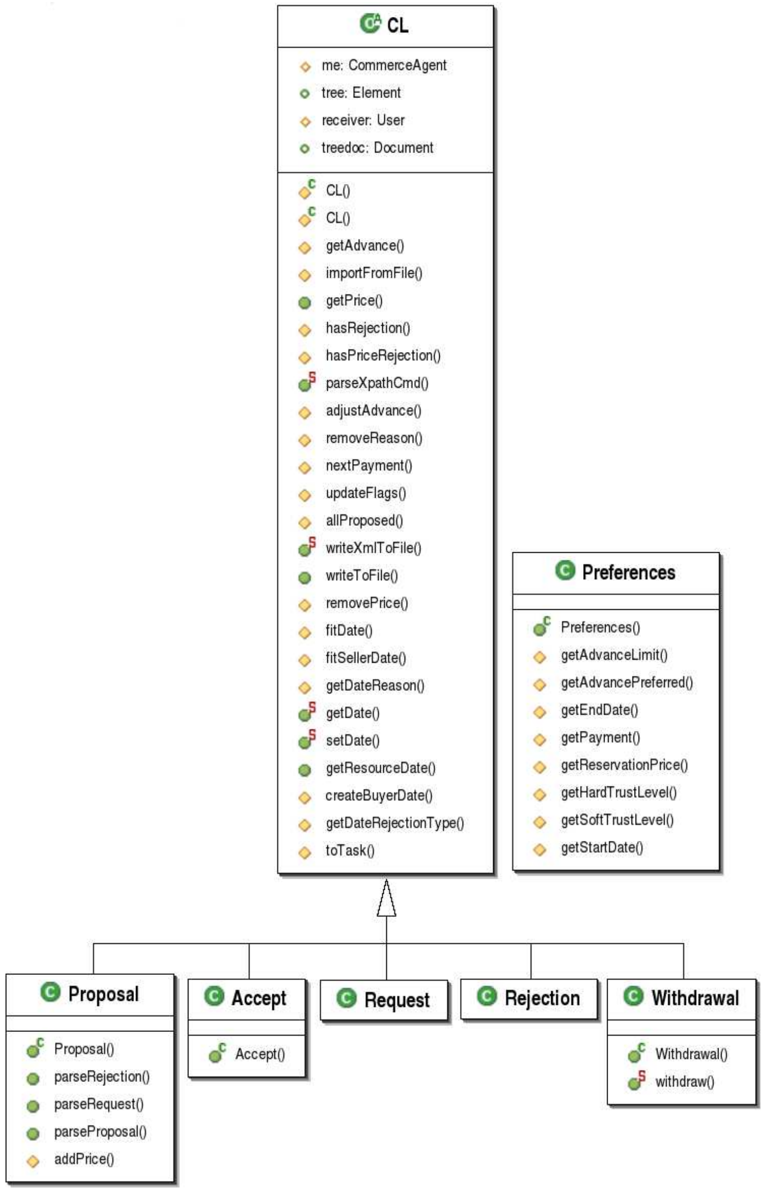

Figure F.5: negotiation package

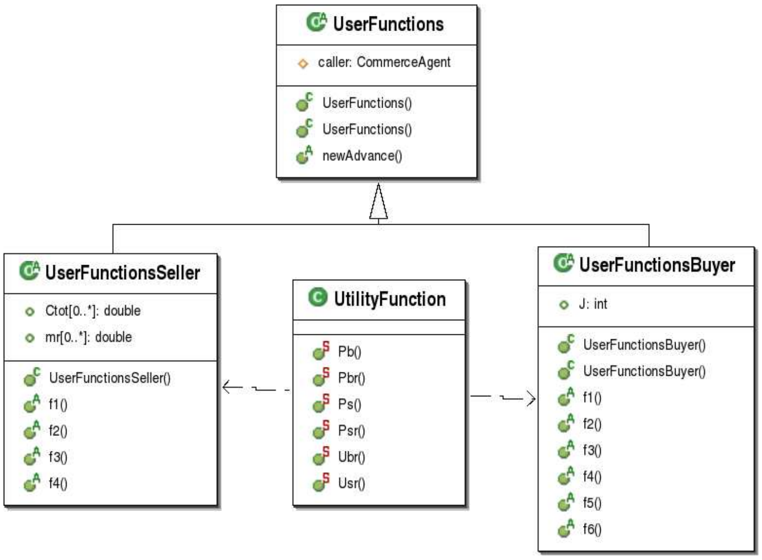

Figure F.6: negotation.utility package

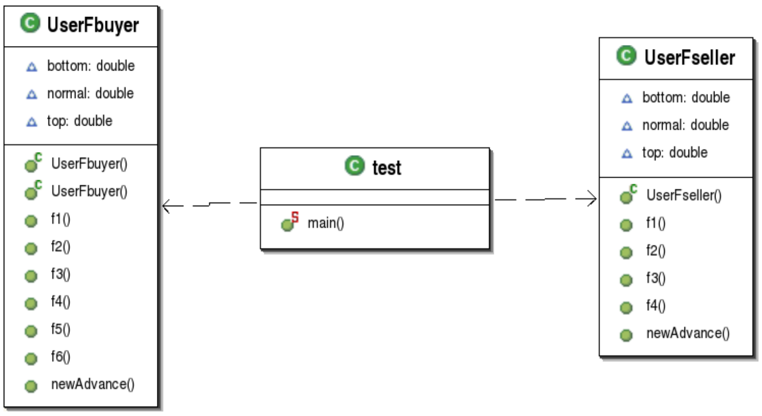

Figure F.7: test package

Appendix G

Prototype test

Even though the prototype does not include all the features in WEGAS, we believe that testing it will support some of the presented ideas. It will also show that the agents can perform the negotiation and that the suggested functions are competent to evaluate a proposal and make the correct decisions. We have created two test cases, one where the buyer and seller will approve of several possible proposals and one where one proposal will be approved. In both test cases, the user functions suggested in section 6.2.1 are used.

The preferences for the buyer, respectively the seller, is shown below.

Buyer

Advance:	`<preferred,limit> = <15%,50%>`

Payment:	`[(fixed,individual,null,0.9),`
	`(variable,null,null,1.1),`
	`(fixed,total,currency,3.0)]`

Dates:	`2004.3.15 12:00 - 2004.3.30 12:00`
Reservation price:	2000
Min trust levels:	`<hard,soft> = <20,30>`

Seller

Advance:	`<preferred,limit> = <80%,30%>`

Payment:	`[(variable,null,null,1.5),`
	`(fixed,individual,null,0.8),`
	`(fixed,total,currency,3.0)]`

Reservation price:	1600
Min trust levels:	`<hard,soft> = <15,25>`

In both test cases, the buyer wants to buy 400 MB RAM and 2GB Disk for 4 days starting at noon on the 16th of March.

When launching the agents with the described preferences, the buyer returns to his home base with 3 accepted proposals. Analysing the negotiation data, it is rather trivial to see that the proposals are accurate. Already before the negotiation starts, we could predict that the buyer would return with three proposals. Both the start date and the advance payment are within a range acceptable for both users. Only the payment method can vary in the proposals. The seller and the buyer has three matching payment methods, so they can construct three proposals.

In the second test case, we make it harder for the users to find an solution. The seller is only willing to accept payment with currency, and has only one time slot available in his schedule where the buyer's task would fit.

As expected, the users find the only solution and no others. Compared to the last test, a greater amount of messages has to be sent, since the seller prefers other payment methods than the buyer.

The above test shows that the basic features in WEGAS works as proposed. Also, one can see that the constructed agents can handle the negotiation and facilitate the users requests with acceptable proposals.

G.1 Results

The results of the two test cases described above, is posted here.

