Master's Thesis Report

Intelligent Agents in an Electronic Auction Context

by

Eric Jönsson
Jens Nordberg

LiTH-IDA-Ex-00/37

2000-04-14

Master's Thesis Report

Intelligent Agents in an Electronic Auction Context

by

Eric Jönsson
Jens Nordberg

Company: Intentia R&D AB
Supervisor: Mikael Berg
Examiner: Nancy Reed

LiTH-IDA-Ex-00/37

2000-04-14

ABSTRACT

As business-to-business electronic commerce evolves, different kinds of electronic auctions are expected to become an increasingly common and important way of selling a wide range of products. Therefore, it's very interesting for a Company like Intentia, aiming to be a leading provider of business-to-business (BTB) e-commerce solutions, to investigate how electronic auctions can be integrated into their solutions, and how intelligent agents can be used for this purpose.

In this work, auction theory and artificial intelligence techniques related to electronic auction handling are discussed at length. The techniques are evaluated in order to discover if they can add user value to a system by offering support to the user during electronic auction processes. The evaluation of the methods was done in a domain consisting of steel products. The conclusion is that the standard AI techniques investigated are insufficient for handling the complexity of the test domain.

Further, a multi-agent system design for handling automatic bidding – particularly at several simultaneous auctions – in an open and distributed environment is presented. The conclusion on the bidding algorithms discussed is that it is indeed possible to predict an auction's end price with a certain precision provided that the price doesn't fluctuate too heavily over time.

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

This chapter gives a background to the thesis work. In section 1.1, the background and motivation for the thesis is briefly described. In section 1.2, the purpose of the work is stated; and in section 1.3 the methodology used is described.

1.1 BACKGROUND

This work has been done in cooperation with Intentia Research & Development AB, Linköping and the laboratory for real-time systems (RTSLAB) at the Department for Computer and Information Science, University of Linköping.

1.1.1 THE COMPANY

Originally founded in 1984 by four students from Linköping Institute of Technology (LiTH), Intentia is today an internationally established software company, based in Sweden. Revenues of 2,452 MSEK and 2,858 employees in 1998 makes Intentia the third-largest supplier of enterprise management systems in Europe and one of the eight largest in the world.

Intentia develops and sells *Movex*, an integrated enterprise management system for the management and control of business operations in manufacturing and distribution industries. Movex is aimed towards making business processes more efficient in the areas of logistics, production, finance, marketing and personnel administration.

Being an actor in a highly competitive line of business, it's imperative to Intentia to always be on the front edge of development. A great share of the revenues is therefore reinvested in research and development of new solutions. This continuous work is carried out within the Intentia group itself, by the designated research and development company Intentia R&D. More information about Intentia and Intentia R&D can be found on the corporate website: **www.intentia.com.**

1.1.2 E-BUSINESS WITHIN INTENTIA

The explosive development of electronic commerce over the past two or three years has generated a vast number of new business opportunities.

All estimates agree that this development is only the beginning of something even bigger, and that the focus gradually will change from Business-to-Consumer (BTC) to Business-to-Business (BTB) e-commerce. According to recent prognosis the BTC market expansion will stagnate over the next few years, while the BTB market, which is still at an early stage in development will expand at a very high rate [Forrester, 2000]. The reason for the stagnation in the BTC market is mainly that the expenses necessary for keeping customers in the BTC electronic market are very high, since consumers have proven inherently unfaithful (when it comes to e-commerce, anyway). However, the renowned IT analyst firm Gartner Group forecasts a fifty-fold increase in BTB e-commerce over the next 4 years, to a total value of 7.3 billion USD. [Gartner, 2000]

As the focus turns from BTC to BTB e-commerce, the providers of underlying systems must meet new customer demands. A great share of the BTC e-commerce solutions available today are delivered by specialized consultant firms and web developing companies, and are often not very well integrated with the customers' core systems and processes. As BTB e-commerce evolves, and becomes a core activity within the companies, even this will have to change. The e-commerce solutions need to integrate tightly with existing systems, business processes etc.

Here lie great business opportunities for deliverers of traditional enterprise management systems like Intentia. These systems are often long-term, great investments for the customers, and rather than trying to integrate some external solution for their BTB e-commerce efforts, these customers would prefer to have this functionality added to their existing system. This is likely to minimize the problems of integrating BTB e-commerce into their business.

Thus, this is a great future market for Intentia, as well as for its competitors. Much research and development is currently being done in order to provide e-commerce capabilities to the existing enterprise management systems, and in order to further develop the notion of BTB electronic commerce.

THE AGENT PERSPECTIVE

While much research have been done in the area for quite a long time, it's just in the last few years the notion of **software agents** has become popular and interesting for commercial applications. The main reasons for this are the birth of the World Wide Web with its vast amounts of publicly available information, and the rapid development of electronic commerce over the last few years. With great amounts of information comes the need for help in finding the right things, and the need to automate tedious tasks like manual search for the best price of a certain product when there are thousands of different offers.

While there is no generally agreed-upon definition of what an agent is or does in the academic world, the commercial ditto seems to have arrived at a de facto definition: the term "agent" is used there to denote a wide range of software that automates user tasks and tries to do this in some way that is perceived as "intelligent" by the user. Perhaps the most typical example of this approach is the various shopping and auction "bots" used to compare prices at certain products or automatically monitor online auctions to find the best deal for a certain product [C|Net, 1999]. Regardless of the level of "intelligence" or "agency" in these bots, their popularity has drawn the attention to agent technology in the field of electronic commerce.

Aiming to be a leading actor in the market of electronic commerce, it's a natural step for Intentia to explore the intelligent agent paradigm, and it's potential in this field. As a part of this agent technology exploration, Intentia has initiated several projects in the artificial intelligence field, particularly concerned with agent technology. The work is divided into two fields: one that's targeted towards artificial intelligence mechanisms and methods for business intelligence, and one that's focused on agent infrastructure and architecture. This thesis work is conducted as an initial part within the field of business intelligence.

USE-CASE FOR MRO PROCUREMENT

The starting point of this thesis work was a simple use-case for a business process that Intentia terms *MRO procurement*. This use-case describes Intentia's view of the buying process for MRO products from a BTB perspective. MRO stands for Maintenance, Repair and Overhaul, and refers to a wide range of product types, for example spare parts and semi-manufactures.

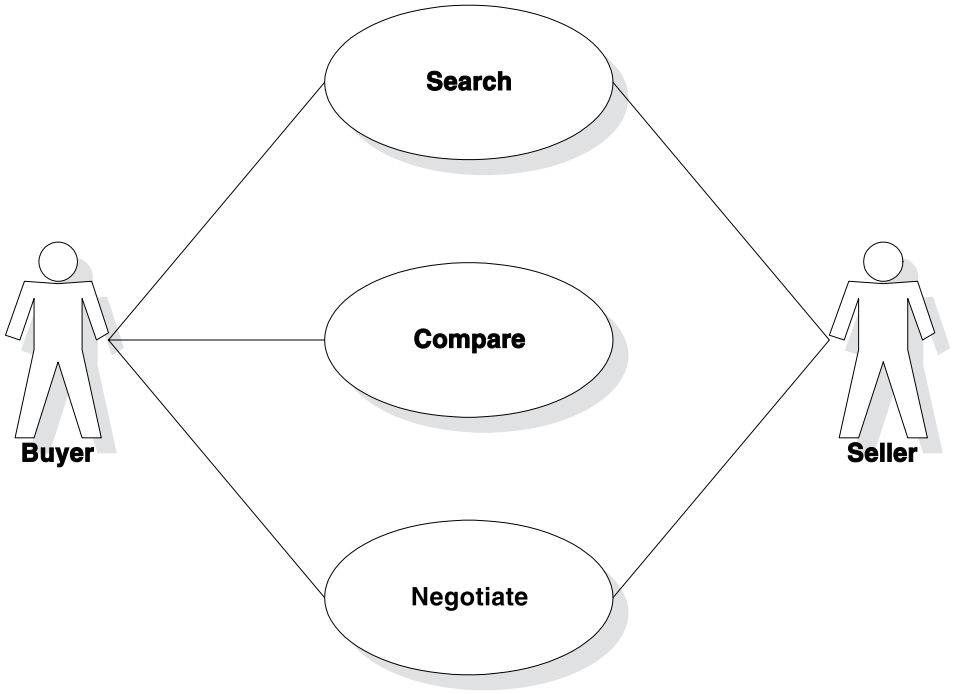

Figure 1. Use-case for MRO procurement.

The scenario is quite simple. The first thing the user has to do is to **find** the desired product, or rather find vendors that can offer it. Different vendors of course have different offers. Not only the prices of the product may vary, other parameters such as time-of-delivery or available quantity may be equally or more important. A typical example is found in airplane repairs. Since the price of a specific spare part is insignificant to the airline company compared to the great costs associated with having an expensive plane standing on the ground for a longer period of time, time-of-delivery is likely to be a more important variable than price in this case.

When the user has found all offers for the desired product, he naturally has to **compare** the different offers to find out where he can make the best deal. The comparison can be based on a single parameter, e.g. price, or a set of parameters.

The user can then try to further enhance the terms of his deal by **negotiating** with the different vendors in order to get a better deal. The simplest case of such a negotiation would be to try bringing the price down by telling vendor A that vendor B offers a lower price on the same product, and give A the chance to match this price. In the BTB field there may be more complicated conditions involved in the negotiating process, such as return deals or special agreements between the parties. One example of the latter is the notion of *preferred business partners* – trading parties who have previously established agreements or partnerships which affects the negotiating process. A buyer may choose to buy from a specific vendor even if that deal is not the best one, if this vendor is a preferred business partner. The partnership adds business value in some other way, compensating for less desirable conditions such as a higher price.

The **compare** and **negotiate** steps may be repeated any number of times (alternating) in the same buying round, until the buyer and some vendor have agreed on a deal acceptable to both parties.

1.2 PURPOSE

The purpose of this work is to investigate how Artificial Intelligence techniques can be applied within an agent context to provide increased user value in an enterprise application such as Movex. Being a modern ERP system, Movex spans across a wide range of areas, providing many diverse types of functionality. In this work however, we're only concerned with functionality for business-to-business (BTB) electronic commerce within the system.

In order to get a more manageable size of the project, we decided to further limit the work to a sub domain of electronic commerce of great potential, namely electronic auctions.

A simple prototype is also to be constructed from the results of the work. The purpose of this prototype is to demonstrate some of the proposed solutions in practice. The prototype is for internal use within Intentia only.

1.3 METHODOLOGY

In this section we discuss the method used in this work, the potential problems, and how these problems can be dealt with.

1.3.1 PROJECT MODEL

The methodology used for projects within Intentia is targeted toward software development projects, and therefore quite unsuited for work of a more explorative nature such as this thesis work. Therefore, we altered this method to one better suited to the characteristics of this task. The most significant differences to the original project model are that more time was spent in the preparatory phases, and less spent in the construction phase.

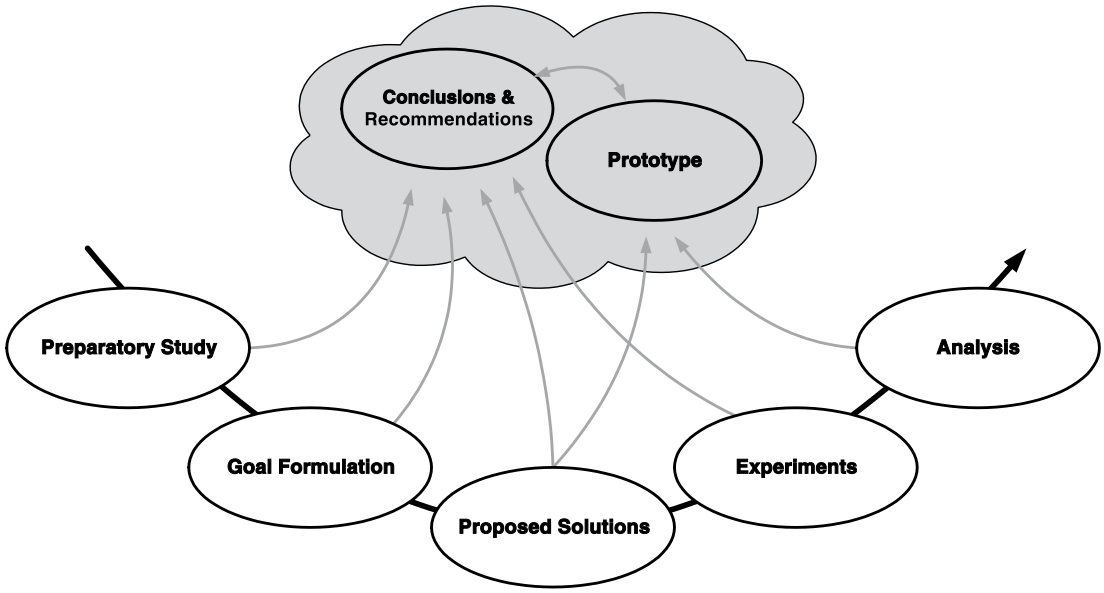

Figure 2. Visual method description, showing the different activities along a time line, and their respective contributions to the results.

- In the first phase, **the inception phase**, a preliminary study of the field was conducted. The original use case was elaborated upon and relevant techniques for the problem domain, like auctions, agents and AI methods were investigated.

- In the second phase, **the elaboration phase**, the techniques deemed viable in the previous phase were investigated at length. The outcome from the preceding phase also served as a basis for formulating the objectives of the work. A test domain, steel, was chosen. The objectives were stated as goals and delimitations.

- In **the construction phase**, user scenarios for the test domain were created. Solutions to problems in the test domain were proposed, and a system design was developed.

- In the fourth phase, **the experimental phase** experiments were conducted to test the solutions proposed earlier. Experiments were done both with bidding algorithms and AI methods.

- Finally, in **the conclusion phase**, the proposed solutions and the experimental results were analyzed and discussed.

The outcome of each and every phase was summarized in the main outcome of the project, a number of conclusions and recommendations, and a demonstration prototype.

The prototype was not included in the thesis work, but was an extension for internal use by Intentia.

1.3.2 POTENTIAL PROBLEMS

The work included a great amount of uncertainty due to its explorative nature. However, with our examiner and supervisor giving us guidelines, we felt that the risks of wandering astray were minimized. Furthermore, we had internal meetings at irregular intervals to see if the work and the purpose of the work coincided with each other.

The meetings with people outside the project, e.g. Intenta's steel expert Carl Tivelius and the IBM Agent Research Group, also provided good feedback on our efforts.

Chapter 2 PREPARATORY STUDY

This chapter will describe the different approaches and general methods that we have looked into in order to find suitable methods for the intelligent design of the system. The purpose of this chapter is twofold. First, we are going to describe the different tasks we found that needed intelligent behavior based on the use-case provided by Intentia. This will be done in section 2.1. Second, we present a survey of areas that we thought were useful in solving the tasks mentioned above. This will be done in sections 2.2 through 2.6. Finally, in 2.7, the conclusions of the preparatory study are drawn.

2.1 USE-CASE DISCUSSION

The use-case for MRO procurement presented in section 1.1.2 describes the BTB procurement process in quite an abstract manner. It makes few assumptions on exactly how products are procured. For example both ordinary and electronic procurement could be modeled by the same use-case. Does procurement from electronic auctions fit into this abstract view as well?

The "find" step is very straightforward and can be mapped straight onto electronic auctions. The task of finding products to buy is virtually the same regardless of whether the seller is represented as an auction site or a regular online store or marketplace. The "compare" step also fits well with the original use-case even when dealing with auctions: the task of comparing different offers translates in the auction domain to comparing the price levels of several auctions on the same product. In this respect it's simpler than would be the case for regular offers; usually just one parameter, the price, is used for the comparison, while comparing regular offers may involve more complex considerations. The "negotiate" step, finally, also fits well, although negotiation in auctions is quite different from regular negotiation of business deals. In the auction domain, we see "negotiation" as the task of deciding whether or not to place a bid in an auction at a certain time. Unlike in regular business negotiations, once a bid is placed, one is obligated to buy if this bid proves to be the winning one.

In the original use-case, the "compare" and "negotiate" steps could be repeated to facilitate multiple negotiation rounds. In the auction domain, this fact will be heavily exploited: one regular negotiation round here corresponds to a bidding round, of which there may be hundreds or even more during each auction. Each time a new bid is placed at one of the auctions we consider to buy from, the price levels need to be compared in order to decide on how to renegotiate, i.e. how the next bid should be placed. Also, we expect the auction domain to be more dynamic. The number of auctions involved in one procurement may change often: some auctions end while new ones start.

The original use-case was very much focused on the purchase in itself. The primary question was not **what** to buy, but **how** the procurement was made. In our opinion this focus was somewhat limited. The vast number of different vendors and different products available in a future world wide electronic market raise the need for supporting the user, not only in buying a specific product, but also in giving **advice** on what products to buy. By this we mean that the system should not only react to the user's input, it should also be able to pro-actively support the user. In the simplest of cases this could mean that the system maintains a history list of all procurements made. When a product that the user has shown interest in before becomes available at an auction, the system notifies the user of this fact. We proposed the addition of such an advice component to the original use-case to Intentia, who considered this modification highly interesting. Thus, the idea was added to the original use-case, as shown in Figure 3 below.

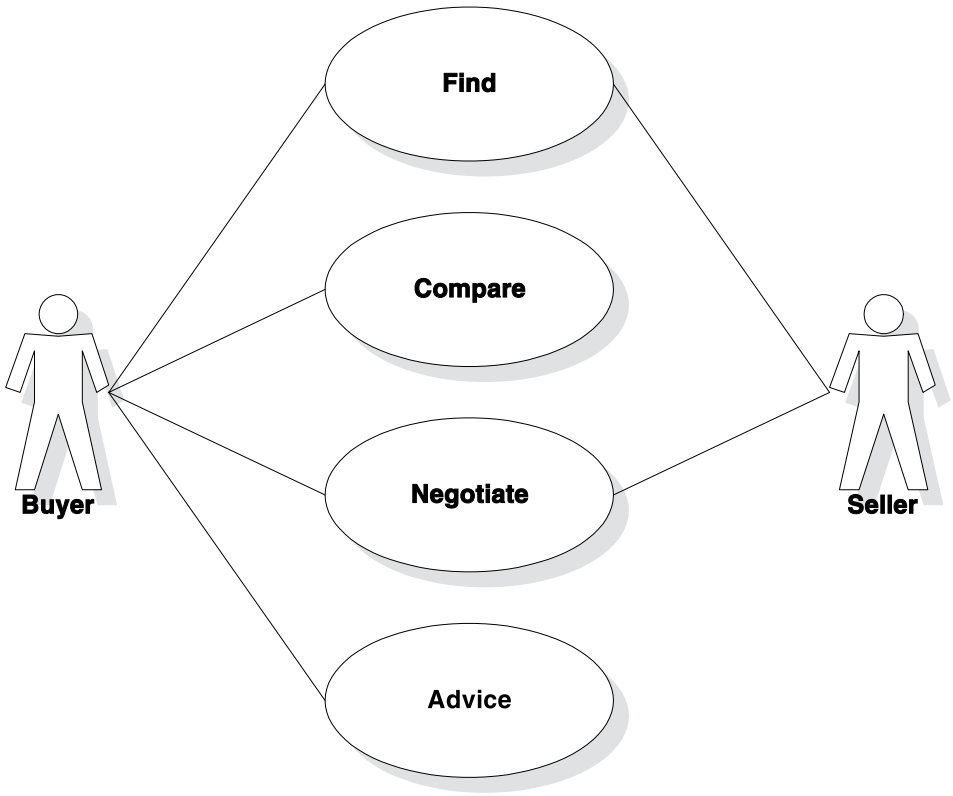

Figure 3. Modified use-case for MRO procurement.

It should be noted that the discussion above is based on observations of currently available electronic auction sites like eBay [eBay, 2000] and Bidlet [Bidlet, 2000]. An objection one might raise against this is that while the focus of this work is on BTB commerce, these sites are, with few exceptions, targeted towards the BTC market. However, today there's nothing implying that the BTB and BTC markets should be any different when it comes to electronic auctions; the user's perspective is more or less the same in both cases. Of course, we can expect higher quantities and somewhat different product categories, and it's reasonably safe to assume that BTB customers will be more competent and demanding as buyers than their BTC counterparts. However, the functionality needed is - in essence - the same for the BTB and BTC auctions, judging from the situation today and in the foreseeable future.

2.2 ARTIFICIAL INTELLIGENCE AREAS

As the title of this work, *"**Intelligent** Agents in an Electronic Auction Context"* indicates, we of course needed to look into the area of artificial intelligence and look at the different methods available. We looked at the following alternatives.

2.2.1 EXPERT SYSTEMS

A popular approach to decision support is to use an expert system. Expert systems are typically rule-based systems where a knowledge engineer takes the knowledge of a certain domain and codes this knowledge as rules and facts in a special type of database called a *knowledge base*. The rules are more or less "Rules of thumb", i.e. heuristics used to achieve a good result. In principle, expert systems solve the same task as regular learning systems, i.e. drawing conclusion about new cases based on the knowledge that has been given as input to the program. Expert systems rely on the fact that there is prior knowledge of a certain application and that we can *elicit* this knowledge from samples or interviews with domain experts and code this into our knowledge base [Wahlster, 1999].

Expert systems should be employed when "good knowledge sources exist and a careful and exhaustive engineering and testing effort is carried out" [Weiss et al, 1991]. This is the case in the sample domain that we used, but the implementation efforts are rather staggering for expert systems. Building an *ontology* [Gruber, 1993] to represent the knowledge in the domain is a non-trivial task for example. The resources available for a master thesis study are very limited and if we had built an expert system we would have needed private discussions with an expert from the prototype domain area in order to build a good system. Even in a simple system this would have taken weeks, which would have taken up too many resources for Intentia. That was simply not an option for us.

This is why we will employ the expert system paradigm neither in our prototype nor in our general design. Next, we'll describe several alternative methods to expert systems: decision trees and version spaces, neural nets, and clustering.

2.2.2 DECISION TREES AND VERSION SPACES

Machine learning is the traditional area of learning and is therefore the most natural choice for us when considering different methods.

The area of machine learning is the area that we know best from our studies and the two best-known algorithms in this area are *decision trees* and *version spaces*. From the beginning we thought that version space would be the most appropriate algorithm since it is more expressive than decision trees. After a while we however realized that the problems with version space are that it is prohibitively expensive with regard to computation and that, more importantly, *it can't handle noise*. This was the main reason that we chose to test decision trees instead.

One of the main advantages of decision trees is that the format of the learning is very easy for the designer to view, since it consists of rules.

2.2.3 NEURAL NETS

Neural nets are a way of classifying samples that is greatly inspired by biology and our expectations about how the brain works. A neural net is a highly interconnected network of nodes that exchange information with one another and therefore the area is sometimes called *connectionism*.

Neural networks have some good properties. For example, neural nets are able to handle much more complex problems than decision trees. One interesting feature of neural nets is that they do not make any assumptions about the population distribution, unlike most statistical methods. However, one of the drawbacks with neural nets is that they have a tendency not to give very good results because of the complexity of the method. It needs a lot more data than decision trees, for example.

Another disadvantage with neural nets is that the representation of the learning is not especially user-friendly. For a designer the representation presents a problem since if something goes wrong with the learning process and some samples are misclassified then it is difficult to know what went wrong. The method is something of a black box unlike for example decision trees.

2.2.4 CLUSTERING

Clustering algorithms are rather powerful tools for *discovery*, i.e. the finding of previously unknown information between the input data. The basic idea is to sort the input, i.e. the samples, into different *clusters* were within each cluster the samples are more similar to each other than to samples in other clusters. Clustering is a good way of finding relationships between data, relationships that were initially unknown. We expected such relationships from our sample domain and therefore clustering seemed like a viable alternative.

2.2.5 INTELLIGENT MINER

IBM produces a data-mining tool called Intelligent Miner. It consists of several AI techniques, among them neural networks, decision trees and clustering algorithms. It is normally used for large data sets, but the methods used in the program can also be applied on small data sets. Furthermore, Intentia and IBM are working closely together and therefore this product seemed like a very good product to do our testing with.