G.1.1 Test case 1


```
<?xml version="1.0" encoding="UTF-8"?>
<savedproposals>
<proposal id="1">
  <payment flag="current">
     <fixed flag="undecided">
        <total flag="undecided">
           <currency flag="undecided" />
           <resources flag="undecided" />
        </total>
        <individual flag="proposed" />
     </fixed>
     <variable flag="current" />
  </payment>
  <advance flag="accepted">50.0</advance>
```

```
  <resources flag="undecided">
      <resource name="disk" flag="fixed">
         <size flag="fixed">2</size>
         <startdate flag="undecided">
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">16</day>
            <hour flag="undecided">12</hour>
         </startdate>
         <length flag="fixed">96</length>
      <price>1.1979</price></resource>
      <resource name="ram" flag="fixed">
         <size flag="fixed">400</size>
         <startdate flag="undecided">
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">16</day>
            <hour flag="undecided">12</hour>
         </startdate>
         <length flag="fixed">96</length>
    <price>1.1979</price>
      </resource>
  </resources>
</proposal>
<proposal id="2">
  <payment flag="current">
     <fixed flag="current">
        <total flag="undecided">
           <currency flag="undecided" />
           <resources flag="undecided" />
        </total>
        <individual flag="current" />
     </fixed>
     <variable flag="undecided" />
  </payment>
  <advance flag="accepted">50.0</advance>
  <resources flag="undecided">
      <resource name="disk" flag="fixed">
         <size flag="fixed">2</size>
         <startdate flag="undecided">
```

```
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">15</day>
            <hour flag="undecided">12</hour>
         </startdate>
         <length flag="fixed">96</length>
      <price>384.91</price></resource>
      <resource name="ram" flag="fixed">
         <size flag="fixed">400</size>
         <startdate flag="undecided">
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">15</day>
            <hour flag="undecided">12</hour>
         </startdate>
         <length flag="fixed">96</length>
     <price>1283.04</price></resource>
  </resources>
</proposal>
<proposal id="3">
  <payment flag="current">
     <fixed flag="current">
        <total flag="current">
           <currency flag="current">
        <price>1853.28</price>
      </currency>
           <resources flag="undecided" />
        </total>
        <individual flag="proposed" />
     </fixed>
     <variable flag="proposed" />
  </payment>
  <advance flag="accepted">50.0</advance>
  <resources flag="undecided">
      <resource name="disk" flag="fixed">
         <size flag="fixed">2</size>
         <startdate flag="undecided">
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">16</day>
```

```
            <hour flag="undecided">12</hour>
         </startdate>
         <length flag="fixed">96</length>
      </resource>
      <resource name="ram" flag="fixed">
         <size flag="fixed">400</size>
         <startdate flag="undecided">
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">16</day>
            <hour flag="undecided">12</hour>
         </startdate>
         <length flag="fixed">96</length>
     </resource>
  </resources>
</proposal>
</savedproposals>
```


G.1.2 Test case 2


```
<?xml version="1.0" encoding="UTF-8"?>
<savedproposals>
<proposal id="1">
  <payment flag="current">
     <fixed flag="current">
        <total flag="current">
           <currency flag="current">
        <price>1853.28</price>
      </currency>
           <resources flag="undecided" />
        </total>
        <individual flag="rejected" />
     </fixed>
     <variable flag="rejected" />
  </payment>
  <advance flag="accepted">50.0</advance>
  <resources flag="undecided">
      <resource name="disk" flag="fixed">
         <size flag="fixed">2</size>
         <startdate flag="undecided">
```

```
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">25</day>
            <hour flag="undecided">19</hour>
         </startdate>
         <length flag="fixed">96</length>
      </resource>
      <resource name="ram" flag="fixed">
         <size flag="fixed">400</size>
         <startdate flag="undecided">
            <year flag="undecided">2004</year>
            <month flag="undecided">4</month>
            <day flag="undecided">25</day>
            <hour flag="undecided">19</hour>
         </startdate>
         <length flag="fixed">96</length>
     </resource>
  </resources>
</proposal>
</savedproposals>
```

Appendix H

CD content

This appendix will give an overview of the included CD. The CD are bundled with report. If it is not, or if any updates are posted, the CD structure can be seen at `http://wegas.mufaza.dk`, where the files can also be fetched.

The CD contains the following elements:

/**report**/ This report in pdf and ps formats.

/**papers**/ Some of the papers used for this thesis.

/**javadoc**/ The implementation documentation.

/**wegas**/ The implementation files.

/**wegas**/**src**/ The implementation source files.

/**wegas**/**lib**/ The external libraries needed to run the implementation.

/**wegas**/**xml**/ The WEGAS xml files are stored here.

/**INSTALL** Installation instructions for the prototype. This file also describes how to run it, and provides an example.

/**README** A file describing the contents of the CD.

If the reader would like to experiment with the source, please refer to the src directory. Thorough installation instructions are given in the INSTALL file, so please read that file.

Bibliography

[1] Why cpu benchmarks are good and bad. http://www.theinquirer.net/?article=9465.

[2] Martin Bichler, Simon Field, and Hannes Werthner. Introduction: Theory and application of electronic market design. *Electronic Commerce Research*, 1:215–220, 2001.

[3] Rajkumar Buyya, David Abramson, and Jonathan Giddy. A case for economy grid architecture for service oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, April 2001.

[4] Adrian Li Mow Ching, Lionel Sacks, and Paul McKee. Super resource-management for grid computing. *London Communication Symposium*, September 2002.

[5] International court of arbitration, international dispute resolution services. http://www.iccwbo.org/court/english/arbitration/introduction.asp.