Apart from the AI algorithms mentioned above, Intelligent Miner also consists of something called associations. Association tries to find connections between entities in the data. For example, which articles sell together. We felt that associations couldn’t be used in our testing since it required buying patterns of the user, which was data that we couldn’t get access to. In principle, it might however be useful for giving advice to the user. For example, “buyers of the product x that you just bought also bought product y”. Unfortunately, we couldn’t test that feature.

2.3 BIDDING STRATEGIES

In this section we discuss several variants for bidding strategies that we examined.

2.3.1 POSSIBILITY THEORY AND FUZZY LOGIC

When we made our initial survey of what had been done before in the field of bidding strategies we found the Fishmarket Project [Fishmarket, 1999]. The bidding algorithms that the originators of this systems use are based on possibility theory. Possibility theory is based on fuzzy logic, which in turn is a logic without the crisp qualities of the normal logic. A proposition in fuzzy logic need not be true (1) or false (0) but can also have values in between. This is a way to insert uncertainty into logic and such a logic is ideal when we work with auctions and have to make decisions based on rather uncertain data.

We chose not to pursue this trail. This was not because we found it uninteresting or unsuitable for the auction context, quite the contrary. Possibility theory is very useful in the auction context, which we can see from rather large amount of papers that the Fishmarket Project has produced at different conferences. The reason why we didn't go into this was that fuzzy logic and possibility theory are rather large fields that we didn't have any training in. The time span of our project simply did not allow us to learn all the theory that we needed in order to make useful bidding algorithms.

2.3.2 STATISTICS

Bidding strategies must be able to handle situations where not all information that they need to make a perfect decision is available. This means that they have to operate under a great deal of uncertainty and statistics is the most natural way to model this uncertainty. We therefore chose to pursue this area in constructing bidding algorithms.

2.4 THE STEEL DOMAIN

In order to test the different learning methods in the Advice component and the different bidding strategies in the Buy component of our system we needed a test domain to perform our tests in. Intentia is a company that works in niches and therefore, they have more interest in some domains than others. Our choices included food & beverages, fashion and steel. We had a discussion with our supervisor at Intentia and he felt that steel would be the most appropriate domain. We also felt that steel was a domain where we would have good expectations to successfully apply learning methods. This was because we had a discussion with Intentia's steel expert Carl Tivelius and he felt that steel could very well be used for learning algorithms [Tivelius, 1999].

2.5 AUCTION THEORY

Since the purpose of the thesis stated that electronic auctions was the context in which we were supposed to work, we found it necessary to look at the different protocols that are available in auction theory. We also needed to see what software, if any, could be used as a test bed for our bidding algorithms.

We found that there are actually at least four different auction protocols that are used, i.e. *open-cry, Dutch, single round closed bid, and multiple round closed bid auctions*. We also found that there are other types of auctions, for example *double auctions*, where there can be both multiple buyers and multiple sellers. See for example [Friedman & Rust, 1994]. Due to time restrictions, these auction protocols are not considered.

2.6 AGENT TECHNOLOGY

We started out the survey on agent technology by trying to get a clear view of what an agent really is. From previous experience we knew that this task would be hard, if not impossible. We compared the different views and results of well known works in the field, as well as some previous work done within Intentia. This was because we considered it important to keep in mind the view of agent technology currently used by the company. Eventually we decided on a suitable approach to software agents, further discussed in section 3.1.1.

2.6.1 AGENT ARCHITECTURE

When designing a new agent system within an initial work such as this, we felt that it was important to use a model of agents as general as possible, so that it could be reused in future developments. At the same time it of course had to fit our specific needs for this work. We looked into both abstract, conceptual models in previous work, and real-life architectures implemented in agent construction frameworks, in order to see if these could be reused.

2.6.2 AGENT COMMUNICATION

Since we're interested in applying agent technology within a domain that is inherently distributed, i.e. trading via the Internet, it could be safely assumed that the communication between the individual agents would be an important issue. We did a minor survey of the work done in the field, and considered a few alternatives on agent communication languages.

2.7 CONCLUSIONS

The main conclusions of the use-case discussion were that we needed a design that could handle the dynamics of electronic auctions, and that we needed an extra component, called *Advice*, that would act pro-actively to give advice to the user regarding what to buy.

- We would use steel as our domain for testing the advice component.
- We would focus on finding a suitable, general way to model agents, and further focus on agent communication in the system design proposed.
- We would use the following methods within Intelligent Miner to test the advice component:
  - Clustering
  - Neural nets
  - Decision tree induction
- We would use the following methods to test the negotiate component:
  - Statistics

After a thorough discussion of the background we will be ready to define the goals more precisely. This will be done in Chapter 4.

Chapter 3 BACKGROUND THEORY

This chapter describes the necessary background that is needed to construct the system design in Chapter 5 and to be able to perform the tests described in Chapter 6 and Chapter 7. In section 3.1 through 3.5 we describe agent theory, information retrieval, auction theory, learning algorithms and bidding algorithms, respectively.

3.1 AGENT THEORY

In this section, some aspects of agent theory are discussed, namely definitions of agents, agent modeling and agent communication.

3.1.1 AGENT DEFINITION

When intending to use agent technology as an approach for solving a particular problem, the first thing that should be done is to try to get a good understanding of what an agent really is. This is, however, a non-trivial task. There exist no generally agreed-upon definitions of what an agent is or what it does, and certainly no commonly accepted formal definitions. Much work has been done in this area, but most proposed definitions are biased by the purpose of the work within which they are stated, and therefore seldom very applicable for other purposes. However, there have been some efforts made toward general agent definitions, two of which we discuss below.

[Petersson, 1998] makes the following definition of an agent:

An agent is a computer system that is (1) autonomous, (2) reactive and (3) pro-active.

This definition is a result of a comparison between several of the most used definitions of agents. By extracting the fundamentals of each of these common definitions and summarizing them, Peterson arrived at the three main characteristics above. These can, in turn, be defined as follows [Wooldridge & Jennings, 1995]:

Autonomy – agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

Reactivity – agents perceive their environment, and respond in a timely fashion to changes that occur in it.

Pro-activeness – agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

In this work, we will be particularly interesting in the last of these characteristics, the pro-active behavior. A system that is able to pro-actively interact with the user is considered highly interesting by Intentia. The benefits of pro-active behavior in Movex were one of the main results of Peterson's work; this work was the initial project on agent technology done within the company. In this work the pro-active behavior comes into play with the Advice component introduced in section 2.1.

Thus, after identifying three main characteristics of agents, we now turn to a rather different, but still very useful definition:

An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments.

[Shoham, 1997]

Although we don't give any precise definitions of these mental components here, the important concept of this definition is still obvious. The keyword above is "viewed as". By defining an agent in this fashion, Shoham places "agenthood in the eye of the beholder". Using this definition, virtually anything could be an agent, including many of the examples that most other definitions desperately try to rule out as definitive non-agents. However, what makes this definition a good one in our opinion is that an agent is only viewed as such if it is **advantageous to do so**. The message is a very non-dogmatic "One may apply agent ideas and techniques to a specific problem domain whenever the solutions can benefit from this".

This is a very appealing thought for this work, since we will not primarily be interested in exactly how agents handle and store knowledge; we only assume that agents consist of mental components such as above. Consequently, we can allow ourselves to reason about agents, even though one could certainly question their "agenthood" using a more conventional definition like those underlying Peterson's three-part ditto above. As we will show, our problem domain (electronic auction handling) could certainly benefit from an agent approach.

3.1.2 AGENT MODEL

[Laufmann, 1998] presents the notion of *Coarse-Grained Agents* (CGA:s). This model, in which an agent is represented visually by a triangle, is based on a simple model of human agents. In this model, the human problem-solving skills are divided into three parts: *general knowledge and behavior, task knowledge and behavior,* and *communication and interaction.* These three parts have been mapped directly to the CGA software agent model where they may be explained as follows:

The general knowledge and behavior part contains the ability to perform task-independent behavior, such as planning, reasoning and prioritizing. A typical facility in this part of a CGA would be some kind of inference-engine.

Task knowledge and behavior is sometimes referred to as *skills***.** These are specific to the task(s) of the individual agent, and may vary depending on the agent. For example, if we would view a web browser as an agent, its basic skill would be the ability to read and interpret HTML pages.

The communication and interaction part addresses issues related to the agent's ways of communicating with other agents. Such functionality could include various communication protocols on different levels and functionality for message security, for example.

Further, in the CGA, this model is extended with two notions specific for software agents:

Each of the three parts described above can be extended by the use of *functional plugs.* These are components containing new functionality, that can be added to improve or extend the functionality of the agent.

When looking at the model from a software engineering perspective, the need for facilities like multi-tasking, asynchronous message handling, shared memory, etc. become apparent. Such low-level functionality is needed to coordinate the different parts of the agent so it can act as a whole. Laufmann collectively terms these facilities the *agent core*.

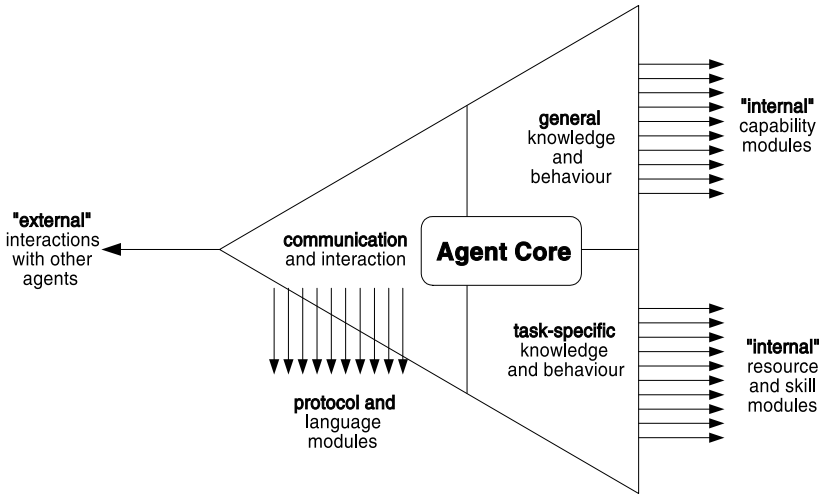

Figure 4. Coarse-Grained Agent model [Laufmann, 1998].

The main property of the CGA model that makes it suitable for our needs is its generality. It's impossible to predict how the BTB e-commerce domain will develop in the future at this early stage, and it's therefore imperative that we don't limit our view on the functionality of the agents to be used in this domain. The CGA model doesn't make assumptions about software platform, implementation language or hardware, which is nice since one of the few things we can say about the application domain is that it's inherently distributed and heterogeneous. Further, it allows for a wide variety of different functionality without altering the basic model.

Although general, the CGA model fits very well with the concepts of object-oriented programming. The Java language [Java, 2000], which is standard within Intentia, provides most of the facilities needed for the Agent Core right in the language itself. Internal multi-tasking and memory sharing, for example, can easily be implemented by using of the thread concept in Java.

By the use of classes and inheritance, the notion of functional plugs can be refined. While Laufmann saw these plugs as a way of adding new functionality to existing agents built according to the model as computer science progresses, we see them as a way to build agents with different functionality from a common base. In our view, a CGA in its simplest form consists of nothing but the Agent Core and three sets of empty "outlets", each set corresponding to one of the parts in the model. Such an agent skeleton thus has no capabilities at all. Different agents are then constructed by adding the desired functional plugs to each part of the agent.

In an object-oriented language, the "outlets" would be modeled as three abstract "plug" classes, from which the different kinds of functional plugs would be derived by inheritance. This would also easily allow for functional plugs to be hierarchical (through the use of inheritance). For example, consider an agent that needs to communicate over the Internet using the TCP/IP transport protocol. Furthermore, it may need to communicate both via the HTTP and FTP application protocols. This functionality would then be modeled using a general "communication and interaction" plug (class) providing TCP/IP communication, and two more specialized plugs (subclasses) derived from this one to provide the HTTP and FTP protocols, respectively (See Figure 5 below). This way, agent code can be re-used effectively, and developing time shortened.

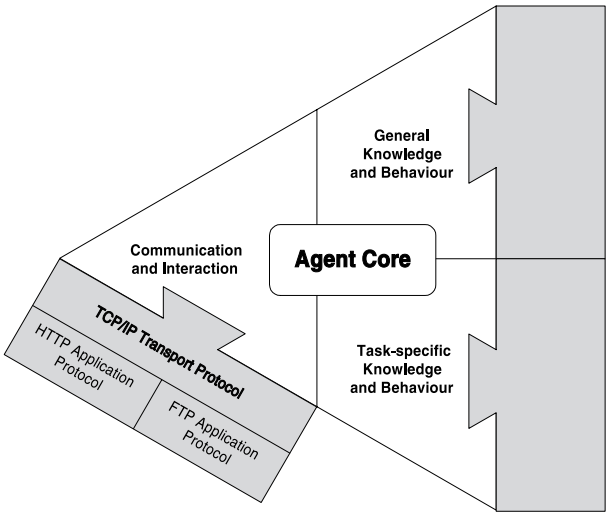

Figure 5. Triangle Agent using a hierarchical functional plug.

3.1.3 AGENT COMMUNICATION

The domain of electronic auctions (and of electronic commerce in general) is inherently distributed. Buyers and sellers (i.e. auction sites) may be spread all over the world, and are connected through the Internet. The purpose of agents for agent-mediated electronic commerce is to bring these buyers and sellers together, ideally in a way that requires minimal human supervision of the process.

Thus, the question of agent communication becomes crucial when designing an agent-based system for handling electronic auctions. We need a way for agents to exchange information of different kinds, both between themselves and with non-agent entities (e.g. the sites hosting the auctions).

This exchange of information needs to extend past simple message passing: the agents need to exchange data that represents facts, beliefs, opinions etc. – they need to be able to express characteristics of the content of the exchanged messages in order to act intelligently. Further, the agents' environment may not be as simple and controlled as is the case for regular peer-to-peer message passing. This is especially true in an open environment such as the Internet. Agents will need to be able to find other agents without prior knowledge about their location and be able to communicate with these, even if they were not explicitly designed for this. Also, the environment is dynamic – agents may come and go. Few assumptions can be made when it comes availability of individual agents and the state of the environment at any given time. Clearly, the communication is an important part of the intelligence in intelligent agents.

Many different approaches to this problem have been proposed, they are usually termed *Agent Communication Languages (ACL:s).* The perhaps best known of these is the *Knowledge and Query Manipulation Language (KQML),* which is discussed below.

THE KQML LANGUAGE

The Knowledge and Query Manipulation Language [Finin et al, 1993] is an ACL based on speech-acts. All messages are attached with a so-called *performative* corresponding to such a speech act. The performative is used to express some attitude about the content of the attached message, such as whether it's an assertion, a query or a response to a query, for example. The performatives are used by the agents to reason about how to handle the message content.

KQML is a three-layered architecture. It consists of the content-, the message-, and the communication layer. The content- and communication layers are fairly simple: the content layer consists of the message content itself. KQML is oblivious to the content, meaning that it completely ignores the information carried. The content does not in any way affect the message sending. The communication layer contains low-level information such as id numbers for sender and receiver.

The message layer is the KQML core. Here, the performatives are specified along with other information such as the names of the sender and receiver. Also, information about the message content can be specified. This means that KQML is capable of routing and analyzing messages although the content is unknown. Closely related to this is the notion of *facilitators.* In KQML, it's possible to have simple agents that provides services to other agents in the environment; they can forward and route messages on behalf of other agents, or act as information brokers providing information about certain resources can be found etc. This is very desirable in an open environment where many different and heterogeneous agents may be desired to interact.

A thorough description of the KQML language and related issues is given in [Finin et al, 1997].

Several extensions to the original specification of KQML have been proposed, one of which is Secure KQML (SKQML) [Finin & Mayfield, 1995]. SKQML addresses an issue that is quite important, especially in open environments like in this work, namely security. It extends the original specification with functionality such as encryption and authentication, making KQML competitive even from a security perspective.

In addition to what has been described above, KQML has a number of features desirable for an ACL in general, and for this work in particular:

- KQML has become a **de facto standard** for agent communication. Even if this isn't exactly a feature of the language itself, it's nevertheless an advantage. The fact that it's well established provides good future prospects for agent systems based on it when it comes to integration with other systems in an open environment.

- KQML is **content oblivious** (see above). This means that the message content can be virtually anything: simple text strings, an XML-based content language (see below) or a complex object. KQML does not specify any message content format; it carries whatever application language the sender uses, and the nature of the content has no effect on the message itself.

- KQML is **transportable.** The layered structure can be put on top of a wide range of different physical transport protocols, including HTTP, Mail and FTP. It can also be realized using Java RMI or comparable techniques. This provides a great deal of flexibility.

- KQML is **available.** Today, there are a large number of implementations of KQML available. These range from straightforward implementation of the language itself to complex agent construction tools that use KQML for agent communication. It's available on several platforms and for different programming languages including Java, which is important for this work.

Naturally, there are also drawbacks. Below, we discuss two of the most important:

The KQML language is very rich in features designed to allow complex communication between agents. That not many implementations need to use all of these features is thus not very surprising. Much to the contrary, experience shows that many implementations that use KQML end up using not even the basic features, i.e. the different performatives, to any great extent. All messaging is done using only one or a very few performatives [IBM, 2000].

Before going into the other (possible) disadvantage of KQML, we must first discuss some aspects of the message content that is particular to the e-commerce domain.

MESSAGE CONTENT: TRADING PARTNER AGREEMENTS (TPA:S)

As described above, the KQML language is oblivious to message content. However, this does not mean that the content can be neglected when discussing intelligent agent communication. On the contrary, the question of message content format is crucial, particularly for agents designed to function in the electronic commerce domain.

Trading between humans is regulated by **contracts** of some kind. Such a contract regulates the terms of a deal and represents an **agreement between the trading partners**. It identifies the seller and the buyer and stipulates which product is concerned and its price, how the payment should be made, when delivery is due etc. In the human world, the form of these contracts may vary, it may even be implicit, like in a deal on a fruit market, for example. In the world of electronic commerce however, particularly when automating the trading, such contracts must be made explicit or the information will be misinterpreted.

The term *Trading Partner Agreement (TPA)* for electronic commerce was originally coined by IBM in their TPAml project [Sachs et al, 2000], which is based on XML (see below). The basic concept however, is general and is to specify an electronic contract between the seller and the buyer so that they can exchange trading information without misunderstandings. In addition to an ordinary contract, a TPA also specifies how the product involved is described, and may also contains things like digital authentication of the involved parties to provide security.

THE IMPACT OF XML

The *Extensible Markup Language(XML)* [XML, 2000] is an industry standard meta language for specifying content. XML is a tag-based format designed to be machine readable (see below), and the basic concept is to separate content from its representation.

XML thus is very suitable for specifying TPA:s. Much work in these lines is underway; software giants like IBM and Microsoft are developing solutions for the exchange of trading information between businesses based on XML. They are known as *TPAml* [Sachs et al, 2000] and *BizTalk* [BizTalk, 2000], respectively. Furthermore, the United Nations body concerned with world trade issues, UN/CEFACT, has initiated work on an independent XML standard solution for electronic commerce, known as *Electronic Business XML* [ebXML, 2000]. While there are many differences between these different efforts, they all address the same main issue: how to exchange trading information electronically between heterogeneous parties.

The basic idea behind an XML-based TPA is simple: the parties, e.g. different companies, wanting to exchange trading information electronically agree on a common format for this trading information, a TPA. Using a proposed TPA standard such as TPAml, for example, would probably do this. The XML schema describing the chosen TPA format is then used as input to a program that turns the schema into code by each of the parties. The generated code realizes the exchange of trading information. The specific implementation details for each party are unimportant: one side may implement their system as C++ programs on a mainframe while the other uses agents written as Enterprise Java Beans [EJB, 2000] running on a workstation; as long as the XML schema is correctly translated, the parties will still be able to exchange information correctly. This process is described in detail in [Sachs et al, 2000]. Although the discussion is specific for TPAml, the general ideas are applicable to other XML-based TPA:s as well.

Returning now to the three-layer view of KQML communication in an e-commerce agent system discussed above, one can observe a very possible impact of XML on agent communication from our perspective:

XML is expressive enough to model not only the content layer, but the functionality of the underlying layers as well. An XML schema describing TPA:s could easily be extended to incorporate agent communication facilities like performatives and routing capability, thus blurring the distinction between the content and message handling layers from an implementation point of view. Since XML is very likely to become the language of choice for describing content, it's also very likely that developers will therefore question the need for KQML as a message-handling format. This is especially true since XML is widely accepted within the software industry – particularly when it comes to e-commerce solutions – while KQML, though well established, still is a language used mainly by the research community. It should be pointed out, however, that there currently are no agent communication solutions based on XML available.

3.2 INFORMATION RETRIEVAL

The "find" process of the use-case presented in sections 1.1.2 and 2.1 above is conceptually very simple: the buyer goes to a specific site and retrieve information about products, offers, available auctions, items available for bidding etc. – whatever information he needs for the next step. In the case of a human user, it's also quite simple in practice.

However, automating this information retrieval is nontrivial due to the fact that almost all presentation of such information is designed to be read by humans. Today, this is typically done through more or less advanced web page and search form solutions that vary a great deal from one site to another.

The discussion below does not specifically address electronic auctions; it applies to all kinds of software-mediated electronic commerce.

3.2.1 SCREENSCRAPING

The process of automatically retrieving relevant information from a web page without prior knowledge about its structure is often referred to as *screenscraping*, and a great deal of work has been done in this field. The most common applications are so called *comparison-shoppers*, which are pieces of software (often referred to as "agents" or "robots" to stress their autonomous behavior) that automatically find the best deal on a given product by screenscraping a great number of vendor web sites for price information. Examples of such comparison-shoppers are the ShopBot project [ShopBot, 2000] and BargainFinder [BargainFinder, 2000]. The ShopBot project facilitates a quite sophisticated method of screenscraping. Making a few very basic assumptions about the general structure of web pages, a learning algorithm based on pattern matching is used to successively improve the automatic search. This has proved to be a surprisingly effective method.

After the "scraping", the collected information is displayed to the user, who then hopefully can find the place that offers the best price on the desired product.

The main problem with screenscraping is that the quality of the information collected generally cannot be guaranteed. In this case, how do we know that an offer found isn't out of date, or if one offer includes taxes while another doesn't, for example? Even if the method of retrieval used is very advanced and accurate, it's very likely that the final validation of the information collected has to be done by a human user, and thus complete automation of the task is impossible.

3.2.2 INFORMATION RETRIEVAL BASED ON PROTOCOLS

The problem above of course stems from the fact that the software that collects the information about the product offers does not have any knowledge about how this information is represented. When humans communicate, for example when a seller presents product information to a potential buyer, they have to commit to certain unspoken "rules of communication" in order to avoid misunderstandings; these are often referred to as *Grice's Conversational Postulates* in cognitive science. The fact that such an unspoken agreement between the communicating parties needs to exist is often referred to as *the cooperative principle* [Ashcraft, 1994]. We will not go deeper into this here. It's a complete research field in itself, and far beyond the scope of this work. However, when replacing the human reader with software, these "rules of communication" must be made explicit in order to avoid the risk of misunderstandings. The presenter has to represent his information in a way that's fully understandable to the reader; this can be thought of as an "agreement" on communication between the two parties.

The above is one of the basic ideas behind the electronic TPA:s described in section 3.1.3 above. By letting the seller and the buyer use a common specification for how trading information is represented, misinterpretations in software can be avoided and the need for human supervision can be reduced. With the introduction of the XML standard (see section 3.1.3), the tools needed to handle the theoretical aspects of this problem are available. However, commercial aspects are sure to complicate things.

Today, virtually all of the actors on the BTB e-commerce market seems to agree on this general idea as the way of building systems for BTB trading that allows for non-human participants (e.g. intelligent agents) in the future. The concept of XML-based trading protocols is widely embraced, although there are as many different designs as there are companies in the market. Great efforts are being made to agree on an open standard, but meanwhile, big companies like IBM and Microsoft market their own solutions, hoping to create their own commercial de-facto standard. It's reasonably safe to assume that many different XML-based trading protocols will exist on the market for several years to come, and that bridging between these different protocols will be a common task for providers of BTB e-commerce systems.

3.3 AUCTION THEORY

In this section auction theory is discussed, i.e. different types of auction protocols and the different questions that these protocols give rise to.

3.3.1 INTRODUCTION

Auction theory has two parts that are interesting to this project. The first part is the *purpose* of an auction and this will be discussed in section 3.3.2. The second part is the different protocols of auctions. Contrary to common belief there are several protocols for conducting auctions. The ones that we will discuss here are *open-cry*, *Dutch*, *single round closed bid* and *multiple round closed bid* auctions. In sections 3.3.3 through 3.3.6 we will present the different auction protocols and their respective problems. For a thorough discussion of the auction scenarios, we refer the reader to [Kumar & Feldman, 1998]. A discussion about multi-auction scenarios and how to handle them appear in Chapter 8, where the Compare Agent is discussed.

3.3.2 AUCTION PURPOSES

Auctions are used in many contexts and the intentions of the participants are not always the same in different contexts. There is therefore a need to explain one major distinction between auctions and classify the project domain into one of them. The definitions *friendly* and *hostile* are not established terms, but are invented by the authors.

FRIENDLY AUCTIONS

Providing information about yourself to your opponents is the basis of *friendly* auctions. In an auction there is of course information that can be deduced about your opponents by observing the actions that they take, but not much else. In a friendly auction the bidders give information to one another, for example how much money they have left, in order to maximize the *utility* for everyone.

Friendly auctions are used in for example resource sharing algorithms. There the bidders compete for different resources like printers or CPU time, and the goal is to have a maximum degree of utility in the system. Therefore sharing your information is necessary.

HOSTILE AUCTIONS

When thinking of auctions in real life they are thought of as being held in a *hostile* environment. Hostile in this context means that a bidder always tries to get the product to the most beneficial price that is possible. Every bidder tries to maximize its own personal utility and the needs and wants of the other bidders are completely ignored. This of course means that giving information to the opponents is out of the question. Information can, as was said in the previous section, be deduced from the actions of a bidder, but no additional information is given to the opponents. For example, only the bidder himself knows how much money he has left.

It is not a bold statement to say that all BTC and BTB auctions are hostile. This is of course due to the fact that the bidders have no reason to help their opponents since they are competitors and not partners. This means that our auction context can be said to be hostile and this in turn will be the basis for some of the assumptions that we make when we construct bidding algorithms in section 6.2.

3.3.3 OPEN-CRY AUCTIONS

The open-cry auction protocol is the protocol that most people associate with auctions. A starting bid is given and after that the potential bidders give their bids in ascending order and the one with the highest bid wins the item for sale.

The ending time of the auction can either be a pre-set time when the auction closes or that no bid has been posted for a certain time period, so that a time-out has set in.

The problems that a designer of an open-cry auction agent faces are several. The most basic one is the maximum price that one is willing to pay for the item. There are of course vast possibilities to solve this problem, all with heuristic capacities. This problem will be called the *end price determination* (own definition) .

Another problem that is specific to the open-cry auction protocol, is **how** to bid on the item to get an optimal price. Shall the agent place a high bid early on in the auction to scare other bidders away or shall he place very low bids hoping to make a good deal? This problem we call *strategy selection* (own definition).

3.3.4 DUTCH AUCTIONS

Dutch auctions differ substantially from open-cry auctions. The auction sets the *start price* very, very high in the beginning of the auction. The price then *decrements* with a fixed price if no one has made a bid for the item within a *time out*. This process continues until one of three things happens: A bidder buys at a given price, the *reserve price* is reached, or a collision occurs. A collision occurs when several buyers bid at the same price. Then the price of the product is raised a certain amount and the auction continues from there. The reserve price is the price level where the item is withdrawn from the auction. The seller sets the reserve price so that he is guaranteed not to loose too much on the deal.

Typical auctions where the Dutch auction protocol is used are auctions of products with short lifespans, e.g. fresh fish or dairy products. This protocol is used for these kinds of products since it has a set deadline when the item is withdrawn from the market, i.e. $deadline = start\_time + \frac{start\_price - reserve\_price}{decrement} * time\_out$. This is the time when the reserve price kicks in and then the product is withdrawn. Having a determined deadline as we have above is very useful if the product has a short life span. The sellers (and the buyers) don't need to worry that their product will turn bad before the auction is over. Of course the product can be withdrawn earlier if a buyer bids on the product.

3.3.5 SINGLE ROUND CLOSED BID AUCTIONS

Single Round Closed Bid auctions use an easy to understand protocol. The participants know that the product will be auctioned at a given time and the participants may give their bids to the auction. The bids are kept secret from the other participants; hence the name *Closed Bid*. At the given end time the bids are publicized and the winner gets the item. This process of disclosing the bids is only done once; hence the name *Single Round*.

In [Milgrom et al, 1982] it is pointed out that the Single Round Closed Bid auction has the same semantics as the Dutch auction. The difference is just on the surface. From a pure game theory point of view it is the same decision that we have to make with the same information available. In both auction protocols the choice must be made at what price we will bid and furthermore this is the *only* decision we have to make. Or as Milgrom puts it: *The apparent complexity of the possible bidding strategies in the Dutch auction is a chimera: the only real choice a bidder has is to select his "bid" p.*

Still, the empirical studies that have been made regarding these two protocols show that the Dutch auctions generally yield a lower price. The reason for this is probably that the chimera that Milgrom talks about, i.e. the syntactic difference between the two protocols does make a difference from a psychological perspective. A person standing in a Dutch auction probably gets excited by the chance to make a bargain and therefore decides to wait a little longer before he offers his bid. This sort of urgency factor is of course not available in the case of Single Round Closed Bid auctions.

Now, does the factor mentioned above make a difference in a software agent context? Well, if the assumption is made that all of the bidders are software agents then the urgency factor is not important. An agent that only needs to determine what price to bid at doesn't see this psychological difference and therefore the same decision algorithm can be used for both protocols. Hence, the protocols are equal. This will be of use when the Buy Agent in described in Chapter 6.

The assumption that all bidders are software agents is of course not realistic in electronic auctions today. On the other hand, what this system is trying to describe and give guidelines for is a product that is supposed to exist in an *agent* framework and therefore the assumption is valid or at least plausible.

Typical sorts of auctions where the Single Round Closed Bid auction is used are auctions with high-value, non-rush products like oil or land rights. The urgency aspect as we discussed above is probably not applicable in the context of for example oil rights since the bidders are typically large companies that have made careful prospects and are therefore probably not rushed.

3.3.6 MULTIPLE ROUND CLOSED BID AUCTIONS

The Multiple Round Closed Bid auctions work like this. The auctioneer posts the leader of the auction at different pre-specified points. At each point the seller has the opportunity to either accept the offer or let another round start. The maximum number of rounds is known in advance.

A very interesting thing to notice is that this type of auction is very similar to an open-cry auction. There is just one difference. The posting of a new leading price is done at regular intervals in Multiple Round Closed Bid auctions while they are done at irregular intervals for an open-cry auction.

The Multiple Round Closed Bid auction is presented here only because of its relation to the Single Round Closed Bid auction. This protocol is discussed neither in Chapter 6 nor in Chapter 8, since it is very rarely used.

3.4 LEARNING ALGORITHMS

This section will describe the different artificial intelligence methods that we have used throughout the project. All of these methods are used in the Advice Agent in order to help the user select what product he would like to buy. There are two reasons why we have chosen these specific methods in our work. First of all, they are the most commonly used methods and second of all, they were available to us in the Data Mining tool *Intelligent Miner* from IBM Corporation.

3.4.1 OVERVIEW

We are now going to describe learning algorithms and see their advantages and disadvantages. Machine learning is the task of taking knowledge, *classifying* it and then from this classification making *predictions* for future cases. For example, assume that we have a set of samples consisting of auction data with features such as: who was bidding, on what product etc, and what the result was (did they get the item or not). From this we could try to classify the samples into groups, trying to decide what strategy makes us win a certain auction. This process is called *classifying* the samples. From this data we can then make a choice on what we should do if a similar situation occurs. This is called *prediction*.

A note to the reader: This is just an example on what we can use machine learning for. In our project we have not used machine learning for predicting auction strategies although that would surely be possible.
Another example of machine learning is the question of whether a product is interesting for the user or not. All samples are classified into two groups, *interesting* and *not interesting*, and from that information we can make a prediction whether new products that are to be auctioned are interesting for the user. This is the application that we use the machine learning algorithms for, although we classify the samples into more than two groups.
Of course, there are a number of pitfalls to avoid when employing these techniques and we will review a few of those risks, i.e. those that are relevant in our domain. Our domain will be steel as was discussed in 2.4 and the task for the learning is product selection advice.

3.4.2 THE PROBLEMS AND HOW TO HANDLE THEM

A few fundamental problems exist when we try to learn by example. A review of the most important ones is made in the sections to come.

INCORRECT SAMPLES

One problem is that the samples that are given as output of the learning algorithm might be incorrect. Two types of such inconsistencies exist, *false negative*, which means that the hypothesis says that the example should be negative when it actually is positive, and *false positive*, which means that the hypothesis says that the example is positive when it is in fact negative. This could give dire consequences if the domain consisted of cancer patients. A false negative would then mean that the test would say the subject didn't have cancer, when in fact he did. The consequences could be fatal. Therefore we must compute the cost of all the false negatives and false positives, the so-called *misclassification costs*. We assume that we represent the classification as an n*n matrix, where n is the number of different classification categories, the columns are the correct classifications and the rows show the classification of the program. This way, the diagonal shows the correct classifications and the other entries are incorrect classifications. In Table 1 below we see that there are 10 correct classifications of type one, while there are 3 examples that have been mistakenly predicted to belong to type 2.

Correct class/ Predicted class	1	2
1	10	7
2	3	30

Table 1. Misclassification costs.

The cost of misclassification can be expressed as $\sum_{1..n}\sum_{1..n} E_{i,j}C_{i,j}$, where $E_{i,j}$ is the value at position (i,j) in the table and $C_{i,j}$ is the cost of such a misclassification. The $C_{i,j}$ value when i = j is 0 since that is not a misclassification.

This approach seems simple enough but one must remember that it is not a trivial process to determine the $C_{i,j}$ costs. Is it twice as expensive to have a false negative as a false positive in the cancer example above? Maybe more? The normal procedure is to cheat and say that all weights are equal, which often is very far from reality.

ERROR RATES

Of course we must somehow measure how correct our prediction is. Theoretically, the *true error rate* of a method is defined as the error rate of an asymptotically large number of samples that converge in the limit to the actual distribution of the population investigated. This is not a realistic approach for a real-world application since the number of samples always is limited and typically rather small. Instead we define the *empirical error rate* as:

$$error\_rate = \frac{number\_of\_errors}{number\_of\_cases}$$

The empirical error rate is also called the *apparent error rate* since it is the appearance of the error rate that the tested method gives before being tested in the real world. Now, in testing the error rate a few mistakes can be made. A very common error in the past was to test the system on the samples that the system was built on. Then of course the error rate will be very low, most often zero. To build a system that is only fitted to the sample cases is a grave mistake, since we don't have a clue whether it resembles the true population of samples. This is called *overspecialization*. To remedy this problem we can split the sample set into two sets, a *training set* and a *testing set*. We train, or build the system on the training set and then test it with the testing set. Normally you split the sample set 70/30, i.e. 70 per cent of the samples goes into the training set and 30 per cent goes into the testing set. The problem is when we have few samples, which seems to be the case rather often. Then we can't spare many samples for the testing set, since we need them to build the system. This means that it is of paramount importance that we use the testing set very efficiently. We will investigate a method that does this very well, the so-called *leave-one-out* method.

LEAVE-ONE-OUT

The leave-one-out method works very well even for small samples since it reuses the information in the samples in a very efficient way called *resampling*. It is not very good for large sample sets since the computational costs will then be high.

The method consists of training the system on all samples but one, hence its name, and then testing the system on the remaining sample. We will then iterate this n times, assuming that the sample set size is n, and each time train on $n - 1$ cases and test on the remaining one. The apparent error rate is the quotient between the number of errors and n.

There are a few other problems regarding the choice of classifier and these will be handled next. Re-sampling will play a prominent part there as well.

CLASSIFIER COMPLEXITY AND FEATURE DIMENSIONALITY

One of the biggest issues in developing a classifier is how many features should be used in order to classify the data, the so-called feature dimensionality. Our intuition says that we should take into account as many features as we can, since more information should yield a better result than less. Unfortunately this intuition often fails us. This is because some features could be full of noise and damage the model instead of improving it. It is also possible that a feature could be dependent on another feature and such a feature would in effect be counted twice. So a good heuristic would be to use as few features as possible.

The other big issue that we will discuss here is the classifier complexity. Intuitively, we feel that the more specific a classifier is, the better it has described the samples. This intuition is also wrong. The reason can best be seen by example. Assume that we have ten samples with certain features (like if the sun is shining or not and if it is a warm day or not) and one feature describing whether we play tennis or not. The easiest way to construct a classifier predicting whether tennis is played or not from these samples is to make a disjunction of the samples where tennis is played and say that only samples that agree with the disjunction of features is a situation when tennis is played. In essence, the classifier will only accept the ten samples and nothing else. It does not scale at all. We can see no pattern between the samples, for example. The classifier has become *overspecialized* to the samples and this is definitely not a desirable property. What we want is a solution that scales when other samples are tested in the classifier. In general, such a solution is simpler than the one above. Simple in this aspect means that is does not overspecialize to the samples.

In the following sub-chapter we will explain the basics of the main machine learning methods, i.e. *statistical approaches*, *neural networks*, and *decision tree induction*. We will only describe the statistical methods superficially, since our testing will be done with neural networks and rule induction. We chose to include the statistical methods for the sake of contrast.

3.4.3 STATISTICAL APPROACHES

The statistical methods can be divided into two sub-groups: those that assume a certain distribution on the samples and those that do not. The first approach that we will describe is *linear discriminants* and it assumes a distribution on the samples while the second approach, the *nearest neighbor method*, does not.

LINEAR DISCRIMINANTS

Assuming that we have *d* number of features, the process of linear discriminants can be viewed as dividing a *d*-dimensional room into two parts, i.e. with a (*d-1)*-dimensional hyper plane. This would mean that in a two-parameter setting the dividing plane would be a line as is shown in Figure 6 below.

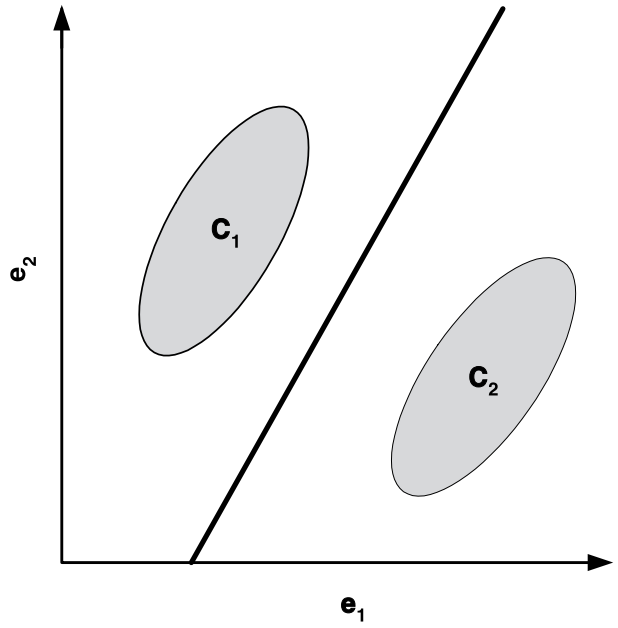

Figure 6. Linear discriminant. The line divides the classes C_1 and C_2.

All classes of problems cannot be classified using this approach. The most common example is the XOR example that is shown in Figure 7.

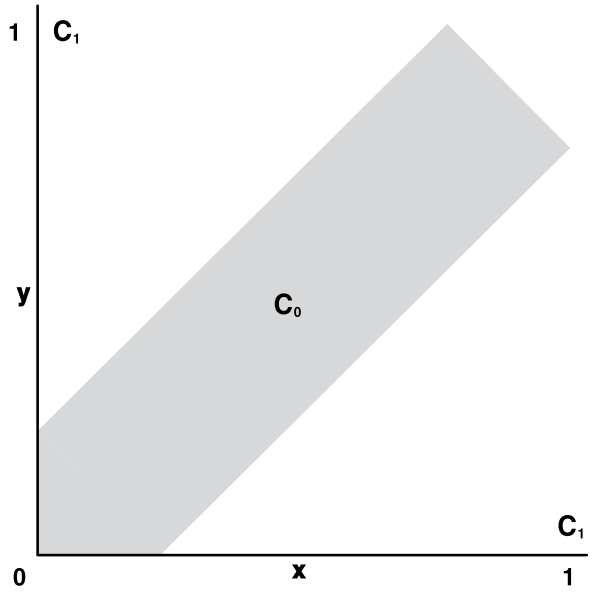

Figure 7. XOR example. A single line can't separate the classes C_0 and C_1 since the C_1 class exists in two areas of the figure and the C_0 area separates the two areas.

As can be seen no line can divide this example into a correct classification. We have to use different methods like a neural network.

As we can see the linear discrimination problem tries to divide the world into *two* classes, a so-called binary discrimination problem. What if our world contains more classes? Then we can rephrase our question so that it will be a sequence of binary problems, each class vs. all the others.

The result from the linear discrimination problem is an expression for each class: $w_1 e_1 + w_2 e_2 + ... + w_d e_d - w_0$ where $\{e_1, e_2, ..., e_d\}$ is the list of features, *d* the number of features and w_i the weights to be determined. When a new sample is presented to the classifier it will be tested for all the *c* equations where *c* is the number of classes. It will be assigned the class with the highest value.

The problem is of course to determine the weights w_i. A complete description of how this is done is beyond the scope of this text, but the main idea is that one has to make assumptions about the distribution of the samples and their classes. The most common method seems to be that one assumes that the values of a certain class are centered on a mean and have a certain variance. Assume also that we have just two classes to choose between. Our classifier will be very simple if we furthermore assume that the variances of the two classes are equal. The linear discriminant for that scenario is shown in Figure 8.

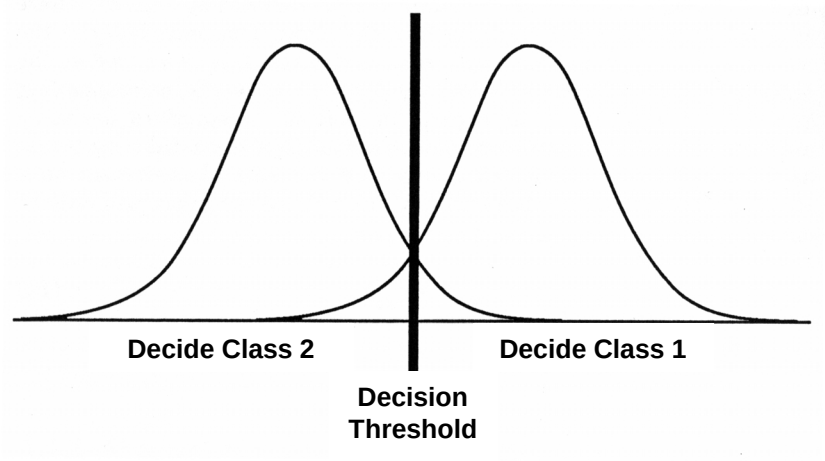

Figure 8. Linear Discriminant with an assumption about the distribution of the samples.

NEAREST NEIGHBOR METHODS

This method makes no assumptions on the distribution of the samples. The idea is that when we encounter a new sample we check it against our classifier. If the sample exists in the classifier we're finished. Else we check which neighbor is closest to the sample and use that class to categorize the new sample. Distance can be absolute distance or Euclidian distance, for example.

The nearest neighbor method demands no computational effort for the classification process, but all the more computation when a sample shall be predicted. All the existing cases must be compared with the new sample in order to find the one with the shortest distance.

3.4.4 NEURAL NETWORKS

A neural net is a method for learning that is appealing from a biological point of view. It is supposed to imitate the learning in the brain with many different nodes that are intertwined in a large network. The different neural network methods do not assume any specific distribution on the sample set, which makes it widely applicable but on the other hand it takes many samples in the input for it to train. The basic unit in a neural network is the *perceptron*, which will be described next.

THE PERCEPTRON

A perceptron is a node with *n* in-values and one out-value, as can be seen in Figure 9 below.

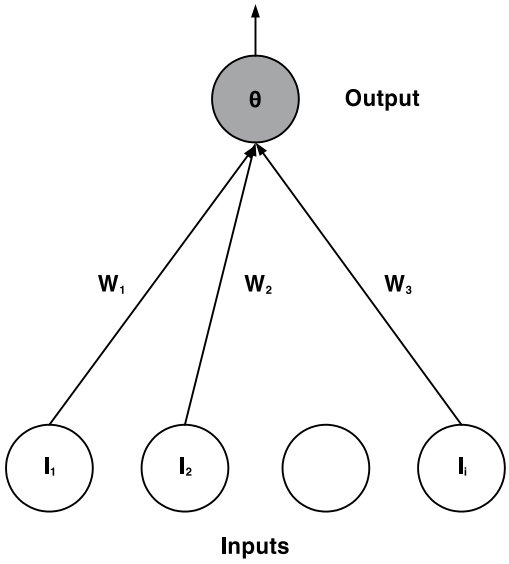

Figure 9. A perceptron.

Each perceptron contains an equation of the same form as the linear discriminant expression above. This expression can be re-written as:

$\sum_i w_i I_i + \theta$ where I_i is the *i*-th input variable, w_i is the *i*-th weight and θ is a constant often called the *threshold*.

The output of the perceptron is either one or zero, depending on whether the equation value reaches over zero or not:

$$Output \begin{cases} 1 : if \sum_i w_i I_i + \theta > 0 \\ 0 : otherwise \end{cases}$$

The weights of the perceptron are initially set to a random number between 0 and 1, but are changed during the training process.

The training process consists of an iterative process called *epochs*. In each epoch every sample is tested in the algorithm and the weights in the perceptron are changed if the output of the perceptron is different from the true value. The weights are altered in the following way:

$$\Delta W_i(t) = (T - O)I_i,$$

where T is the true value of the sample and O is the output from the algorithm. If the true value is 1 and the output is 0 then the weights increase, which is intuitive since the value of the algorithm must have been too low (below zero). The reverse is true if the true value is 0 and the output is 1.

The threshold is also altered:

$$\Delta\theta(t) = (T - O)$$

The perceptron approach is in itself a powerful learning mechanism, but it is not as good as for example the easily implemented linear discrimination technique. These two approaches are equivalent in expressive power, but the linear discrimination technique tends to yield a better result, probably since it makes an assumption about the distribution of the samples. Furthermore the perceptron training mechanism has a tendency to oscillate between different weights and therefore may not always arrive at the optimal solution. It is also susceptible to local minima. Instead of using perceptrons individually, they form the basis of *multilayer neural networks*.

MULTILAYER NEURAL NETWORKS

The perceptrons can be intertwined to form a network like in Figure 10 below:

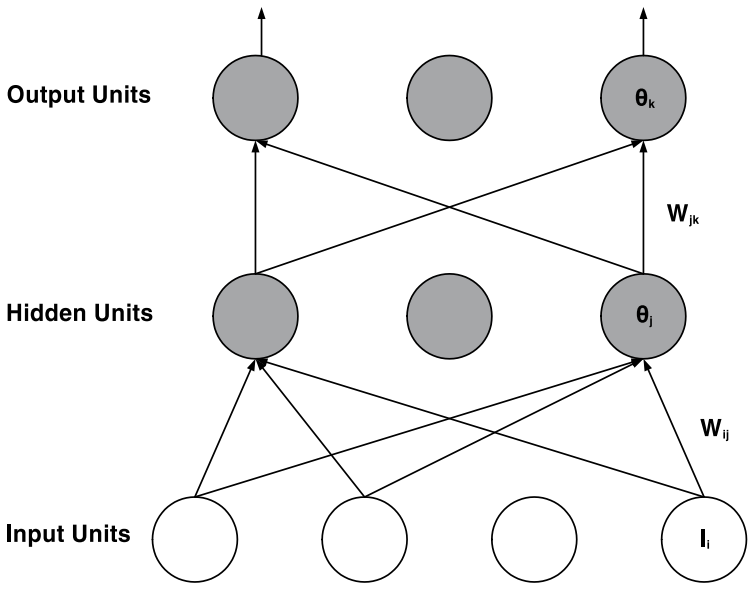

Figure 10. Neural network. The input I_i is weighted with w_{ij} to yield, in conjuction with other input units, the value of the internal node θ_j.