[6] Frequently asked questions about the processor. http://www.demandtech.com/FAQsCPU.htm#CPUFAQ3.

[7] Measuring cpu performance cross platform. http://www.aselabs.com/articles.php?articleid=18.

[8] Spec cpu2000. http://www.spec.org/cpu2000/.

[9] Rob F. Van der Wijngaart and Michael Frumkin. Nas grid benchmarks version 1.0. Technical report, NASA Advanced Supercomputing (NAS) Division, July 2002.

[10] Mark d'Inverno and Michael Luck. *Understanding Agent Systems*. Springer Series on Agent Technology. Springer-Verlag, February 2001.

[11] Giusella Finocchiaro. The conclusion of the electronic contract through "software agents" a false legal problem? brief considerations. *Computer Law and Security Report*, 19(1):20–24, 2003.

[12] Ian Foster. What is the grid? a three point checklist. *GRID Today*, 1(6), July 2002.

[13] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[14] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Application*, 15(3), 2001.

[15] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J.P. Müller, M.J. Wooldrigde, and N.R. Jennings, editors, *Intelligent Agents III - Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193, pages 21–35. Springer-Verlag, 1997.

[16] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, Inc, sixth edition, 1995.

[17] Alessio R. Lomuscio, Michael Wooldridge, and Nicholas R. Jennings. A classification scheme for negotiation in electronic commerce. *Lecture Notes in Computer Science*, 1991:19+, 2001.

[18] Peter Lynggaard. *Driftsøkonomi*. Handelshøjskolens, 2001.

[19] R. Mourtada and F. Szidarovszky. Bargaining in e-commerce: A special negotiation scheme. In *ACS/IEEE International Conference on Computer Systems and Applications*, pages 546–548. IEEE, 2001.

[20] Nordugrid, a grid research and development project in the nordic countries. `http://www.nordugrid.org`.

[21] H. Ouchiyama, T. Yamazaki, and R. Huang. An e-shopping system with different negotiation model. In S. Peng, V.V Savchenko, and S. Yukita, editors, *Proceedings 2002 International Symposium Cyber Worlds: Theory and Practices. CW2002*, pages 311–317. IEEE Comput. Soc., 2002.

[22] Carl Pacini, Christine Andrews, and William Hillison. To agree or not to agree: Legal issues in online contracting. *Business Horizons*, 45(1):43–52, 2002.

[23] Simon Parsons, Carles Sierra, and Nick Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[24] Performance management guide. `http://www16.boulder.ibm.com/pseries/en_US/aixbman/prftungd/2365c11.htm%#i19191`.

[25] Emma Regner. Telephone interview with a swedish employed lawyer on 2003-10-27.

[26] A. Sameh and D. Fakhry. Security in mobile agent systems. In *Applications and the Internet*, February 2002.

[27] Carles Sierra, Nick R. Jennings, Pablo Noriega, and Simon Parsons. A framework for argumentation-based negotiation. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Fourth International Workshop on Agent Theories Architectures and Languages (ATAL-97)*, volume 1365 of *Lecture Notes in Computer Science vol. 1365*, pages 177–192. Springer-Verlag, 1998.

[28] Stephen Smale. Dynamics in general equilibrium theory. *The American Economic Review*, 66(2), May 1976.

[29] William Stallings. *Cryptography and Network Security.* Prentice-Hall, Inc, 1999.

[30] Gary Stonebumer, Alice Goguen, and Alexix Feringa. Risk mangement guide for information technology system. Technical Report 800-30, National Institute of Strandards and Technology, 2001.

[31] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. Technical Report GWD-PERF-16-2, GGF Performance Working Group, August 2002.

[32] Brian Tierney, Brian Crowley, Dan Gunter, Mason Holding, Jason Lee, and Mary Thompson. A monitoring sensor management system for grid environments. In *HPDC*, pages 97–104, 2000.

[33] Orazio Tomarchio, Lorenzo Vita, and Antonio Puliafito. Active monitoring in grid environments using mobile agent technology. In *2nd*

Workshop on Active Middleware Services (AMS'00) at HPDC-9, August 2000.

[34] Kenneth Lolk Vester and Moses Claus Martiny. Pmp. Technical report, IMM, DTU, June 2003.

[35] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[36] Giorgos Zacharia, Theodoros Evgeniou, Alexandros Moukas, Petros Boufounos, and Pattie Maes. Economics of dynamic pricing in a reputation brokered agent mediated marketplace. *Electronic Commerce Research*, 1:85–100, 2001.