As we can see, there exists an intermediate layer or *hidden layer* inside the network. It is called hidden because the user does not see its output. This extra layer gives the network an expressive power that is much greater than with an individual perceptron. This type of network can for instance solve the XOR-problem cited above. The expressive power is not dependent on the number of hidden layers that exist in the network so the only reason to use more than one hidden layer is if the application needs them for clarity.

The training procedure is called *back propagation* and will not be described fully here. It is a variant of the perceptron training procedure, where the weights and the speed at which the training occurs are altered.

The multilayer neural network is susceptible to certain problems, i.e. local minima and very slow convergence towards the optimal solution. If the learning rate is too slow then the network converges very slowly and if it is too fast then the network may oscillate between bad solutions. A careful setting of the training rate is therefore necessary.

3.4.5 DECISION TREE INDUCTION

There are two methods within machine learning that are famous, *decision tree inducation* and the *version spaces algorithm*. The decision trees are not as expressive as the version spaces algorithm but have other advantages like being able to handle noise, something version spaces can't do. This drawback with version spaces makes it more or less useless for the kind of applications that we have and we therefore won't describe it further.

DECISION TREES

Decision trees rely on the concept of "non-incremental learning from examples" which is a possible drawback in our domain since we want to gain knowledge as we are working. However, it may still be applicable. A simple solution would be to construct a new decision tree at certain intervals and by doing that get some form of dynamic behaviour.

The idea of the decision tree is to form a tree where the nodes are conditional expressions. Depending on the answer a certain path in the tree is chosen until a leaf is found. A picture of a decision tree is found in Figure 11 below.

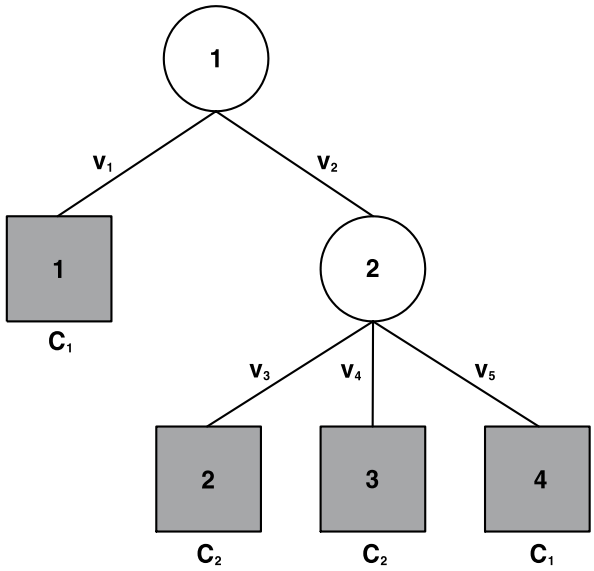

Figure 11. A decision tree. The v:s represent the choices at each node and the C:s represent the classes. For example, a sample conforming to v_2 and then v_5 is classified as C_1.

How is this tree formed then? Assume that we have a set of samples where each sample has the features $\{e_1, e_2, ..., e_n\}$ and a class membership c_j. The most common approach is the ID3-algorithm where the feature with the most *information content* (the lowest entropy) is chosen. Another common approach, which by the way is the one used in Intelligent Miner, is the so-called *gini function* [Mehta et al, 1998]. The objective for these two approaches is that at each node in the tree, we try to decide upon a *split* of the node into parts where the randomness or impurity is as low as possible. A node is impure if it consists of samples from more than one class.

The expressions for these approaches are stated below, where p_j is the probability for class j at the node to be evaluated.

$$-\sum_j p_j \log p_j \text{ (entropy)}$$

$$1-\sum_j p_j^2 \text{ (gini)}$$

Both equations have the property that if a node is impure then the value is high. The split is done on the feature e_i that produces the lowest value; i.e. gives the purest nodes.

Intuitively, this means that the feature that can make the best separation between the classes is chosen. A feature that has the ability to say: "All the samples where my feature has the value X belong to class c" is of course rather powerful. For example, we want to know if tennis is going to be played (class 1: yes, tennis will be played and class 2: no, tennis won't be played). Furthermore, all of the training samples where the feature weather has the value rain belong to the class 2. Then the feature weather has very high information content and will most probably be chosen early in the tree construction.

After the best feature is chosen the samples are split into different groups depending on which class they belong to and a new feature will be found at each new branch in the tree.

The above procedure will have the effect that each path through the tree comes from one or more samples. This of course means that if there is noise in one of the samples then the decision tree is incorrect in that branch. This can of course be remedied if the sample is removed, but the whole tree must then be remade, which is a very expensive maneuver.

The Intelligent Miner decision tree algorithm is optimized for large data sets, but the gini function that it uses is equally good for small data sets, so that won't be a problem when we start testing. The optimization is seen mostly in the way the Intelligent Miner accesses its data, which is done breadth-first. Breadth-first methods are uncommon when all the data can be stored in the memory, but since Intelligent Miner is supposed to handle extremely large data sets that can't be stored in memory it uses breadth-first in order to only have to insert each node into the memory once.

For more information regarding the Intelligent Miner decision tree algorithm, see [Mehta, et al, 1998]. For more information about learning algorithms, see [Weiss et al, 1991].

3.4.6 CLUSTERING

Clustering is a set of techniques that takes a set of samples and divides them into a number of clusters. The number of clusters is sometimes preset, but not always. The main combining feature of all clustering methods is that "the clusters which are formed should have a high degree of association between members of the same cluster and a low degree between members of different clusters" [Frakes, 1992]. Clustering is also sometimes called *automatic classification*. The word "automatic" comes from the fact that clustering is an unsupervised learning method and not like the standard classification, where the method is supervised. Clustering has traditionally been used in areas such as Information Retrieval.

There are several different methods to do clustering. The most basic distinction is between *nonhierarchical* and *hierarchical* methods.

The simpler nonhierarchical methods divide the samples into different clusters and assign every sample to the cluster it bears most resemblance to.

The hierarchical methods are somewhat more complex. The idea is that each pair of samples or clusters are successively linked until the whole data set is connected. There are two ways of doing this linking, *agglomerative* and *divisive*. The agglomerative is a bottom-up method that starts with the single samples and successively connects the data, while the divisive is a top-down method that starts with the entire sample collection and progresses by dividing it into smaller and smaller clusters.

In order to separate the clusters from one another, there is a parameter that needs to be set. That's the *similarity threshold,* the factor that decides how similar the samples must be in order to be placed in the same cluster. This is the main factor in describing the "high degree of association" that was discussed above.

Intelligent Miner uses a nonhierarchical approach, determines how many clusters it needs by itself (with a maximum set by the user) and uses a method called *Condorcet's criterion* to determine which cluster a sample should be placed in. A detailed description of Condorcet's criterion is beyond the scope of this work. The interrested reader is referred to [Grabmeier et al, 1998].

3.5 BIDDING STRATEGIES

The problem that we want to solve with a bidding strategy is simply the following: Given a history of bids, what will the next bid be? We therefore want to *predict* the future value by looking at the previous values.

The field of bidding strategies can be seen as a special case of reasoning under uncertainty and the traditional theory to use when reasoning under uncertainty is statistics. In this section we describe a simple but fundamental algorithm from the field of statistics, the *linear least squares*.

3.5.1 REGRESSION AND LINEAR LEAST SQUARES

Within the field of statistics there is a sub field called *regression* and even more specificly, we will here discuss *one-dimensional regression*. Regression in general is used for finding connections between two or more quantities, for example between someone's intelligence quota and their income, or between cigarette consumption and life expectancy [Blom, 1989]. In our project we use regression to find connections between the previous auctions ($x_1 \ldots x_n$) and their bids ($y_1 \ldots y_n$). The most common method to solve such a problem is the *linear least squares* algorithm:

What we want to do is to place a straight line through *n* points, i.e curve fitting. This amounts to the same thing as solving a system of equations with *n* equations and two unknowns, which generally is unsolvable. However, we can still do an approximation:

The idea of the linear least square algorithm is to approximate a set of (x_i, y_i) points with a straight line and at the same time minimize the deviance from the y_i points to the line. More formally:

We want to find an equation

$y = \alpha + \beta x$ that approximates the set of $\{(x_i, y_i)\}$ points. This means that we want to minimize the function $\sum_{i=1}^{n} (y_i - \beta x_i - \alpha)^2$. The function is then solved by standard linear algebra techniques. For a more complete discussion of the linear least squares method, see [Hackman, 1996].

We are not limited to just linear functions with the least squares method. We can also use higher polynomials for curve fitting. In our testing in Chapter 6 we will also use two-dimensional curve fitting.

Chapter 4 OBJECTIVES

In this chapter the specific objectives of this work are presented in the form of goals and delimitations. The goals are stated in section 4.1 and the delimitations in section 4.2.

4.1 GOALS

We will propose an abstract system design solution for handling MRO procurement from multiple electronic auctions. The focus of this solution will be on generality:

- First and foremost the proposed solution must be coherent with other parts of the Movex System, both existing and future.
- Further, the agent model used within the proposed solution should be general enough to allow for a wide range of future functionality, as well as effectively support the specific functionality needed for this domain.
- Agent communication should be based on a standard solution, to achieve a high level of generality, and allow for future expansion and integration with other agent systems in the domain.

The proposed system design will serve as a context for discussions of different artificial intelligence techniques needed in order to realize the desired functionality:

- In order to achieve the negotiation capability of the Buy Agent, we will test different bidding algorithms, to see which performs best in the steel domain.
- To achieve the advice capability of the Advice Agent, we will test the clustering, neural network and the decision tree functionality in the data mining tool Intelligent Miner from IBM. We want to see if they can be of use in helping the user decide on what product they should choose.
- When it comes to the functionality needed in the Compare Agent, we will investigate which auction we should choose to bid on when there are multiple auctions going on at the same time.

4.2 DELIMITATIONS

- The specific functionality of the find process (see sections 1.1.2 and 2.1) is not discussed in any length. The issue was discussed in section 3.2, and the conclusion is that the problems are beyond the scope of this work. The first approach (screenscraping) discussed is not likely to remain relevant in the future, while the second (using protocols) is highly interesting but an entire research field in itself and therefore cannot be covered within the limits of this work.
- The issues concerning XML-based TPA:s and their relation to KQML message content, discussed in section 3.1.3, is closely related to the find process (see section 3.2.2). Consequently, this aspect of agent communication is delimited out from of this work.
- Mobility of agents is not considered. This was an initial delimitation given by Intentia. Other active projects within the company are exploiting this aspect of agent technology, independent from this work.

- Multiple auctions where the auctions use different auction protocols, i.e. heterogeneous auctions, are not discussed. This is indeed a very interesting area and something that needs to be addressed if a complete system is to be constructed, but it is not possible for us to address that issue within this work due to the time limitations.

Chapter 5 SYSTEM DESIGN

This chapter gives an overview of the system design proposed. The use case given by Intentia is translated into an agent context in section 5.1. The system design is described in section 5.2. The distribution of system functionality between different agents is described in section 5.3 through 5.6 and the interaction between the agents is presented in 5.7. Further elaborations are done in the subsequent sections.

5.1 TRANSLATION OF USE-CASE

As a first step in designing an agent system with the desired functionality, that functionality had to be distributed over a number of agents. We chose to do this in a very straightforward manner, and defined one agent for each step in the use-case initially described in section 1.1.2 and modified in section 2.1 above. This distribution of functionality between different agents was to some extent considered (albeit not explicit) when the modifications to the original use-case were made.

Thus, we ended up with four different types of agents: a **FIND** Agent, a **BUY** Agent, a **COMPARE** Agent, and an **ADVICE** Agent. The "negotiate" process in the original use-case was renamed "buy", since this is more suitable in the auction context. Its functionality is not altered in any way. The functionality of the Find Agent isn't discussed at any length in this work.

These agents are constructed after the model described in section 3.1.2 above, namely as an agent core common to all the different agent types, and functional plugs that provides the different types of functionality needed in each agent type. For example, the Advice Agent uses a special "advice plug", providing the capabilites described in section 5.6 below.

Aside from the "straightforwardness" of the design choice, the one-to-one mapping from the use-case has very significant advantages in itself. Dividing the functionality between four different agents, each of which corresponds to a step in the standard use-case, provides for a strong relation between the abstract view of the MRO procurement process and the actual implementation of it. This relation is an obvious advantage when integrating the functionality with other parts of the system (i.e. a Movex implementation).

5.2 SYSTEM DESIGN OVERVIEW

An overview of the basic design of the system is depicted in Figure 12 below. It shows how the environment is divided into two parts, the seller side and the buyer side, corresponding to the buyer and seller roles of the use-case of section 1.1.2. The two sides aren't physical entities in the design model, merely a way to group the agents logically as described below. Consequently, the fact that two agents exist on the same side in the model does not necessarily mean that they execute on the same physical server, the same network, or even within the same country.

The seller side contains all available vendors, represented by their respective auction sites. Note that the term "site" here doesn't necessary have to be a traditional web site; in the future, it isn't even very likely that this is the case. In this context, a "site" is merely a place where agents (human or artificial) can retrieve information about the auctions offered by this vendor. Depending on the specific underlying technology used for implementation, such a place may look very differently. For example, when using a component-based technique like CORBA [CORBA, 2000] or Enterprise Java Beans (EJB) [EJB, 2000], it would be a business transaction server. The latter has been explored in [Blixt & Öberg, 2000].

Further, a "vendor" site doesn't necessarily have to correspond to a specific vendor, it might as well be a place where several different vendors offer their products to a community of buyers. Such sites are often termed "marketplaces" in the e-commerce context.

From our perspective, only one property of a site is interesting: it exposes its information to non-human agents in some way fully understandable to them (see the discussion on XML-based TPA:s in sections 3.1.3 and 3.2.2). The agents in the system communicating directly with the vendor sites, the Find and Buy Agents (denoted by F_i and B_i respectively in the picture below) are viewed as interfacing the seller side. The two other agent types, the Advice and Compare Agents (denoted A and C, respectively) are viewed as belonging to the buyer side.

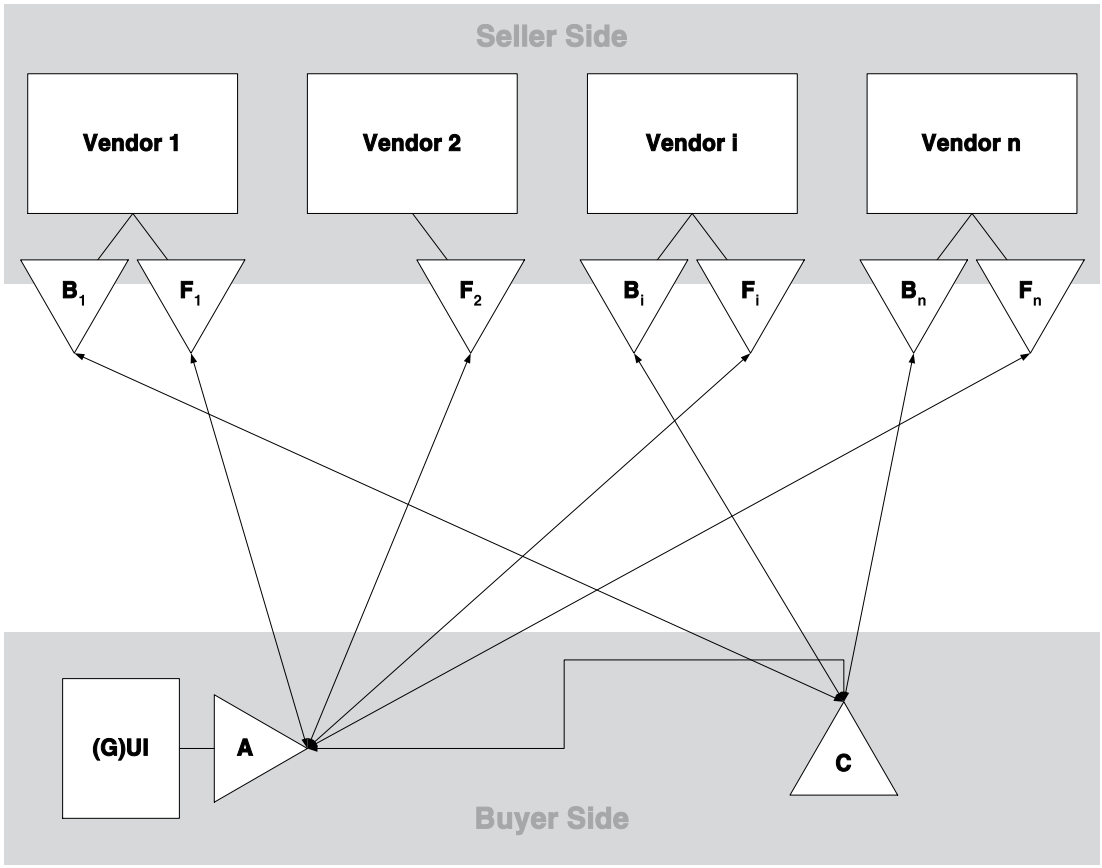

Figure 12. System design overview, showing the system in a typical state.

The buyer side consists simply of the end-user of the system, and the agents that this end-user utilizes in performing his tasks. In the typical case, the buyer side would consist of a user, an employee of a potentially large company, and his PC or workstation, executing the buyer side agents. However, this machine could also be a distributed solution, with the different agents executing on different physical machines.

Separating the two sides, we have some kind of network capable of carrying the communication between the agents. No further assumptions about this network are made; it may be an open network like the Internet, or a closed one, like a corporate intranet or an extranet connecting several business partners. The underlying protocols used are also of less interest, as long as they provide means for transporting KQML messages. As of today, the simplest way to do this over existing networks is probably by the use of the Hypertext Transfer Protocol (HTTP) over a TCP/IP connection, i.e. by sending the messages as ordinary text strings.

5.3 THE FIND AGENT

Although the functionality of the Find Agent isn't discussed at any length in this work, its basic behavior is vital to the system design.

As soon as a vendor site, say i, is registered within the environment, a corresponding Find Agent, denoted F_i above, is started. The functionality of this agent is fairly straightforward: it receives KQML messages from the Advice Agent and replies to these messages as soon as it knows the content of the reply. The agent uses its knowledge about how information is represented (see section 3.1.3 and 3.2 above) at the site to translate the requests to appropriate queries for the vendor site, and then translates the resulting responses into an understandable KQML message reply.

The Find Agent responds to two different kinds of requests:

Find auctions of items matching the product descriptor of the request.

Monitor the site for future auctions of items matching the product descriptor of the request.

A monitoring request is very similar to a find request; the only difference is that no negative reply is sent if the no currently available auction matched the request. Instead the reply is delayed until a match is found, in which case a positive reply is sent, or until the request expires and a negative reply results.

Thus, the Find Agent is **not autonomous**. It acts only on behalf of the requesting agent. However, since monitoring requests may result in quite long-term commitments, the behavior can be perceived, to some extent, as **pro-active**.

5.4 THE BUY AGENT

The field of work for the Buy Agent is one specific auction – one specific product. A Buy Agent has two basic tasks, namely to forecast how the monitored auction will evolve in the near future, and to collect data about the auction for use in the system.

To fulfill its first task, the Buy Agent must be capable of analyzing the current state as well as the bidding history of the auction. The amount of information available to the Buy Agent may vary from one auction to another, as may the usefulness of it; some auctions may not expose the association between a bid and the bidder to the agent while others will, for example. The agent uses the available information to make reasonable predictions about how the auction is developing, with the intent of placing a competitive bid itself at some point. To decide whether a bid is competitive or not, the Buy Agent communicates all proposed bids to its associated Compare Agent (see Figure 13 below), who decides which bids to issue.

The Buy Agent is strongly connected with its Compare Agent, which can be viewed as its "parent" in some way. First and foremost, the Buy Agent is started by a Compare Agent – probably together with a number of other Buy Agents. Second, the Compare Agent with which the Buy Agent is associated has quite a high level of control over the actions of the latter. This is due to the nature of its task: since an auction bid is in some sense legally binding once it's placed, all bids must be carefully considered or the user or company representing the buyer side may end up paying a lot of money. The Buy Agent isn't able to make these considerations on its own, since it doesn't have knowledge about the world outside its own auction.

A Buy Agent is started and connects to a specific site, when the system finds an auction hosted by this site in which it wants to participate. The lifecycle ends as soon as the auction is closed. During its entire lifetime, the Buy Agent may communicate the information it collects about the auction to other parts of the system. This information may be used for example by the Compare Agent to improve its decisions by taking history into account. In the proposed system, all communication with Buy Agents goes through Compare Agents. This is no requirement, however, direct communication between all agents is possible.

5.5 THE COMPARE AGENT

A Compare Agent corresponds to one procurement process, i.e. the task of buying one particular product. That is, its life cycle spans from the start of the first auction involved to the closing of the last, or until the desired product has been obtained and the procurement task accomplished. Several of these processes may be active at the same time, see section 5.8 below. The Compare Agent communicates primarily with its associated Buy Agents, and decides which of these is allowed to place a bid.

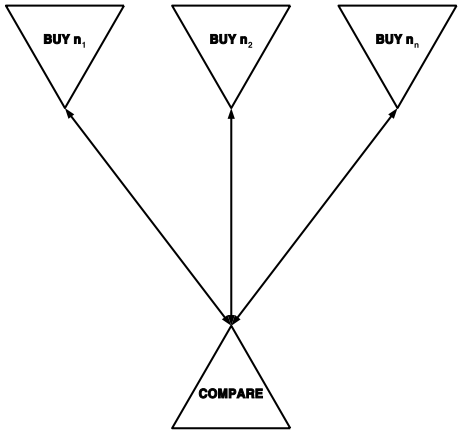

Figure 13. A Compare/Buy Agent combination, showing a Compare Agent with n associated Buy Agents, i.e. handling n concurrent auctions.

It also communicates with the Advice Agent, reporting the results and status of the ongoing procurement task so that this information can be presented to the user if desired. Via the Advice Agent, the Compare Agent can also get user confirmation on specific decisions before making them.

In its simplest form, the Compare Agent can be seen as just a comparison function of the bids proposed by the different Buy Agents.

5.6 THE ADVICE AGENT

The Advice Agent is the only part of the system visible to the user, and also the only of the agent types that's active at all times, i.e. its life cycle is the same as for the system as a whole. It's responsible for all interaction with the user, which means that it has to provide some kind of user interface, most likely one based on graphics (GUI). The exact design of the interaction may vary depending on the user interface, but there always has to be some way for the user to input descriptions of the product he wants to buy, and some way for the system to notify the user and provide feedback. Using a GUI in its simplest form, this would be done by the use of standard forms and notification requesters.

The Advice Agent communicates with Find Agents, that provide information on which auctions and products are available, and with Compare Agents that provide feedback from the current bidding processes. It can be seen as the hub of the system, since it's responsible for starting new agents of the other types (Buy Agents indirectly) when needed, and it's also the one that translates the users input into search and buy orders which are then issued to the Find and Compare Agents, respectively.

The main characteristic of the Advice Agent is that it's not only reacting to the user input through the user interface, it also acts **pro-actively** in trying to support the user in his task. To be able to do this the Advice Agent needs to record the behavior of the user and forecast what the user is likely to do next, based on history. Combined with the monitoring functionality of the Find Agents, this makes it possible for the Advice Agent to notify a user when a product which he has shown interest in before becomes available at an auction, for example. For simple forms of this functionality, keeping a simple history list would suffice, while more sophisticated forms raise the need for more intelligent mechanisms.

Aside from the pro-active support functionality, the Advice Agent has one other very important task, quite crucial for the functionality of a multi-auction system: it has to be able to decide whether two products are "alike", in some respect, from their descriptions as they are collected and communicated by the Find Agents. When the system tries to find multiple offers on a specific product, it's very important that it's capable of looking beyond the differences in the product descriptions that is likely to appear between different vendor sites. This is especially true for some types of products, such as steel. Two perfectly comparable steel types may be described differently, simply because they have different manufacturers, for example.

Thus, without this capability of the Advice Agent, the system would probably never find more than one auction of a specific product, and therefore be of very limited use. The problem of identifying different product descriptors as describing the same product (or at least a comparable one) is nontrivial. Different possible methods are discussed in Chapter 7.

5.7 AGENT INTERACTION

Below, we further describe the functionality of the proposed solution in terms of a typical example scenario.

5.7.1 A SIMPLE SCENARIO

This scenario describes how the agents cooperate to achieve a pro-active behavior and multi-auction bidding functionality:

An employee at a company building bridges, say the Öresundsbron consortium, uses the system to buy steel suitable for offshore constructions (e.g. bridges). He has done several such procurements in the past, and based on information collected about these prior procurements the system knows that the user is interested in steels with this specific area of application. It now detects that a steel with offshore construction as application area is being auctioned at one of the vendor sites.

The system notifies the user of this fact, and the user confirms that this is indeed a steel suitable for his purposes, and that he wants to buy it. He also instructs the system to wait for a certain amount of time in order to see if there exist other auctions on similar steels in an effort to get a better deal, and he also dictates the maximum price he's willing to pay. Eventually, the system finds another such auction, and then initiates automatic bidding at the two auctions. Hopefully it then manages to achieve the desired steel within the stipulated price range at one of the auctions. Finally, the user is notified of the result of the procurement process.

An alternative, non pro-active example scenario would be that the user initiates the search for the desired steel himself, and that the system then proceeds to bidding for this product at the suitable auction(s) found in the same way as above.

5.7.2 THE INTERACTION STEP BY STEP

Figure 14 below shows how the agent interaction proceeds for the example scenario of the last section.

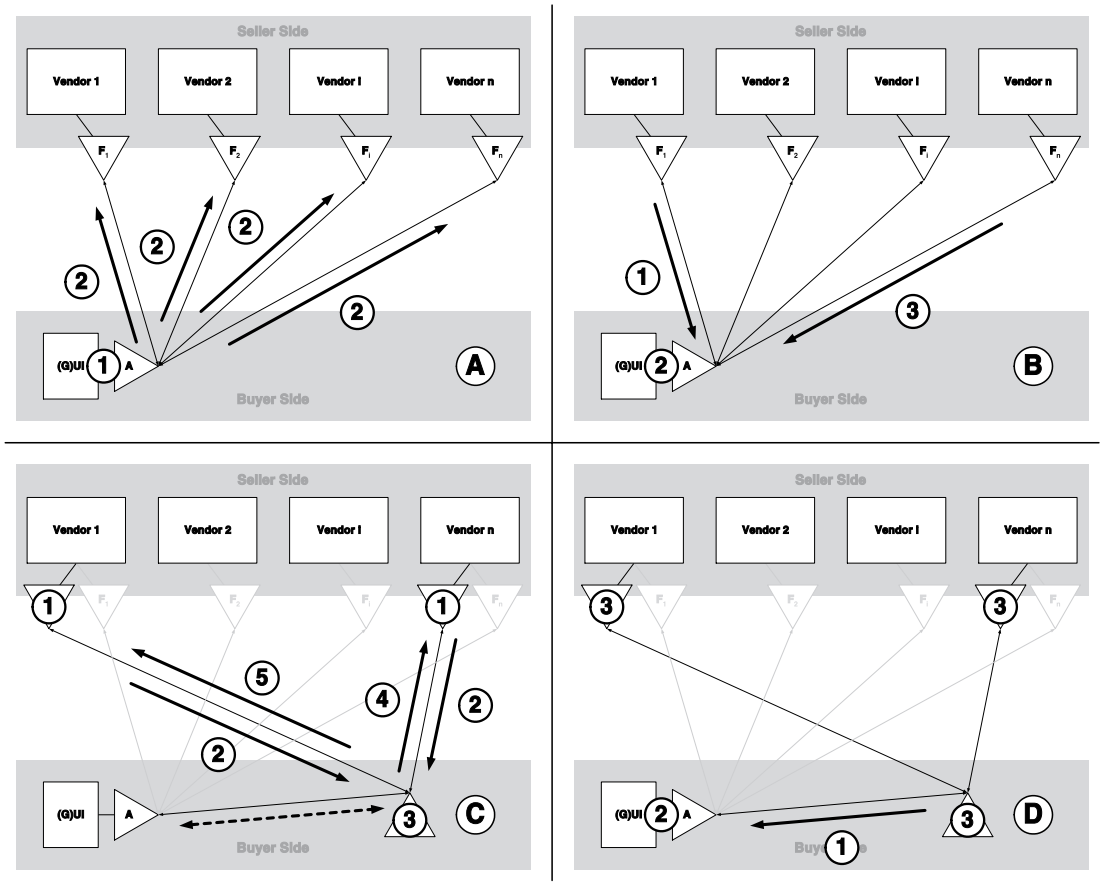

Figure 14. Sequential states of the system during the procurement process.

Figure 14a shows how the Advice Agent (1) issues a request (2) to all the Find Agents to search for auctions on steels matching the offshore application area. These requests might be initiated by the system itself (pro-active behavior) or result from an explicit search request from the user, as described above. The time between the search request and the response from the Find Agents may be very varying, as discussed in section 5.3. (The Advice Agent might have a request pending for an infinite period of time, i.e. when the system doesn't find a product matching what the system believes is interesting to the user. To handle these cases, we need to set an expiration date for each request, to prevent the Find Agents from getting overloaded with old requests.) The important point is that all responses from Find Agents originate from a request issued earlier. A Find Agent never initiates actions on its own.

Figure 14b shows how one of the Find Agents responds to the request issued (1), i.e. that it has found a suitable steel for offshore use. When the Advice Agent receives this response, it notifies the user through its user interface (2), and initiates a dialog. The user confirms that he indeed is interested in buying the steel found, and also enters the highest price he is willing to pay. In some cases, he might also supply other restrictions, such as the latest acceptable time of delivery. Finally he also enters the period of time that the system should wait for other suitable auctions before starting to bid. This period should not extend past the starting time of the auction already found, or that item could be lost without even submitting any bids.

The Advice Agent then sits idle until it receives responses from other Find Agents (3) or until the stipulated period of time expires, and then it proceeds to the next step.

Figure 14c shows the state of the system when the automatic bidding process has been initiated. Buy Agents have been started for each of the two auctions found, as well as a coordinating Compare Agent, forming a Compare/Buy combination as shown in Figure 13 above. The Buy Agents start to collect information about their auctions, i.e. which bids are submitted etc. They apply their bidding algorithm to this information and calculate which their next bid would be, the bid that they consider optimal for eventually getting the steel at the best possible price (1). This information is communicated to the Compare Agent continuously (2) as the bidding progresses. The communication can be viewed as a bidding request: the Buy Agent communicates which bid it wants to place next, and asks for permission to place this bid. The Compare Agent compares the two different requests (3) and its response to the Buy Agents will be either "yes, place your bid" or "no, hold". At any given time, no more than one Buy Agent is allowed to have the highest bid – we don't want to end up with more steel than requested by the user. This is the sole responsibility of the Compare Agent.

In this case the responses numbered (4) and (5) mean "yes" and "no" respectively, i.e. the Buy Agent at the vendor site labeled "n" has proposed the lower bid of the two and is allowed to place this bid, while the other Buy Agent is placed on hold.

Steps (1) through (5) are of course repeated as many times as needed while the auctions are active. Each time a new bid is placed at one of the auctions, its Buy Agent calculates a new bid and asks the Compare Agent for permission to place this bid. If the maximum price set by the user is exceeded, the Compare Agent communicates this to the Advice Agent so that the user can be notified of this fact (the dashed line). This could be done in one of two ways: either the bidding process is immediately terminated and the user simply notified that the steel couldn't be obtained, or the bidding process could remain active but on hold and the user asked whether he would consider adjusting his maximum price so that the automatic bidding can continue.

Figure 14d shows how the Buy Agent of vendor site n has placed the winning bid at its auction. Since we assume that a bid placed is legally binding, the deal is thereby closed. The Compare Agent communicates information about the deal, such as price and time of delivery to the Advice Agent (1) which assembles this information and presents the user with an order confirmation for printing (2). After this, all Buy and Compare Agents are terminated (3) and the system returns to the state in Figure 14a.

5.8 HANDLING MULTIPLE PROCESSES

Since most online auctions stay open for quite a long time (typically from one or a few hours to several weeks) it's very likely that the system has to handle multiple requests of service from the user simultaneously. This is handled within the system by simply instantiating a new Compare Agent for each procurement task added. Each of the Compare Agents in turn activates the Buy Agents needed to participate in the auctions that is interesting for its procurement task, thus creating a Compare/Buy Agent combination as shown in Figure 13 above. In Figure 15 below a system with two such Compare/Buy combinations, i.e. with two concurrent procurment processes active, is shown.

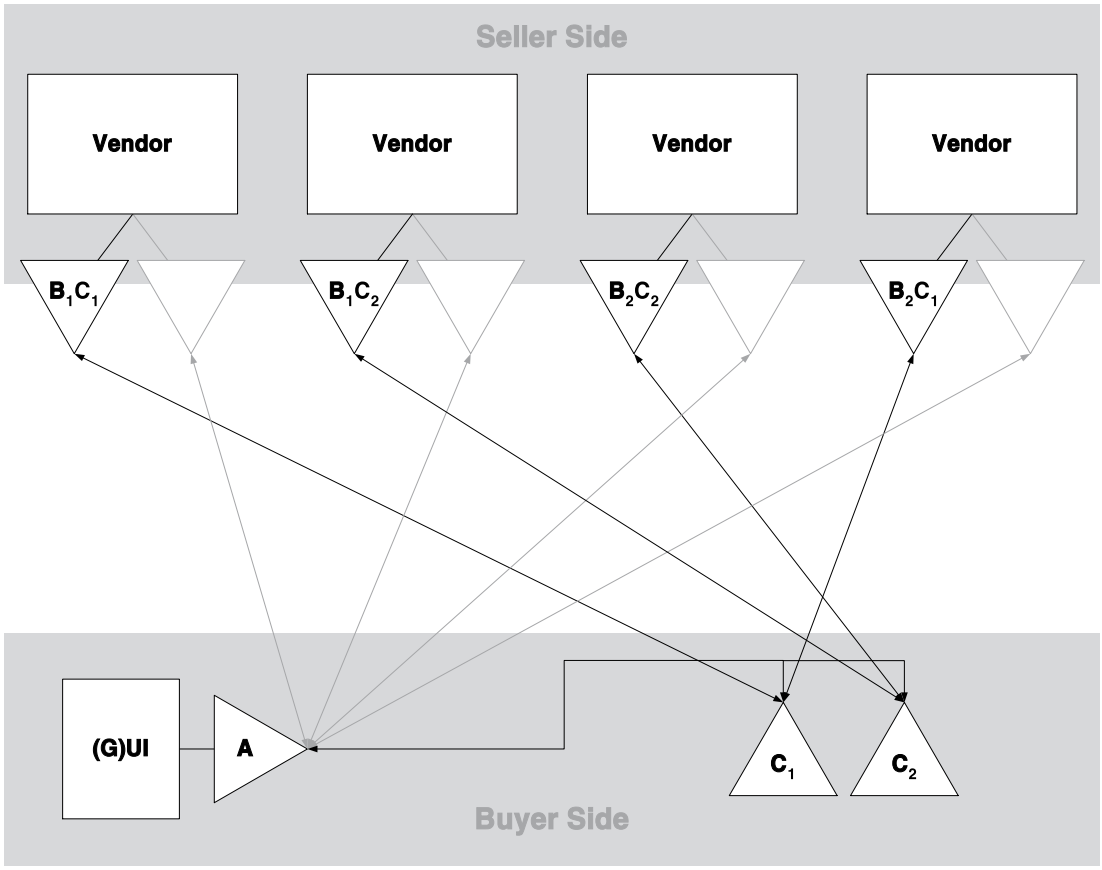

Figure 15. System design overview, showing two concurrent procurement processes with corresponding Compare Agents, C_1 and C_2. Connected to each of the Compare Agents are their respective Buy Agents.

5.9 INTEGRATION AND GENERALITY

The proposed design is very oriented towards integrating the auction process with other processes, future and existing, within the Movex ERP system. The reason for this is generality: since it's based on, and closely related to, the use-case for MRO Procurement (see section 1.1.2 and 2.1) there's no difference between regular procurement processes and auctions on a higher level. This provides transparency with respect to the procurement method used, and also makes it easier to integrate the different kinds of procurement methods.

Also, the use of a standard communication language instead of a special solution is important for integration reasons. The proposed design doesn't exploit very much of the expressiveness provided by the KQML language, and could easily have used a less complex message format. However, the use of KQML makes it possible to insert other, heterogeneous agents into the system, or to use the functionality of our agents for other purposes in future solutions. Section 3.1.3 discusses in more detail the use of KQML as a standard solution for agent communication.

5.10 A WORD ON MOBILITY

Although we do not consider mobility of agents in this work, a few comments on the design proposal from this perspective might be in place. The design supports mobility in one sense of the word, in that it doesn't assume anything about where the agent code is actually executed physically. On the other hand, it does assume that the functionality of the agents doesn't rely on the ability to move physically during their life cycle. In the proposed design, we assign one Find/Buy Agent to each and every vendor site involved in the procurement process. A different approach would be to let the Find and Buy agents be mobile entities, and to let them move between the vendor sites in order to collect the information needed, to place bids etc. This approach has been exploited by [Blixt & Öberg, 2000] in another agent technology research project within Intentia. That report shows that this approach, although very viable, presents serious drawbacks, most of which are related to security. The basic concern is that a mobile agent will execute at several servers owned by competing owners during the procurement process. The agent thus is vulnerable to manipulation by a dishonest vendor trying to take advantage by accessing and/or manipulating information collected about the bids of other vendors. A malicious host could also alter the agent code in order to gain advantage, or simply terminate the agent in order to sabotage the process. While these problems hopefully can be solved using standard encryption and authorization methods, the alternative approach with complete separation of the different vendor sites at the Find/Buy level is still equally interesting in our opinion.

5.11 SUBSEQUENT CHAPTERS

In the following chapters, Chapter 6 through Chapter 8, we will investigate the Buy Agent, the Advice Agent and the Compare Agent, respectively. All agents use a common agent core (as discussed in 3.1.2), but have different functional plugs attached to them. The tasks described in each chapter are to be added as one or more functional plugs to the *task knowledge and behavior* part of our CGA-inspired agent model, described in section 3.1.2.

Chapter 6 THE BUY AGENT

Here we discuss the Buy Agent, i.e. the agent that does the actual bidding at an individual vendor site. We investigate three rather simple bidding methods and see how they perform under test conditions.

In 6.1 we present the task of the agent. In 6.2 we state the basic assumptions for the environment that the bidding algorithms operate within. In 6.3 we present the three algorithms that we will work with. In 6.4 the data that we use is presented and finally in 6.5 the tests and the results are described.

6.1 TASK

In section 5.4 we were introduced to the functionality of the Buy Agent. The main task of the Buy Agent is to decide at what price it thinks a product will be sold, i.e. the *expected price* of the product. The rest of this chapter will be focused on that issue. As a side effect of the prediction procedure, we of course get the history of previous bids of the product we want to bid on. This information can be sent to the Advice Agent that presents it to the user and thus we have fulfilled the second task of the Buy Agent, i.e. to collect data to be used in the system as was stated in section 5.4.

The functionality needed in solving these tasks is then added as functional plugs to the Coarse-Grained Agent. The result is the Buy Agent.

We now turn our attention to the task of deciding at what price to bid. We first make some assumptions about the auction scenario.

6.2 BASIC ASSUMPTIONS

The following assumptions are made about the auction scenario:

1. We don't know how much money our competitors have.

2. We always try our best to get the product. We don't have time to "starve" our competitors out.

3. We compete in a Dutch auction context.

The first assumption might seem obvious, but in a general auction context it isn't known. The reason was explained in section 3.3.2.

The second assumption exists since an alternative would be to maximize the number of auctions won over a period of time. In such a scenario the agent could be dormant in some auctions and be very active in others. This is not compliant with the user's wishes since he wants to have the item at the current auction and does not want to take the risk that the agent doesn't bid in this auction.

The third and last assumption stems from the fact that we need to make a decision on which protocol to use. The bidding algorithms are different depending on what protocol they operate under. The Dutch auction protocol was chosen because we only need to determine the price at which to buy and nothing else. Since the protocol is equivalent to the Single Round Closed Bid auction protocol, we could just as easily have used that protocol.

These three assumptions form the basis for our bidding algorithms below.

6.3 BIDDING ALGORITHMS

Since evaluating bidding algorithms is something rather tricky due to the many parameters involved (e.g. how many competitors there are, what algorithms they use, what protocol to use), we felt that we couldn't test too many aspects of the bidding strategies. We instead focused on some basic questions and their solutions:

We wanted to see if normal linear regression with linear polynomials methods would be sufficient to predict the future price of a product at a specific auction, i.e. the expected price of the product. In other words we wanted to see if the next bid of a product could be predicted just from the history of previous auction bids. We also tried a regression method with square polynomials and a simple weighting scheme.

We describe the regression method in the next section.

6.3.1 THE REGRESSION METHOD

The linear regression method is also called the linear least square method and most of the readers will know it from statistics or linear algebra. Otherwise a brief introduction is available in section 3.5.1.

A DISCUSSION ON REGRESSION

This method is based on the assumption that the bids follow a trend. The algorithm is totally oblivious to the bidding algorithms of specific bidders, but concentrates on the trend issue. Either the price is going up or going down. The previous auctions are described as (x_i, y_i) points where x_i is the time of the i:th auction and y_i and is the price that the product was sold for at the i:th auction. In order to see what the future will bring, we associate a polynomial with the most recent points so that we can make a prediction of the next price. That prediction will be the expected price of the product at that auction.

The regression approach might seem a bit simple but that doesn't mean that it is uninteresting as we can see from a project definition from the Artificial Intelligence Laboratory at the University of Michigan [Hu, 1999]:

In this project we have created a configurable agent that can participate in certain auctions hosted on the Michigan AuctionBot using three different bidding strategies. We perform regression on the bidding histories of other agents and use this to predict a clearing price for the auction.

Thus, we see that regression is actually used for bidding strategies.

The reason why we don't try with a higher degree polynomial than two for the *curve fitting* is that the *Runge phenomenon* comes into play, i.e. if we adapt n or fewer points with an n-1 polynomial the variation between (and after the points) will be enormous. For more information about the Runge phenomenon, see [Eldén & Wittmeyer-Koch, 1996]. Such variations between the bids are not likely to occur in the auction domain and we are therefore satisfied with a simpler, but more stable algorithm.

6.3.2 WEIGHTING SCHEME

The regression method gives equal weight to all the points used as input for the prediction. This system does not work very well if the price changes very swiftly.

To accommodate this problem we used the following algorithm:

$y_{i+2} = \bar{y}_{i-1,i,i+1} + Diff(y_{i+1}, y_i) + DiffDiff(y_{i+1}, y_{i-1})$, where y_{i+2} is the value to be predicted, $\bar{y}_{i-1,i,i+1}$ is the weighted mean of the three previous values, $Diff(y_{i+1},y_i)$ is the difference between the two previous values and $DiffDiff(y_i,y_{i-1})$ is the difference of the difference. The $Diff(y_{i+1},y_i)$ is there to represent the speed of the increase (or decrease) of the function and $DiffDiff(y_i,y_{i-1})$ represents the acceleration.

The $\bar{y}_{i-1,i,i+1}$ mean looks like this: $\bar{y}_{i-1,i,i+1} = \alpha y_{i+1} + \beta y_i + \gamma y_{i-1}$ and $\alpha + \beta + \gamma = 1$. The last equation is there for the mean to be consistent.

The equation has the property that we can weigh the previous values differently. In the test in section 6.5.1 below we test different weights for the parameters.

6.3.3 COMPLICATIONS

We must of course realize that there is no guarantee that the history of bids will give a correct prediction on the value to come. There may exist other factors that affect the price since the last time it was auctioned, e.g. that the supply has suddenly vanished, and this would of course be impossible to see in a prediction.

Contextual information, like the supply factor mentioned above, are certainly not uninteresting, but it is beyond the scope of our work. Here we focus solely on the history of the bids in order to make a new prediction, and nothing else.

6.4 THE DATA

The bidding algorithms assume that there exist some data to work on from the beginning, at least in order to yield sensible predictions. The data we needed was at what price the product had been sold at in previous auctions. No information about steel auctions was available to us at the time of our thesis, since steel auctions still are uncommon. However, the price history of carbon steel and stainless steel CR304 (the most commonly used stainless steel) in the German market during the years 1980-1998 was available.

It is important to notice that the prices above are fixed market prices and not prices that have been determined in the dynamic price setting of an auction. However, Germany is considered the toughest market for steel and the competition is razor sharp. Therefore the prices are just as pressed as they would be in an auction context and therefore the data can be used to simulate auction behavior. The prices over the period 1980-1998 for both these two steel types are displayed in Figure 16 below.

Figure 16. Carbon and stainless steel prices in the german market, 1980-1998.

The prediction methods will be tested both on the erratic stainless steel and the more stable carbon steel.

6.5 TESTS AND RESULTS

In this section we describe the tests that were conducted and the results from these tests.

6.5.1 TESTS

LINEAR REGRESSION WITH LINEAR POLYNOMIALS

We tested linear regression with the following properties: The number of values that we used as input for the algorithm was set to five. The reason for this number of values was that it was sufficiently small to respond to sudden changes in the data and at the same time sufficiently large to level out the occasional extreme values. So the past five prices were used to predict the next price. In Figure 17 below we see a singular prediction for the linear case.

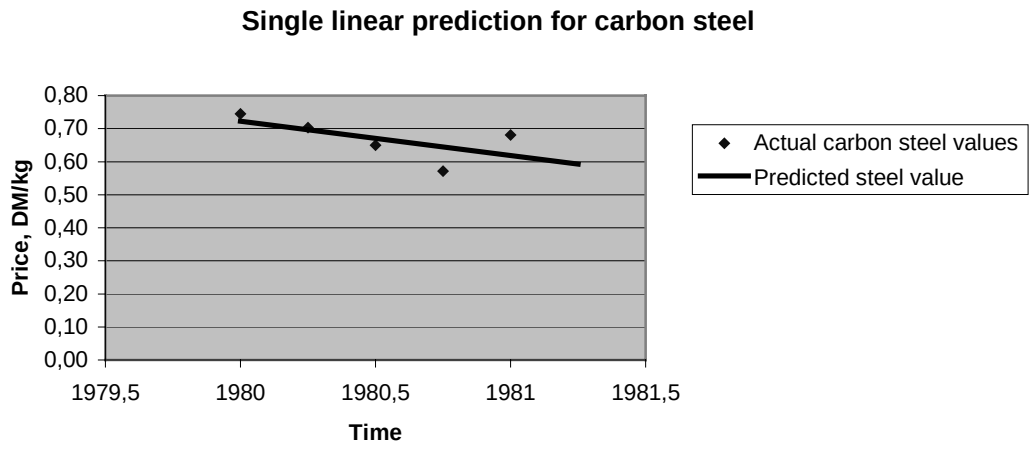

Figure 17. Linear regression with linear polynomials. Five values are used to predict the next one. The prediction for the second quarter of 1981 is as we can see 0,6 DM/kg.

LINEAR REGRESSION WITH SQUARE POLYNOMIALS

We tested linear regression with square polynomials with the same number of inputs as for the linear case, for the same reasons as described in the linear case and to make the two methods comparable. In Figure 18 below we see a singular prediction for the square polynomials case.

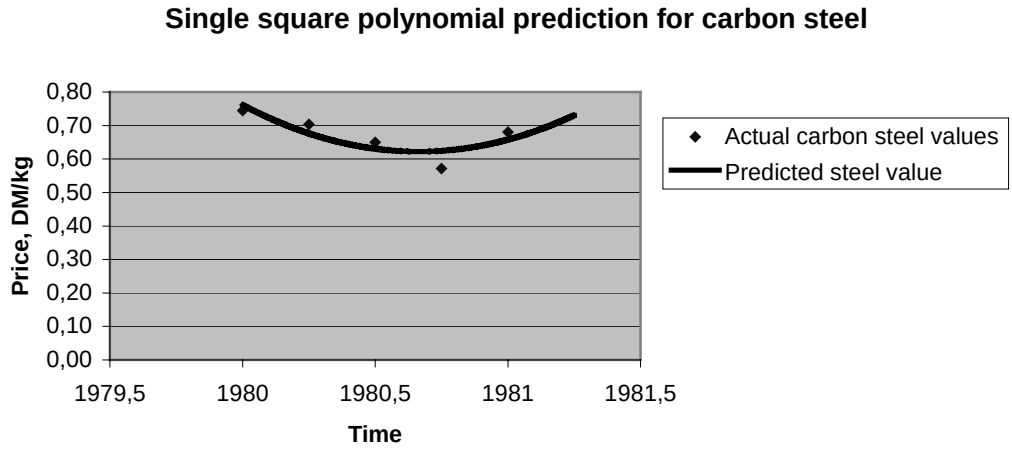

Figure 18. Linear regression with square polynomials. Five values are used to predict the next one. As we can see the prediction for the second quarter of 1981 is 0,72 DM/kg.

SIMPLE WEIGHTING

Finally, we also tested the algorithm described in 6.3.2. Here we tested different weights for the different parameters to see which weighting scheme produced the best results. In the results below we only present the weighting that produced the best results.

6.5.2 RESULTS FOR CARBON STEEL

The linear prediction for the carbon steel is shown in Figure 19 below.

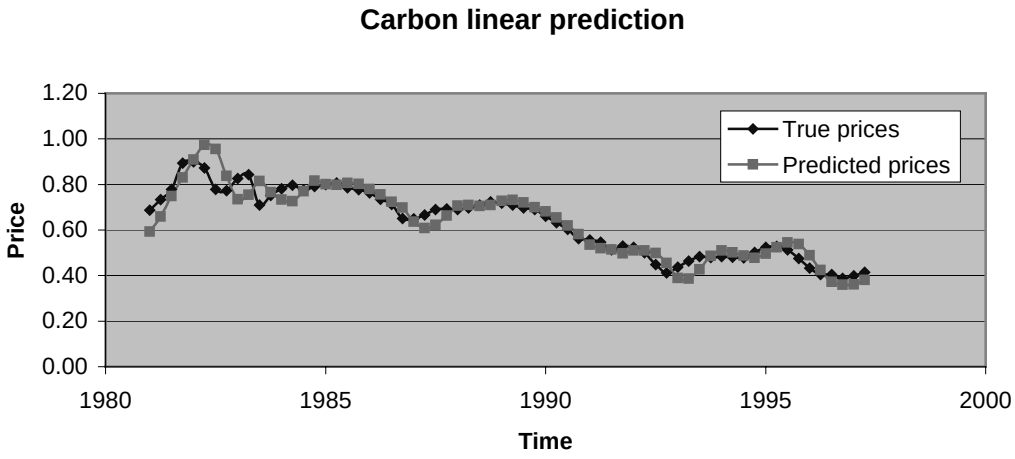

Figure 19. Linear prediction for carbon steel.

From the figure we see that the prediction works quite well except in the beginning. We can also see that the prediction lags behind the true prices. This is normal since changes in the true price will not have an effect on the predicted prices until the next prediction and it may take even longer for the value to effect the prediction, depending on how close the previous predictions were.

The square polynomials prediction for the carbon steel is shown in Figure 20 below.

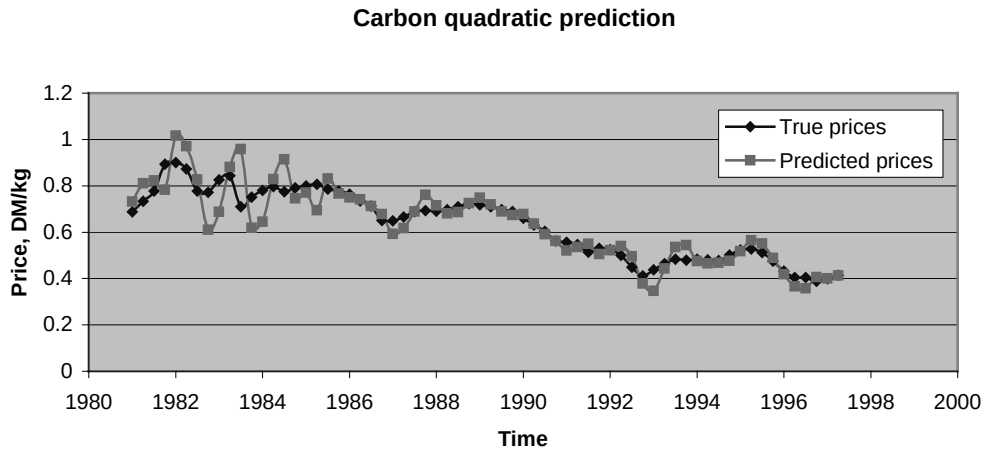

Figure 20. Square polynomials prediction for carbon steel.

From the figure we see that the square polynomials prediction has serious problems in the beginning when the price fluctuates but recovers after a while to more normal predictions.

The result for the weighting scheme is shown in Figure 21 below. We only show the best result in the figure.

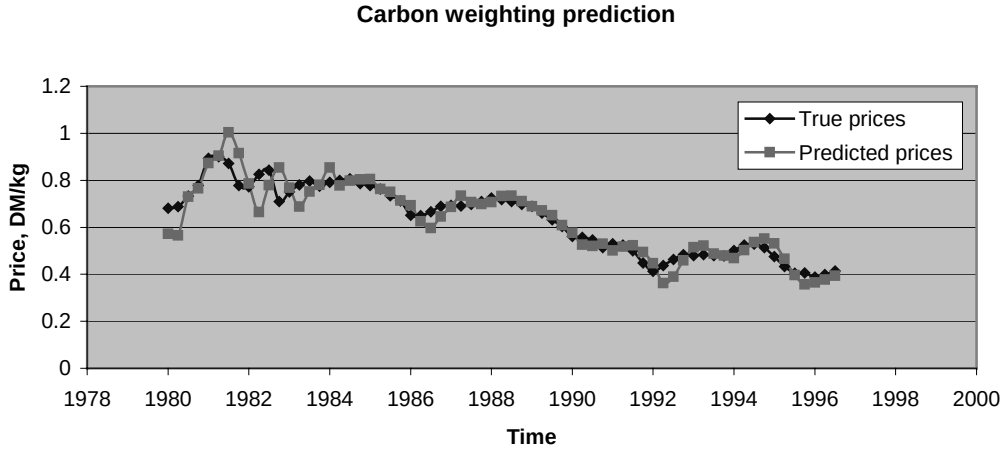

Figure 21. Weighting scheme. $\alpha = 1/3$, $\beta = 1/3$ and $\gamma = 1/3$.

As with the other approaches the weighting scheme has problems in the beginning but recovers after a while. The interesting thing here is that the best approach is to weigh the three most recent values equally high, disqualifying the assumption we made when constructing this method, i.e. that equal weighting of the recent values might be a bad idea.

In general the predictions come to rather good results for long periods of time but at periods when the price fluctuations is very heavy we get terrible predictions. The relative errors of the different methods are shown below.

ERROR RATES

To get a good measurement of how good the different methods are we checked the relative error of the predictions, i.e. $(truevalue - predictedvalue)/truevalue$.

The standard deviations for the different prediction methods on the sample domain are shown in Table 2:

Method	Standard deviation of the relative error for predictions
Linear regression with linear polynomials	7,5 %
Linear regression with square polynomials	9,3 %
Weighting scheme: $\alpha = 1/3$, $\beta = 1/3$, $\gamma = 1/3$	7,9 %

Table 2. Error rates for carbon steel.

The standard deviation for the relative error with the linear method was as we can see 7,5 % and for the square polynomials method it was 9,3 %. The meaning of these results is that the linear method in general missed the correct value with 7,5 % on average and the square polynomials method missed with 9,3 % on average. The results for the different methods are very similar, but the linear regression with linear polynomials method has slightly better results than the other methods on this data.

6.5.3 RESULTS FOR STAINLESS STEEL

We suspected already before the tests were made that the predictions would be worse when we tested the stainless steel than for carbon steel, since the variations in price are much more severe for stainless steel. As we will see, our suspicions turned out to be correct.

The linear prediction for stainless steel is shown in Figure 22 below.

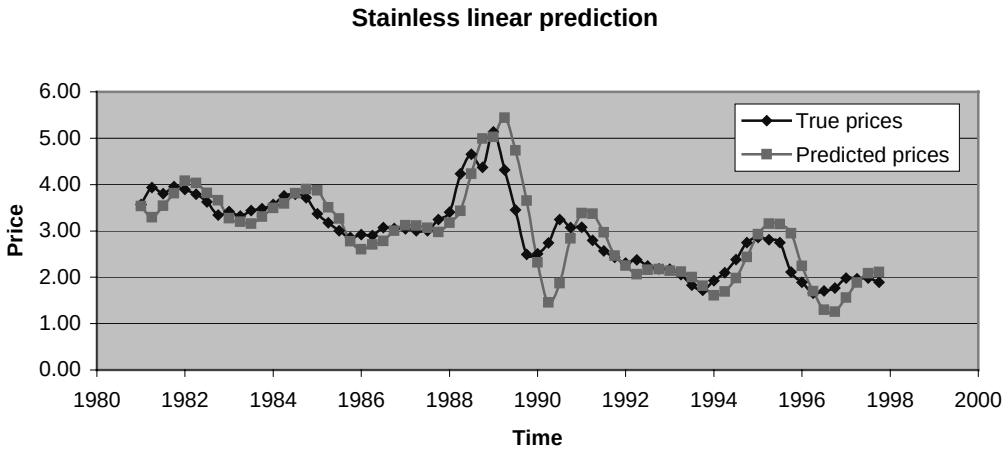

Figure 22. Linear prediction for stainless steel.

The prediction is rather good in the beginning, but when the prices soared in 1989 to 1990, the prediction is unusable.

The square polynomials prediction for stainless steel is shown in Figure 23 below.

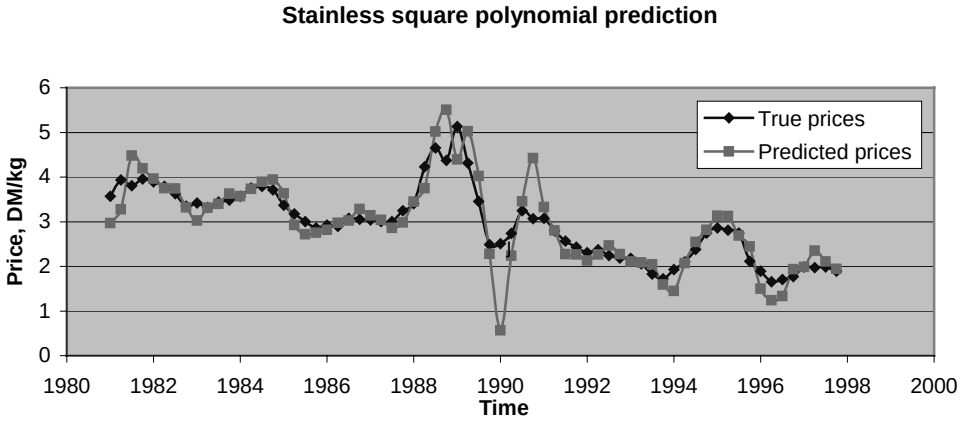

Figure 23. Square polynomials prediction for stainless steel.

The results are generally rather good, but in 1989-1990, the predictions are very poor. They are even worse than with the linear predictions.

The results for the weighting scheme are shown in Figure 24 below. As with the carbon steel we only show the result for the best weighting scheme in the figure.

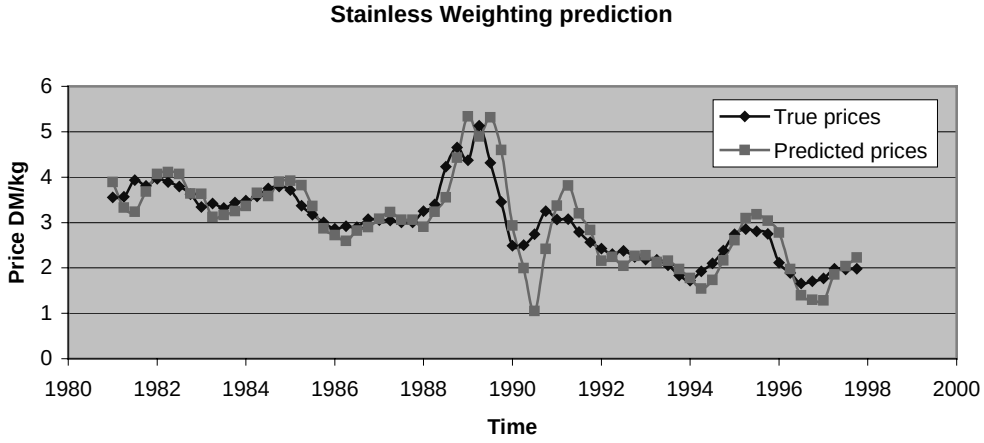

Figure 24. Weighting scheme prediction for stainless steel. $\alpha = 0,5$, $\beta = 0,3$, $\gamma = 0,2$.

The weighting scheme has results comparable to the other methods, and at the extreme points from 1989-1990, it does not stray from the true values as much as the square polynomials method tends to do. Worth noticing is that the weighting scheme with equal weights to the values produced the error rate 14,7 %, i.e. quite comparable with the other methods. This once again indicates that our assumption regarding this method might be false.

ERROR RATES

As with carbon steel we tested the relative error to see how good the predictions were. The standard deviation for the algorithms is shown in Table 3 below.

Method	Standard deviation for the relative error for predictions
Linear regression with linear polynomials	15,9 %
Linear regression with square polynomials	15,1 %
Weighting scheme: $\alpha = 0,5$, $\beta = 0,3$, $\gamma = 0,2$	14,4 %

Table 3. Error rates for stainless steel.

Again the results are very similar, but the weighting scheme $\alpha = 0,5$, $\beta = 0,3$, $\gamma = 0,2$ is slightly better than the rest.

6.5.4 MODIFIED TESTS

As we see, the predictions are quite bad when the market is changing rapidly. An alternative method is for the bidding algorithm to only make bids when the market is stable. This means that when a prediction has gone over a threshold value for the relative error (say 10 % in the carbon steel case), then the agent terminates and lets the user bid instead until the market has calmed down. We regard the market as calm when one prediction comes within 10 % of the true price. If we have such a threshold value the algorithm should perform better for the predictions that it actually does make.

CARBON STEEL

With a threshold value of 10 % for the carbon steel, we get the results in Table 4 below.

Method	Standard deviation for the relative error	Coverage
Linear prediction	5,8 %	80 %
Square polynomials prediction	8,9 %	77 %
$\alpha, \beta, \gamma = 1/3$	7,3 %	83 %

Table 4. Error rates for carbon predictions with 10 % threshold.

The “Coverage” column shows on how many percent of the cases that we made predictions. In the rest of the cases our predictions were too bad to be used.

The result is as we can see that all algorithms improve, and that the linear prediction improves the most. With 80 % coverage and an error rate of 5,8 % we have a suitable algorithm for at least some areas. A more detailed discussion about the conclusions that we can draw is presented in Chapter 9.

STAINLESS STEEL

With a threshold value of 10 % for the stainless steel, we get the following results in Table 5 below.

Method	Standard deviation for the relative error	Coverage
Linear prediction	12,5 %	64 %
Square polynomials prediction	15,7 %	71 %
$\alpha = 0,5, \beta = 0,3, \gamma = 0,2$	11,6 %	67 %
$\alpha, \beta, \gamma = 1/3$	10,4 %	62,5 %

Table 5. Error rates for stainless predictions with 10 % threshold.

The result for the stainless steel is that we can get down to almost 10 % relative error but that the coverage then goes down to only two thirds of the samples. This is probably not a result sufficient for use in a real application. Further conclusions from these tests are drawn in Chapter 9.

Chapter 7 THE ADVICE AGENT

This chapter will describe the Advice Agent and the tasks it fulfills. The primary aim will be to try to solve the user questions that are elaborated in 7.1. The section 7.2 is devoted to the domain we chose for our tests. Section 7.3 describes the testing that we made, and the conclusions that we could draw from the tests.

7.1 TASKS OF THE ADVICE AGENT

In the discussion in section 5.6 we concluded that there are principally three problems that the user needs help with when it comes to the product:

1. The user might not know what product to buy, i.e. he needs help with *product selection*.

2. The user does not know what price to pay for the product, i.e. *price setting*.

3. The user might not know that a product he likes becomes available, i.e. *pro-active advice*.

These first two problems are both in general user initiated, but the Advice Agent also has a pro-active part, i.e. the third problem. This means that when the user has worked with the system for a while, the system can give suggestions on what to buy based on the history of the user behavior.

The functionality described above is then added as three functional plugs to the task knowledge and behavior part of our CGA-inspired agent model from section 3.1.2. The result is the Advice Agent.

7.1.1 PRODUCT SELECTION

When it comes to product selection we have identified two major problems that the user might get into:

1. The user might not know exactly what product he needs.

2. The user knows what product he needs, and he wants to find it or if it is not available he wants to find a comparable product.

The general solution to these problems that we have chosen is data mining techniques. To this purpose we have used the product Intelligent Miner from IBM and the features *clustering*, *neural networks* and *decision trees*.

7.1.2 PRICE SETTING

The aim of the Buy Agent algorithm is to find a suitable price for the product chosen based on the purchase history of the product. This information is of course not only important for the Buy Agent but also for the user and we could easily distribute this knowledge to the user. In addition, we could also let the Find Agents that were described in 5.3 sit at non-auction vendor sites in order to get the retail price of the product and present this to the user.
From these two information sources, the user can then make his choice on what maximum price he wants to pay for the product in an auction.

Since this information is gathered from other sources and since no other computation is needed in the Advice Agent we will stop here and not investigate this subject further.

7.1.3 PRO-ACTIVE ADVICE

The user might not always know when a product becomes available and this raises the need for a pro-active behavior. The Advice Agent is able to see what products the user wants and look for them in auctions and tell the user if an interesting auction starts.

The Advice Agent can also use the same mechanism as in the section 7.1.1, i.e. product selection, in order to find products that are similar to the one that the user wants and alert him if any of those products are being auctioned.

This information is derived from the solutions to the problems that were discussed in section 7.1.1 and therefore we need not investigate this type of problem further.

7.2 THE STEEL DOMAIN

In this section we describe the steel domain and its specific problems. Furthermore, we discuss how the tasks the user wants help with (as stated in section 7.1 above) are represented in the steel domain.

7.2.1 A PRESENTATION OF THE STEEL DOMAIN

As was stated in section 2.4 we chose steel as the test domain. More specifically, we studied stainless steel from Avesta Sheffield. The data that we used, i.e. the steels, were taken from the Avesta Sheffield home page [Avesta, 2000]. It consisted of 29 samples with 11 application areas. Examples of the application areas were offshore industry, furnace parts and chemical plants. The parameters from the steels were Carbon (C), N, Cr, Ni, Mo, Name, Rp0.2, A5 and application area. Rp0.2 is the tenacity of the steel and A5 is the stretch of the steel, measured in percent. Other parameters exists but are left out because they are directly dependent on the other parameters. It is imperative that the parameters are independent of each other or else the feature dimensionality problem that we discussed in 3.4.2 will occur. There we also saw the risk of noise in the data.

What are the specifics for the steel domain? First of all the set of samples that we have available is very small. Depending on what kind of test we conduct it will be from 20 up to 30 samples. Each classification type has a maximum of three samples connected to it, which makes it a very hard classification. This of course also means that we can’t do the splitting into a training set and a testing set, but we need to use some other method as was explained in the background section 3.4.2

Second, the features available are not independent. On the contrary, some of the fields describe exactly the same property of the steel, although with a different scale. Hence we needed to sort out these attributes in order to get more correct results, as was stated in 3.4.2.

Third, a problem is that many of the steels have several application areas. For example, a steel could have both "offshore products" and "pulp bleaching" as application areas. One solution would be to create a class called "offshore products and pulp bleaching" and try to classify the steel in that category. The problem with this approach is that the classifier would see it as a wrongful classification if the steel were classified just as an "offshore product", while that actually is at least partially true.

We chose to solve this by removing the steel with multiple application areas. If we could find a classifier with simpler input we could focus on the issue of multiple classifications at a later stage. This left us with 20 samples and 7 application areas.

7.2.2 USER ISSUES

In section 7.1, we saw that one of the issues that the user needed help with was that he wasn't sure of what he wanted. This could in the steel domain be translated into the issue: *"The user wants to build a certain application"*. This means that the user does not know exactly what he wants but at least he knows what it must be able to handle. The application areas are sometimes available from the steel manufacturers, at least from Avesta Sheffield, so it that's not an impossible problem. The big problem is that not all steels are described with their application area. So a possible situation is that the user knows that he wants to build for example an offshore product, but the steels that are available at an auction don't have the "application area" parameters filled in. Then the Advice Agent could look through the features of the steels and decide what steels would be appropriate for the offshore product, based on its information about the features of other offshore products.

The other problem that we expected the user to want help with in section 7.1 was that he was sure what he wanted and needed to find that product or a substitute product if no such product is available. This could in the steel domain be translated into *"The user wants steel with the properties $\bar{x}$ "*. If such steel is available everything is ok and the system will initiate a buy process for this product. However, if the steel isn't available the system should give suggestions on similar steels that are available. The problem is that the system shouldn't just give the *nearest neighbor* (see section 3.4.3). The reason is that a steel lying closely to another in a nearest neighbor matrix with maybe only the carbon rate differentiating them, might still be used for very different purposes since the carbon rate determines the purpose rather strongly. The only solution that we could think of was to look at the application area for the steel with the properties $\bar{x}$ and find another steel with the same application area, i.e. we translate this problem to the first.

So in conclusion we only have one problem to address: *The user want to build a certain application*. In the next section we will present the tests we did and the results obtained

7.3 TESTS AND RESULTS

Here we describe the different experiments that done using Intelligent Miner. When we discussed the steel domain, we concluded that the user wanted help with just one type of question. In the following sections we will see how this scenario is tested together with the different mechanisms in Intelligent Miner.

Before the test started, it was necessary to determine how good the results from a classification method needed to be in order for it to be considered successful. We decided that at least 80 % of the classifications had to be correct and a maximum of 10 % of the classifications could be wrong. In our opinion, these figures stated the minimum requirements for this part of the system to be useful. A user has to be able to trust the agent's advice on nine occasions out of ten. Otherwise he won't use the advice.

7.3.1 TESTS WITH NEURAL NETWORKS

How would we use neural networks to help us give advice on which steel to bid on? Classification methods like neural networks and decision trees can make predictions on a variable based on other variables. In the steel domain we could try to determine a steel's application area from its parameter and that was exactly what we tried to do.

A few problems that were discussed in 3.4.4 and in 3.4.2 have to be addressed:

1. Dependent variables. In 7.2.1, we removed the dependent variables from the data.

2. Local minima. This is difficult to handle, but by using many epochs we minimize this risk.

3. Misclassification. In the domain we have more than two values for the variable to be classified. This means that we can't divide the misclassifications into false negatives and false positives. We assume, since no other information is available, that all misclassifications are equally bad and that we only determine the cost by the error rate measure.

4. Overspecialization. This will be determined partly by Intelligent Miner from its *internal error rate* estimation and by our *external leave-one-out error rate* estimation. Note the difference between these two error rates. Intelligent Miner determines the first one and we determine the second one.

5. Few samples. We'll test with the leave-one-out method.

FIRST SETTINGS

Here we describe the settings that were made in Intelligent Miner.

1. When constructing the classifier 80 % of the samples were used in order to train the model and 20 % was used to test the model. This was the division used to internally determine the error rate.

2. The accuracy of the model had to be at least 80 % before the training could stop; i.e. at least 80 % of the samples had to be classified correctly.

3. The error rate of the model had to be at most 10 % before the training could stop; i.e. at most 10 % of the samples could be misclassified.

4. The maximum number of epochs was set to 500.

All of these settings were default settings but they were also very reasonable in our domain. We need at least 80 % accuracy for the system to be useful. The most difficult thing to determine was the number of epochs. We reasoned that since Intelligent Miner is built to handle large amounts of data and that it then is supposed that 500 epochs should be sufficient, then 500 should also be sufficient for the small number of samples that we had in our test.

In this test we had 20 steels with 7 application areas (or categories), i.e. more or less three samples for each category. Since we tested with the leave-one-out method, 20 test rounds were made. 19 samples were used to train and the remaining one to test the model at each separate test round.

THE FIRST RESULTS

Since we had 20 steels, 20 tests were made where each test varied from the others by exactly one sample, so the tests should have produced comparable internal error rates. However, that was not the case. The internal estimated accuracy varied between 0 and 79 %. The internal estimated error rate with this setting varied between 0 and 16 % and the internal estimated unknown rate varied between 10,5 to 100 %. Such variations should not be the case with such little differences between the tests, indicating that the neural network had big problems deciding on a correct network.

Our leave-one-out method produced the following results: 10 % correct, 40 % incorrect and 50 % unknown. Clearly, that is totally unacceptable. The big difference even from the extreme values in the internally determined error rate regarding the accuracy and error rate measures may have the following explanation. In order to fulfill the accuracy level determined by us in the settings (80 %) the neural network needs to overspecialize on the samples to get the results and therefore the big differences occur.

MODIFIED SETTINGS AND RESULTS

We did not think that any alteration of the settings would produce sufficient improvement but we tried with one new setting:

The number of epochs was set to 1000. There is also a factor that makes the outlier samples gain more influence on the weighting process and we therefore increased the importance of that factor since we felt that the domain could have great variation. However, this had a negative effect on the convergence and the network didn't converge after 1000 iterations. We had to set the number of iterations to 3000 for it to converge and even then it failed to converge in one test.

The result was worse than in the previous test: 5% correct, 60% incorrect and 35% unknown for our leave-one-out method. The internal error rate produced values similar to the previous test. As a comparison pure guessing would have produced 1/7 correct samples, i.e. 14 %.

In conclusion, our test results were nowhere near the necessary requirements. Therefore, the neural network method was unsuitable, and further testing was abandoned.

7.3.2 TESTS WITH DECISION TREES

Decision trees are also a classification method and therefore the same argument for their usability applies as for the neural networks.

We also need to discuss the problems with classification and decision trees that were introduced in 3.4.2. All the issues discussed in relationship to neural networks are applicable here as well, with the exception of local minima. The solutions are the same here.

THE FIRST SETTINGS

- The maximum tree depth was set to unlimited. This wasn't a risky setting since the number of samples was very few.
- The maximum purity per node was set to 100 %, i.e. the splitting of the nodes is completed when each node contains just one class of samples.
- The minimum number of records per internal node was set to one, i.e. that the splitting of the nodes is completed if the internal node contains just one sample.

Just like in the neural network tests we had 20 samples with 7 categories.

THE FIRST RESULTS

The Intelligent Miner acted very strangely from the start with this input. It classified all samples into the same classification: e.g. all samples were classified as "offshore products". This was due to the pruning of the tree that did the pruning in a wrong way. After turning the pruning off, the algorithm at least produced a result.

The results were equally bad as with the other methods. The leave-one-out method resulted in 15 % of the predictions being correct, 80 % incorrect, and 5 % being unable to classify. This was clearly not a useful method. However, the internal error rate was zero so a possible cause was overspecialization of the input.

THE SECOND SETTINGS

We tried to remedy the overspecialization mentioned above by allowing slight pruning of the input, thereby making the tree a bit shallower and making the tree not so specialized to the input.

- The pruning power option was set to 0.1.

- The rest of the variables had the same values as in the first test.

THE SECOND RESULTS

The result was unfortunately even worse with the second settings. The internal error rate was about 20 %, which we felt was good. That meant that the method hadn't overspecialized on the input. However, the leave-one-out method gave the following results: 10 % correct, 85 % incorrect, and 5 % unknown. The testing thereby produced equally poor results as with the neural network method.

Again we had to conclude that the results were far below the stipulated requirements. Therefore, decision trees were considered a failure and further tests were abandoned.

7.3.3 TESTS WITH CLUSTERING

The clustering method is an unsupervised learning method that was described in 3.4.6. The method would be useful if we could partition the samples into groups where each group consisted of more or less only samples with the same application area. In this way, when new steel became available and the Advice Agent didn't know the application area it could decide what cluster it belonged to and determine the application area from that.

The tests with clustering were evaluated slightly differently than the above tests. The decision rule was that if the tests managed to get the samples with the same application area into the same cluster, then the tests could be deemed successful.

What we wanted was to gather the samples with the same application area into the same cluster. We chose the parameters that were discussed in 7.2, but without the application areas. This was because when the Advice Agent would use the model it would not have the application area available and therefore that parameter shouldn't be a part of the construction of the model either.

Again, the tests consisted of 20 samples with 7 application areas.

THE FIRST SETTINGS

- The maximum numbers of passes (i.e. the number of times all samples are computed and placed in a cluster) was set to ten. The default is two and the reason we increased it was at this stage in the tests we knew that the domain was hard to classify and could need that many passes.

- The maximum number of clusters was set to seven since we had seven application areas.

- The similarity threshold was initially set to 0.5, but that meant that almost all samples were put in the same cluster. That in turn of course meant that we could draw no conclusions so the first change was to increase the similarity threshold to 0.8. This at least had the positive effect that the samples were more evenly spread over the clusters.

THE FIRST RESULTS

The result was equally depressing as for neural networks. None of the clusters consisted of samples with just one application area. Three of the clusters consisted of two samples with the same application area, and the other four were totally mixed. This is dangerously close to what a stochastic division of the samples into seven clusters would give.

THE SECOND SETTINGS AND RESULTS

We altered the settings slightly to see if any improvements could be observed.

- The maximum clusters were set to nine, in order to see if some clusters could be split with good results.

The result was approximately as bad as with the first settings. Two clusters contained samples that all had the same application area, but in both those cases there were samples with the same application area that were assigned to other clusters as well. The rest of the clusters were more or less totally mixed.

OTHER SETTINGS AND RESULTS

We also tested with different weigh to the parameters, but in vain. The results were still roughly the same as in the tests above. Further testing was abandoned.

For a post study and conclusions about the tests, see Chapter 9.

Chapter 8 COMPARE AGENT

This chapter will discuss the Compare Agent. After the introduction in 8.1, the issue of homogeneous auction types will be discussed in 8.2.

8.1 INTRODUCTION

The Compare Agent actually has only one task: To decide which Buy Agent should be given permission to bid on its auction. This task is also not very complicated as long as the auction protocols are homogeneous, i.e. if all the auctions that participate use the same protocol. However, if the auction protocols are of different types, i.e. homogeneous, then the situation quickly becomes complicated, since the task involves several protocols with different characteristics.

The functionality described above is added as a functional plug in the task knowledge and behavior part of the Coarse-Grained Agent. The result is the Compare Agent.

In this chapter we make the assumption that the Compare Agent always knows which Buy Agent has the lowest expected end price at the moment. The Buy Agents solve this task. However, it is not always that the Compare Agent should bid at the auction with the lowest expected price, as can be seen in for example 8.2.2 below. So we must still decide **which auction to bid at**. Furthermore, we must also solve the task of deciding **when we will reevaluate a bid**, i.e. after which events it is necessary to determine which auction to bid at. There are several events that make reevaluation necessary and these events are different depending on the auction protocol.

These two tasks will be the focus of this chapter.

8.1.1 DEFINITIONS

First we introduce a concept called the *auction window*. These are the auctions that are considered when placing a new bid and the members of this window may change over the lifetime of a Compare Agent. This means that changes in the auction window might mean that we have to change our bid, depending on what protocol we use. Figure 25 shows how the process works.

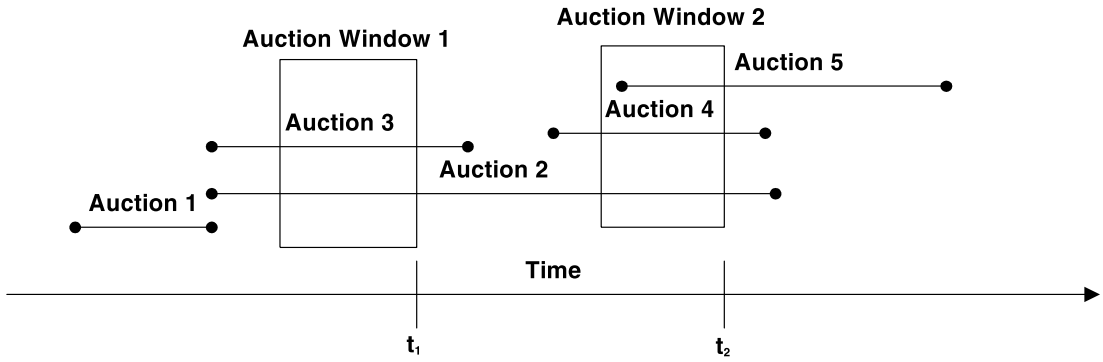

Figure 25. Auction window. The auctions are represented with a start and an end point. At t_1 the auction window consists of auction 2 and 3, while at t_2 the auction window consists of auction 2, 4 and 5.

The term *expected price* was defined in Chapter 6 and refers to the price the Buy Agent of an auction predicts that the product will be sold at.

We also make a total ordering of the expected prices of the different auctions. The auction with the lowest expected price is called A_1 and the rest of the auctions will be identified by A with letter subscripts.

8.1.2 BIDDING PRIORITIES

We must also state what our priorities for the methods below are. Our first priority is safety, i.e. we always want to have the opportunity to bid on the auction with the lowest expected price since that is the safest method to get the product that we want. However, as long as that is fulfilled we also want to try to bid at other auctions, to see if we could get the product at an earlier time or at a lower price.

8.2 HOMOGENEOUS AUCTION PROTOCOLS

When we assume that all auctions have the same auction protocol, the decision process is substantially easier than if they have different protocols. The main advantage is that it is very easy to decide when a new bid should be placed, since all auctions follow the same rules. We cover the Dutch, Single Round Closed Bid and the Open-cry auction protocols in the sections to come.

8.2.1 DUTCH AUCTIONS

For a brief presentation of the Dutch auction protocol we refer to 3.3.4.

WHICH AUCTION SHOULD WE BID AT?

This is very simple. We should always place our bid on the auction with the lowest expected price, i.e. A_1, but since the actual bid won't be placed until the price at that auction drops to the expected price, reevaluations of the bid play a major part in this protocol.

WHEN MUST WE REEVALUATE OUR BID?

We have identified three situations when this is necessary:

- When our current bid at A_1 has been topped or a collision (i.e. many bids at the same amount) between two or more bids has occurred at the A_1 auction.
- When a new auction becomes active within the auction window.
- When an auction other than the one we want to place our bid on drops to the same price or lower than the price we expect to get at the chosen auction A_1.

The first situation is of course rather clear, since this constitutes a fail situation at the A_1 auction. The second one is due to the fact that the new auction might very well be better than the ones we had to select from the last time we had to make a choice and then we naturally need to reevaluate our current bid. The reason for the third situation is that if the price for another auction, say A_i, drops to the expected price of A_1, then the other auction is a better choice (and of course the expected price of auction A_i, that was previously higher than A_1, is now lower or equal). We can see this situation in Figure 26 below.

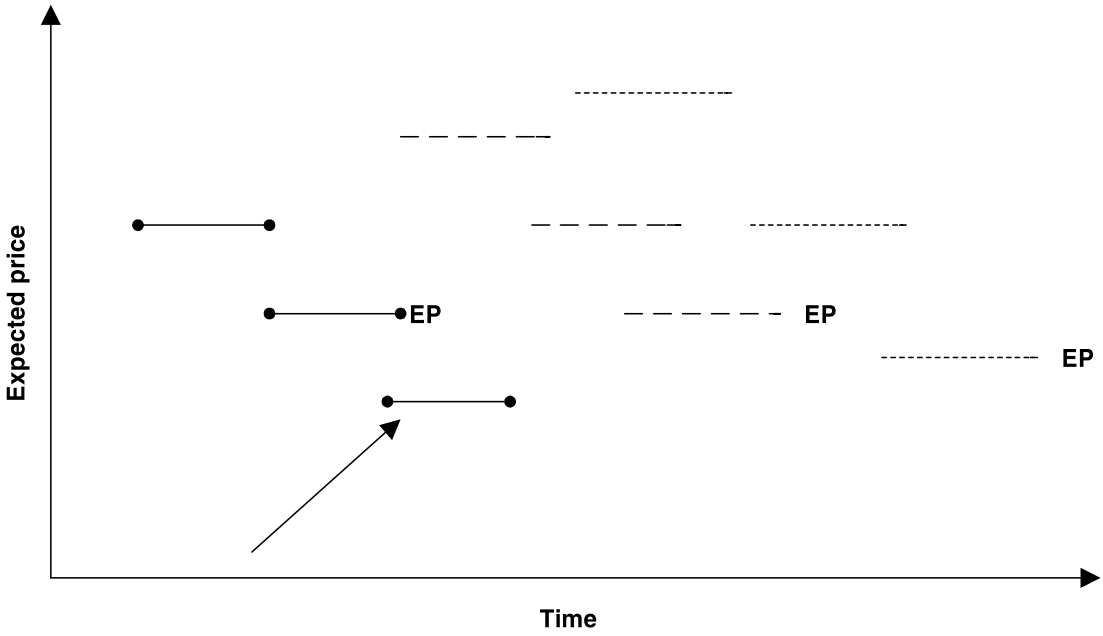

Figure 26. Dutch auctions. Each auction has a designated line representation and each line describes how long an auction stays at a specific price. The arrow points to the moment when the first auction has dropped below the lowest expected price of any auction and it therefore is the best choice for our bidding.

8.2.2 SINGLE ROUND CLOSED BID AUCTIONS

For a brief presentation of the Single Round Closed Bid auction protocol we refer to section 3.3.5.

WHICH AUCTION SHOULD WE BID AT?

We place our bid at the auction with the earliest expiration time, for example A_i, and at the expected price of the A_1 auction. This might seem a bit strange since the auctions held before A_1 all have higher expected prices and we shouldn't expect to win at these auctions. However, the expected prices of these auctions might be wrong and since we don't loose anything by placing a bid we will do so. The worst scenario is that we won't get the product, but then we can still make a bid at the A_1 auction later. The other case is that we get the product at the expected price, but earlier than we had expected.

Hence, the result with this method is that we can place bids at more auctions and maybe get the product a bit earlier. There are of course other methods but this is the best one that follows our priority as stated in 8.1.2.

One assumption must be made: The disclosure of the winner of the A_i auction must occur before the bid expiration time of the A_1 auction so that if we fail in the A_i auction we can still have a try at the A_1 auction.

This process is described below in Figure 27.

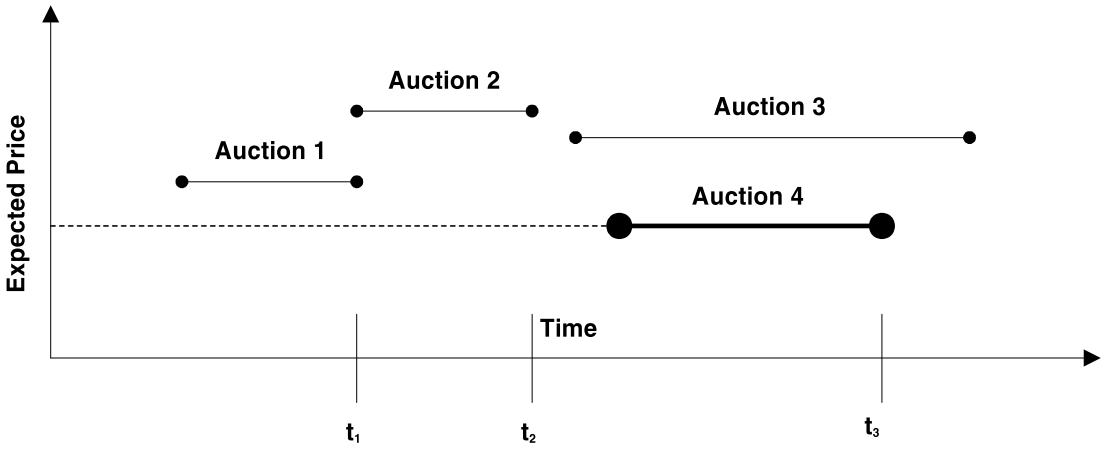

Figure 27. Single Round Closed Bid auction. The auctions are represented with a start and an end point. At t_1 a bid is made to Auction 1 at the price the dotted line prescribes. If we don't get the product then a new try is made at t_2 and then at t_3.

WHEN MUST BIDS BE REEVALUATED?

In section 3.3.5, we discussed the equality between the Dutch auction protocol and the Single Round Closed Bid protocol. This is true from a Buy Agent perspective but not from the Compare Agent perspective.

The difference is that in Dutch auctions we know immediately when we submit the bid if we have won, while in the Single Round Closed Bid auction it may take some time from the submission of the bid until the disclosure of the winner. This delay makes a difference in when the Compare Agent should place a new bid.

We saw above under which three circumstances a new bid must be evaluated when using the Dutch auction protocol. In the case of Single Round Closed Bid auctions we have the following circumstances when a new bid is to be evaluated:

- When our bid at A_1 has been topped.
- If a new auction becomes active within the auction window, and no submission of a bid to another auction has yet been made.

The first situation is natural since it is a fail situation. In the second situation, we need to have the guard "no submission of a bid to another auction has yet been made" since if a bid had been placed, there would be a penalty to remove it, if it is at all possible to withdraw the bid. At the auction site MetalSite, bids can only be revoked if the description of the product being sold has serious typographical errors, or if it's been changed [MetalSite, 2000]. Of course, we can't place two bids since that would make it possible for us to end up with twice as much of the product than we actually wanted.

8.2.3 OPEN-CRY AUCTIONS

For a brief presentation of the open-cry auction protocol we refer to 3.3.3.

WHICH AUCTION SHOULD WE BID AT?

In the open-cry auction we should of course bid at the auction with the lowest expected price, A_1, but if there are auctions before A_1 then we miss a lot of opportunities if we don't try to bid at those auctions. How can we provide a safe method for bidding at those auctions?

First of all we must realize that in open-cry auctions the leading bid can stand unchallenged right until the end of the auction and only then be beaten. This means that we could give a very low price at an auction A_k that starts before A_1 and this bid might remain unchallenged for a long time, during which the A_1 auction is completed, and in the end of the A_k auction our bid could be beaten. The result is that we didn't get any product since the bid in the A_k auction blocked the A_1. How do we avoid this situation?

The solution that we adopt is that we can only place bids at auctions that have end times before the start time of A_1. This is a very prohibitive method but in order for us to fulfill the safety priority stated in 8.1.2 we need to be that restrictive. This means that we can be sure that if we haven't got the product when the A_1 auction starts, we will bid at the A_1 auction.

In doing so we can ensure that we will be able to bid at A_1, but we might make an unexpected profit by bidding at the previous auctions.

WHEN MUST WE REEVALUATE OUR BID?

The open-cry auctions are easier in this aspect compared to the other types. There is actually only one situation in which we need to reconsider our bid:

When the bid at an auction has been topped.

The auction window doesn't play a part here, since even if an auction with a lower expected price would become active within the window, we could do nothing until our bid at the present auction has been beaten. If we made another bid at the more favorable auction we might get stuck with several products and that is not acceptable.

For conclusions we refer to Chapter 9.

Chapter 9 CONCLUSIONS

In this chapter we put together the conclusions that can be drawn from tests and discussions made in Chapter 5 through Chapter 8. In section 9.1 the conclusions on the design are made. In sections 9.2 through 9.4 conclusions on the Buy, Advice and Compare Agents are drawn.

9.1 THE PROPOSED DESIGN SOLUTION

As stated in the objectives (see section 4.1), the focus of the proposed design is on generality. By doing a one-to-one mapping from the use-case in section 2.1 to the multi-agent system proposed, we can easily integrate the solution with other parts of the Movex system based on the same use-case. This is important, since a system such as this is not very likely to be used as a stand-alone application, but rather as a part of a bigger system for handling MRO procurement.

Further, the agent model used (see section 3.1.2) is general, reusable for most types of agents for electronic commerce within Movex, and provides a simple conceptual view of software agents. The notion of functional plug makes it very suitable for object oriented and component based programming paradigms, which is a definitive advantage since Intentia bases their future solutions on these techniques.

It is imperative to notice that the agent design used in this work is not limited to the current test domain, i.e. steel, but is applicable to any form of procurement incorporating auctions.

Finally, we've found the KQML language very suitable for our needs; it provides the standardized platform for agent communication desired for future integration of this system with other agent solutions.

However, there are two things that should be pointed out in connection to this statement. First, the design we've proposed is of low granularity when it comes to the agent communication. We don't currently use many of the features provided by KQML. Therefore, it's uncertain at this time whether a further development of the design will really exploit it's full potential and benefit from this. One of the drawbacks of KQML that was described in section 3.1.3 above, namely the tendency that systems don't exploit KQML fully, may apply here.

Second, the suitability of KQML may be questionable, though still of course usable, when using XML as a basis for the message content language, as discussed in section 3.1.3. A solution based on both KQML and XML might prove unnecessary complex. Since XML solutions for electronic commerce is developing rapidly, this issue is important to consider in future work.

9.2 THE BUY AGENT

From the tests and the results that were made in 6.5 we can draw the following conclusions:

- For the carbon steel we could make predictions on the price for carbon steel with the linear method that worked in 80 % of the cases with an error rate of 5,8 %, i.e. the algorithm missed the true price with 5,8 % on average. It is quite possible that a human would have got better results, although not certain. However, even if a human would have got perfect results there are other aspects to be considered. How much does it cost to have a human perform these tasks? Are the alternative costs higher if we use humans? This is of course up to our contractors to decide but 5,8 % might very well be a figure good enough to consider using agent negotiators.

- When it comes to the stainless steel CR304 the situation is different. Even when we limited ourselves to only bid at two thirds of the cases we couldn't reach a better result than 10,4 % relative error rate. These poor results seem to be due to the fact that, as we have mentioned before, the price for stainless steel fluctuates more than carbon steel. The reason is that one of the main ingredients in CR304 (and many stainless steel's) is Nickel and when the prices of Nickel go up just a little bit, then the price of CR304 soars. We discuss this further in Chapter 10.

The result gives us reason to suggest that prediction algorithms like the ones that we have used can only be used in non-volatile domains, like carbon steel. A volatile domain is a domain affected heavily by external factors like the Nickel prices. If the prediction algorithms are used in more volatile domains like stainless steel, the fluctuations are so great that the predictions might strike very wrong.

9.3 THE ADVICE AGENT

After finding that the AI techniques tested in Chapter 7 were insufficient for our sample domain, we made a short investigation to see what could be the possible reasons for them. After discussions with Avesta steel expert Hans Nordberg [Nordberg, 2000] and Intelligent Miner expert Ulf Edholm [Edholm, 2000] we came to the following conclusions:

- The samples in the domain have many hidden dependencies between the parameters. This means that there are functions for describing the application area from the parameters, but these functions can't be found by the Intelligent Miner methods.

- The Avesta products are too diverse to have a good chance to be classified with good results. If we had chosen another steel company's products instead, e.g. Uddeholm products, we could have had a better chance. Of course, an investigation whether the Uddeholm or the Avesta products are the most representative for the entire steel domain must be made.

- The data set was too small to give any sufficient results. Hence, there were too few steels per application area too give decent results. This is rather a probable cause actually, but we must also realize that this is in all probability what the domain will appear. More steels will of course be added in a full-scale product, but there will also be more application areas so the number of steels per application area will in all likelihood be the same.

- The numbers of parameters was not sufficient to model the classification in a correct way. This is also a possible cause, but we couldn't get hold of more parameters.

In retrospect, we see that we couldn't classify the steel's application areas based on the parameters, at least not using the methods provided in Intelligent Miner. Does this mean that it isn't possible to determine a steel's application area from the other parameters? The answer is no, it *is* feasible, but we need other methods. The best approach will probably be an expert system, since the domain fulfils the requirements for being able to be modeled in an expert system as discussed in [Wahlster, 1999]. The hidden parameters can then be set up as rules in the expert system, for example.

The reasons why we chose not to pursue the expert system approach were explained in section 2.2.1 earlier.

However, it's important to recognize that even if the approach of finding a product's application area from its parameters didn't work in the steel domain, it might still be applicable in other domains. For each domain that we want to support with the advice agent, a thorough investigation need to be undertaken, in order to see if AI techniques like neural networks or decision trees might be applicable.

9.4 THE COMPARE AGENT

We have arrived at some methods for choosing which auction to bid on as can be seen in Chapter 8. We must however realize that this has been done with a rather heavy assumption, that we always prioritize safety. We saw safety as paramount, but we also know that this might not be the first priority for everyone. We will investigate this further in the future work chapter.

9.5 GENERAL CONCLUSIONS

As we have seen, at least one of the functional plugs that we have described in earlier chapters could not be solved with the methods in this thesis. However, the division into these plugs is still a good idea, since it provides a clear understanding of the functionality that is needed of the respective agent.

Chapter 10 FUTURE WORK

Many issues have not been addressed in this thesis, partly due to time restriction and partly because we didn't have the resources or information at hand needed to solve them. In this chapter we summarize some ideas for future work. First, in section 10.1, the Buy Agent issue is handled. Second, in section 10.3, the Compare Agent future work is discussed and finally in section 10.4 a problem with the auction context is elaborated.

10.1 THE BUY AGENT

The bidding algorithms that we have looked into only use the history of prices in order to predict the future price. This works, at least in part, rather well if the domain is stable, but in the future we would also be interested in having viable bidding algorithms for volatile markets. In our case we have the stainless steel domain as the volatile market. What other factors besides the history of bids are important for the stainless steel domain?

When we did some research into this area it turned out that the price for Nickel was ten times more important for the total price than that of the raw steel itself. The prices for stainless steel soared when the prices on Nickel turned shy-high in 1989. One area of future work would be to add information about Nickel and the other elements that constitute the stainless steel to the bidding algorithm to see if better results will be produced.

Another issue that needs to be addressed is the data. We need to get actual auction data for the steel domain to test the prediction algorithms on. This might be obtained from MetalSite [MetalSite, 2000] or some other BTB auction vendor site.

10.2 THE ADVICE AGENT

When we looked at the steel samples we divided the application areas into seven categories. This means that we had seven classes to choose from when we were to classify each steel sample. An alternative is to express the problem as seven single problems, where each problem is determining whether a steel is a certain application area (e.g. offshore products) or not. This might increase performance.

10.3 THE COMPARE AGENT

The most important thing to address next in the Compare Agent is the issue of heterogeneous auctions. What auction should we bid on if there are auctions with different auction protocols?

Another important thing that should be discussed is how the Compare Agent should behave if we didn't prioritize safety. The decisions are then much more intricate to make. For example, whether we will continue on a sure auction or jump to a more uncertain one, but where the possible profit is much higher.

10.4 THE AUCTION CONTEXT BREAKS DOWN

There is a possibility that the buyer might have preferences from which seller he wants to buy. And correspondingly, a seller might have preferences for which buyer he wants to sell the product to.

For the seller this poses a genuine problem. It will lead to a breakdown of the auction scenario, since some auction protocols simply can't handle this very well, or even at all. For example, in a Dutch auction it is difficult for the seller to say that "No, sorry, you can't get the product at this price, because we like your competitor better and he might give a price soon".

For a seller the most obvious solution would be to hold Single Round Closed Bid auctions instead so that he can look at the bids all at once and decide who will win. This of course poses a very big problem for the buyer when deciding a bid, since he does not know how biased his bids will be.

If we want this bidding to be done automatically even when we have preferences about who's selling the product, every company needs to have a value determining how interesting it is. Determining such values, and updating them for that matter, is something for the future.

Appendix A. PROTOTYPING TOOLS

This appendix gives a brief overview of the different existing frameworks considered for the implementation of the prototype within this project. Further, the framework of choice, AgentBuilder is evaluated.

A.1 THE PURPOSE RESTATED

As stated in section 1.2 earlier, the main purpose of the prototype is to demonstrate the proposed design solution presented in Chapter 5. Additionally, the implementation of the prototype will also serve as a basic test of this design solution. The prototype is intended for internal use by Intentia only.

A.2 PROTOTYPING TOOLS

A survey of different tools and frameworks for the implementation of (intelligent) agents was done within the limits of this work. The purpose of this survey was to find a suitable tool for the implementation of the prototype. Doing a complete, new implementation of an agent system wasn't an option; implementation details weren't within the focus of this work, nor did the timeframe of the project allow for such an extensive implementation project. Thus, we had to find and use an existing solution as a base for the prototype.

A.2.1 THE DEMANDS

The first and most restrictive demand that had to be met by the tool or framework chosen for the prototype implementation was that it had to be implemented in Java, or at least be able to generate Java output. This cut down the number of available solutions considerably, though we still had quite a few candidates left to choose from. The simple reason for this rigid demand is that Java is the platform of choice for Intentia in the future.

Secondly, we wanted a tool that allows the created agents to communicate via the KQML language. This issue was discussed in detail in section 3.1.3 above.

Further, we wanted a prototyping tool rather than an API or a framework for agent implementation. The reason for this is that we wanted to be able to model and test our concepts efficiently on a quite abstract level, without the risk of getting stuck on specific details of the implementation such as Java-related problems, for example. Since the concept of agents and agent communication can be quite complex, and very likely gives rise to non-trivial implementation issues, this was something we definitely wanted to avoid if possible. This was especially so due to the limited time at hand.

The fourth and last demand was that we, if possible, wanted to use a system that could be considered for a future, actual implementation of an agent system within Movex. Early, we suspected that this demand was likely to conflict with the third one above; any tool or framework considered for large-scale development of a large and complex software like Movex need to be very flexible, efficient at run-time and available for commercial use (that is, licensable under such terms that incorporation with a commercial software is economically and legally possible). Such properties are hard, though not impossible, to combine with those of an efficient prototyping tool. In case of conflict, we regard this demand as less important than the third one above.

A.2.2 DECAF

DECAF (Distributed, Environment-Centered Agent Framework) [DECAF, 2000] is a toolkit for the development of software agents. It's the result of research work conducted at the University of Delaware, and is designed explicitly to provide rapid development of software agents.

DECAF provides a programming API in Java, that lets the developer program agent behavior without extensive knowledge of low-level implementation details. The API provides functions that allow agent programming at an abstract level; it offers functionality for sending KQML messages, asserting beliefs etc. Thus, the developer does not need to focus on language specific details (e.g. socket creation, memory handling etc.) but can concentrate on the agent related concepts.

The basic unit of the DECAF agent behavior is an *action*. An action consists of one or more manipulations of the agent's state or its environment through calls to the API. Several actions can be combined into a hierarchical structure, called a *Hierarchical Task Network (HTN)*, which describes complex behaviors. A "happens-before" relation orders the actions within an HTN, and loop- and conditional constructs are provided to allow for complex behavior modeling. Actions can also be connected to the state of the agent's environment through a *Performance Profile*, allowing an agent to behave differently depending on the state of the environment, although it uses the same set of actions.

All manipulation of actions and HTN:s is made using a graphical user interface, the *Plan Editor*, in which the HTN is represented as a tree-like structure, with the actions (representing sub-goals) as nodes, and the root representing the overall goal of the agent. Actions can be easily duplicated through this graphical interface, allowing for easy re-use of components.

From the graphical representation, a *plan file* is generated. This is the actual agent program.

The DECAF run-time environment consists of a number of agent programs, the API providing agent functionality, and an *Agent Name Server (ANS).* The ANS is basically a regular name server, providing a way to find the physical location of an agent by a logical name. This way, DECAF can be used to build distributed multi-agent systems.

A more detailed overview of the DECAF Framework is given in [Graham & Decker,1997].

A.2.3 AGENTBUILDER

AgentBuilder is an "integrated tool suite for constructing intelligent software agents. It allows software developers with no background in intelligent systems or intelligent agent technologies to quickly and easily build intelligent agent-based applications". It's a commercial software solution provided by Reticular Systems, Inc. [Reticular, 2000]. Even if the above words are from the manufacturer and cannot be regarded as unbiased, they actually quite well describe what AgentBuilder is trying to achieve.

The AgentBuilder software consists of two main parts: an agent development toolkit and a run-time engine. The agent development consists of a central **repository**, in which objects like agent definitions and ontologies (see below) are stored, and several **managers** and **editors**, used to manipulate the different objects. Since all objects are stored in the same repository, reuse of components in several projects or in several agents is well supported. The most important tools include the following:

The *Ontology Manager* is used to manipulate domains. To put it simply, an *ontology* is a specification of the concepts in a domain [Gruber, 1993] provides a detailed definition of ontologies). In AgentBuilder, this means that an ontology consists of all objects used by the agents to fulfill their tasks. All domain-specific properties of an agent, like graphical user interfaces for example, are modeled by the use of *Project Accessory Classes (PAC:s).* A PAC is a regular Java class, which is imported into an ontology and instantiated. An *Object Modeler* is provided for managing all object and relationships between objects in standard UML [UML, 2000] Notation.

The *Agent Manager* is used to define the mental model and the behavior of the individual agents. The behavior is modeled in the *Rule Editor,* by means of *behavioral rules*. These rules are used to trigger certain *actions,* which are defined in the *Action Editor*. Actions typically consist of invocations of methods in PAC:s, and are the agent's way of manipulating its environment. AgentBuilder agents also use the notion of *commitments*. The AgentBuilder definitions of the notions above are based on Shoham's early work on Mentalistic agents and are described in detail in [AgentBuilder, 1998].

The *Agency Manager* is used when developing multi-agent systems. The most important feature is the *Agency Viewer,* which lets the user monitor several agents and the communication between them at run-time.

The *Protocol Editor* is used to create and modify communication protocols consisting of KQML messages, involving two or more agents. A protocol is represented by a state diagram, which can be manipulated graphically. When applying a communication protocol to a set of agents, behavioral rules that implement that protocol are created within the individual agents.

AgentBuilder agents (behavior, communication etc.) are specified in a proprietary definition language called *Reticular Agent Definition Language* (RADL). Although possible, the user does not generally write any RADL code himself. Instead, he uses the different managers to specify the behavior and nature of his agents on an abstract level. This abstract specification is then used for automatically generating RADL code, which can be executed by the **run-time engine**.

The run-time engine executes agent programs, i.e. the combination of a RADL specification and it's associated PAC:s. It also handles all inter-agent communication, and provides ways for the user to control parameters of the environment at run-time. The run-time engine in turn executes in another run-time environment, namely a Java Virtual Machine (JVM).

AgentBuilder agents communicate via KQML. The KQML messages are transported between the different agents using Java RMI (Remote Method Invocation), which makes it possible to build distributed multi-agent systems consisting of several agents executing on different physical machines.

A.2.4 ABE, JKQML AND JATLITE

Three frameworks, *ABE* (Agent Building Environment) [ABE, 2000] and JKQML [JKQML, 2000] developed by IBM Corp, and *JATLite* [JATLite, 2000], developed at Stanford University, have been evaluated and used in an earlier agent-oriented project within Intentia [Pettersson, 1998].

ABE does not provide a complete framework for building software agents. It's basically an inference engine, and a tool for creating agents capable of rule-based reasoning. It doesn't provide any means of agent communication via KQML.

JKQML and JATLite, on the other hand, do provide means for developing KQML-speaking agents in Java. However, they don't provide support for creating intelligent behavior. Thus, by combining ABE with either JKQML or JATLite, we would get functionality comparable to that of AgentBuilder or DECAF (see above).

The combination of ABE and JKQML was used in the project mentioned above. An evaluation of this solution, as well as more detailed overviews of the three frameworks, is available in the project report [Pettersson, 1998].

A.2.5 OTHER FRAMEWORKS

In addition to those mentioned above, several other tools and frameworks were considered to some extent within our survey of the field. An extensive list of solutions from different vendors and research institutes is provided on the AgentBuilder web site [AgentBuilder, 2000]. The solutions listed here range from dedicated agent building frameworks like AgentBuilder and DECAF to general solutions like the VOAGER server [VOAGER, 2000], which can be used to support modeling of software agents although it's not specialized for this purpose.

Thus, the support for various aspects of agents, such as mobility, intelligence and communication, vary a great deal between the different solutions.

A.2.6 CHOICE OF PROTOTYPING TOOL

After completing our survey, we decided to use AgentBuilder as the basis for our prototype work. The main reason for this was its high level of abstraction and multitude of tools for modeling different aspects of an agent. Particularly, we were interested in the possibility to model agent communication in an abstract way through a graphical interface. To our knowledge, this feature is unique to AgentBuilder.

Furthermore, we suspected that the documentation and support provided with a commercial package would be better than that of a research project such as DECAF, for example. This aspect would be especially important if the tool was to be considered as a base for a future, real implementation of an agent system within Movex. Also, in that case, the quality of the tool would have to be guaranteed by the manufacturer, something that would be impossible for a research product.

A.3 TECHNICAL REVIEW

Overall, AgentBuilder is a very competent tool for agent development. It supports many aspects of agent development, and provides an efficient way to construct agents by the use of abstract tools. This is a significant improvement over many other tools comparable in terms of functionality, that only provide this functionality to the developer in the form of a library or programming interface (API).

However, the prototype multi-agent system implemented, although very simple, unveiled some serious drawbacks of AgentBuilder:

The graphical development tools aren't always as intuitive and easy to use as one would like. Even after getting used to the system, some tasks felt more complicated than necessary; it's easy to forget a step somewhere and have to recheck later. For example, the Protocol Manager proved unable to generate full specifications of the communication protocols we modeled. Several of the key arguments in the communication rules generated from an abstract protocol have to be supplied manually afterwards, thus reducing the value of such a protocol. This was a severe drawback, since the Protocol Manager was one of the main reasons why we chose to use AgentBuilder in the first place.

Also, the task of importing PAC:s into an ontology can be quite tedious; one has to go through several steps that are far from intuitive to a user not very familiar with how classes and objects are handled in Java. In other words, the tools do not hide issues related to the implementation of the very well. This makes it harder to focus on the task of specifying agent behavior rather than specific implementation details.

Finally, the development environment isn't always as stable as one might expect from a commercial product.

Other drawbacks are related to the run-time environment:

It's not possible to dynamically create new agents at run-time. In the proposed design (see Chapter 5 above) new agents are created on demand, and terminated as soon as their task is accomplished. This behavior can't be modeled within the current version of AgentBuilder. In the prototype, a rough work-around managed this problem: it's possible to have inactive agents within the system, and to activate/deactivate these during execution to simulate dynamic creation of agents. However, this means that all agents to be used during an execution must be specified prior to system startup, which in turn means that we have to set a maximum total number of agents in advance. Also, system resources for this total number of agents will have to be allocated, even if most of these agents are never executed. See below for a further note on system resource use.

Thus, this work-around is not suitable for a more complex system like a real product, or perhaps even a larger prototype.

Another drawback is the need for the run-time engine itself; not being able to create stand-alone agents affects flexibility in a negative way. Any system incorporating AgentBuilder agents is completely reliant on third-party software, which has to be licensed. Also, it may be hard for the system developer, Intentia in this case, to guarantee the quality of the run-time engine, something that is imperative when developing critical systems based on it.

The run-time engine also consumes a fair amount of system resources, something that rather quickly affects scalability. Each run-time engine is in turn executed in a Java Virtual Machine (JVM). During the work on the prototype, we have been unable to run the system without using one individual JVM for each agent, thus putting a heavy load on the underlying system even with quite a small number of agents. Although AgentBuilder is capable of distributing this load over several machines, it's a serious problem.

The conclusion is that AgentBuilder, although suffering from some drawbacks, is quite suitable for prototyping moderately sized agent systems such as the one proposed. It provides a high degree of abstraction in the development work, and supports development of both intelligence mechanisms and inter-agent communication. As a basis for implementing intelligent agents within a real system such as Movex, however, Agentbuilder is not very likely to be suitable due to the inflexibility and high usage of system resources discussed above.

Appendix B. REFERENCES

[ABE, 2000]

IBM Corp. (2000). Agent Building Environment.
[Online]. Available: http://alphaworks.ibm.com/tech/abe/

[AgentBuilder, 1999]

Reticular Systems Inc. (1999). AgentBuilder – A Toolkit for Constructing Intelligent Software Agents. Revision 1.3, February 18, 1999.
[Online].Available: http://www.agentbuilder.com/Documentation/white_paper_r1.3.pdf

[AgentBuilder, 2000]

Reticular Systems Inc. (2000). AgentBuilder Toolkit.
[Online]. Available: http://www.agentbuilder.com/

[Ashcraft, 1994]

Ashcraft, Mark. (1994). *Human Memory and Cognition, Second Edition.*
New York: HarperCollins College Publisher.

[Avesta, 2000]

Avesta Sheffield AB. (2000). [Online]. Available: http://www.avestasheffield.com

[BargainFinder, 2000]

BargainFinder. (2000). [Online]. Available: http://www.bargainfinder.com/

[Bidlet, 2000]
Bidlet AB. (2000). [Online]. Available: http://www.bidlet.se/

[BizTalk, 2000]
Microsoft Corp. (2000). BizTalk.
[Online]. Available: http://www.microsoft.com/biztalk/

[Blixt & Öberg, 2000]
Blixt, K; Öberg, R. (2000). SAFT – Software Agent Framework Technology.
Master's Thesis Report, University of Linköping. LiTH-IDA-Ex-2000/14.

[Blom, 1989]
Blom, G. (1989). *Sannolikhetsteori och statistikteori med tillämpningar*.
Lund: Studentlitteratur.

[C|Net, 1999]
Festa, P. (1999). *"Intelligent Agents" Make a Comeback*, CNET News.com, Oct. 28, 1999. [Online].
Available: http://abcnews.go.com/sections/tech/CNET/cnet_intagents991028.html

[CORBA, 2000]
Object Management Group, Inc. (2000). Common Object Request Broker Architecture. [Online]. Available: http://www.corba.org/

[DECAF, 2000]
University of Delaware. (2000). Distributed, Environment-Centered Agent Framework. [Online]. Available: http://www.cis.udel.edu/~graham/DECAF/

[eBay, 2000]
eBay Inc. (2000). [Online]. Available: http://www.ebay.com/

[ebXML, 2000]
UN/CEFACT. (2000). ebXML. [Online]. Available: http://www.ebxml.org/

[Edholm 2000]
Edholm, U. (2000, spring). DM Technical Sales IBM Software, Sweden. Interviews at Intentia R&D, Linköping.

[Eldén & Wittmeyer-Koch, 1996]
Eldén, L; Wittmeyer-Koch, L. (1996). *Numerisk analys – introduktion.*
Lund: Studentlitteratur.

[EJB, 2000]
Sun Microsystems, Inc. (2000). Enterprise Java Beans.
[Online]. Available: http://www.javasoft.com/products/ejb/

[Finin et al, 1993]

Finin, T; Weber J; et al. (1993). Draft Specification of the KQML Agent Communication Language.
[Online]. Available: http://www.cs.umbc.edu/kqml/kqmlspec/spec.html

[Finin et. al, 1997]

Finin, T; Labroy, Y. and Mayfield, J. (1997). KQML as an Agent Communication Language. *Bradshaw, J. (Ed.), Software Agents*. United Kingdom: MIT Press.

[Finin & Mayfield, 1995]

Finin, T; Mayfield, J. (1995). A Security Architecture for the KQML Agent Communication Language. CIKM'95 Intelligent Information Agents Workshop, Baltimore, December 1995.

[Fishmarket, 1999]

Artificial Intelligence Research Institute, Spanish Scientific Research Council. (1999). The Fishmarket Project.
[Online]. Available: http://www.iiia.csic.es/Projects/fishmarket

[Forrester, 2000]

Forrester Research Inc. (2000, February). *Is Nonstop Enough*? Commercial report.
[Online]. Available: http://www.forrester.com/

[Frakes, 1992]

Frakes, W; Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall.

[Friedman & Rust, 1993]

Friedman, D; Rust, J. (1993). *The Double Auction Market*. Addison-Wesley.

[Gartner, 2000]

Gartner Group Corp. (2000). [Online]. Available: http://www.gartnergroup.com/

[Graham & Decker, 1997]

Graham, J.R; Decker K.S. (1997). Towards a Distributed, Environment Centered Agent Framework. Department of Computer and Information Sciences, University of Delaware. [Online].
Available: http://www.cis.udel.edu/~graham/DECAF/Papers/decaf.ps

[Gruber, 1993]

Gruber, T.R. (1993). Toward principles for the design of ontologies used for knowledge sharing. Stanford Knowledge Systems Laboratory Report KSL-93-04.
[Online]. Available: http://ksl-web.stanford.edu/knowledge-sharing/papers/README.html#onto-design

[Hackman, 1996]

Hackman, P. (1996). *Boken med Kossan på*. Linköping: Hut, Hyfs och Hållning Productions.

[Hu, 1999]

Hu, J. (1999). *Agent Service for Online Auctions*.
[Online]. Available: http://ai.eecs.umich.edu/junling/papers/workp/workp.html

[IBM, 2000]

IBM Agent Research Group. (2000, February 8-9). Interview with IBM Agent Research Group (Chair: White, S.R. Senior Manager, Massively Distributed Systems Research Division) Thomas J. Watson Research Center, Yorktown Heights, NY. Interview at Intentia R&D, Stockholm.

[JATLite, 2000]

Standford University. (2000). JATLite. [Online]. Available: http://java.stanford.edu/

[JKQML, 2000]

IBM Corp. (2000). JKQML.
[Online]. Available: http://alphaworks.ibm.com/formula/jkqml/

[Kumar & Feldman, 1998]

Kumar, M; Feldman, S.I. (1998). *Business negotiations on the Internet*.
IBM Research Division,T.J. Watson Research Center,Yorktown Heights, NY.
[Online]. Available: http://www.ibm.com/iac/reports-technical/
reports-bus-neg-internet.html

[Laufman, 1998]

Laufmann, S.C. (1998). Agent Software for Near-Term Success in Distributed Applications. *Agent Technology – Foundations, Applications, and Markets.* Springer Verlag.

[Mehta et al, 1998]

Mehta, Manish; et al. (2000). *SLIQ: A Fast Scalable Classifier for Data Mining*. Almaden Research Center, IBM Corp.

[Metalsite, 2000]

Metalsite Virtual Marketplace.(2000). [Online]. Available: http://www.metalsite.com

[Nordberg, 2000]

Nordberg, H. (2000, spring). Professor, Avesta-Sheffield AB, Sweden. Telephone interviews, Linköping and Stockholm.

[Petersson, 1998]

Petersson, M. (1998). Software Agent Technology in Business Applications. Master's Thesis Report, University of Linköping. LiTH-IDA-Ex-98/51

[Reticular, 2000]

Reticular Systems Inc. (2000). [Online]. Available: http://www.reticular.com/

[Sachs et. al, 2000]

Sachs, M; Asit, D; et. Al. (2000). Executable Trading-Partner Agreements in Electronic Commerce. IBM Corporation.
[Online]. Available: http://www-4.ibm.com/software/developer/library/tpaml.html

[Shoham, 1997]

Shoham, Y. (1997). An Overview of Agent-oriented Programming. *Bradshaw, J. (Ed.), Software Agents.* MIT Press.

[ShopBot, 2000]

ShopBot. (2000).
[Online]. Available: http://www.cs.washington.edu/homes/bobd/shopbot.html

[Tivelius, 1999]

Tivelius, Carl. (1999). Interview at Intentia, autumn 1999. Intentia Sweden.

[UML, 2000]

Object Management Group, Inc. (2000). Unified Modelling Language.
[Online]. Available: http://www.omg.com/uml/

[VOYAGER, 2000]

ObjectSpace Inc. (2000). Voager Platform.
[Online]. Available: http://www.objectspace.com/products/prodVoyager.asp

[Wahlster, 1999]

Wahlster, W. (1999). Vorlesung im SS 99 – Expertensysteme und deduktive Datenbanken. Universität des Saarlandes.

[Weiss et al, 1991]

Weiss, S; Kulikowski, C; et al. (1991). *Computer Systems that learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers.

[Wooldridge & Jennings, 1995]

Wooldridge, M; Jennings, N.(1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*.