Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Màster en Intel·ligència Artificial

Tesi de Màster

Predicting Stock Trends: A Multi Agent Approach

Estudiant: Xavier Dolcet Figueras
Director(s): Javier Béjar Alonso

Data: 19/06/2009

Acknowledgments

It's been a very long journey, but finally here I am, writing the last page (and, ironically, the first) of this document. This work puts an end, at least momentarily, to my student life. All these years around FIB are condensed into this pages.

Computers have eased our lives in many aspects, from writing a document to shopping groceries. Artificial Intelligence is another step further in this direction, an enormous step. Thanks to it, computers are becoming autonomous, and thus, are beginning to be able to perform tasks in our behalf. This thesis is the starting point for creating such a system, in this case applied to investment management.

Artificial Intelligence is one of the main reasons why I decided to study Computer Science. I still remember how excited I was when I started reading Russell and Norvig's book. Joining this Master was my next logical step. And now, this journey is over.

All this wouldn't have been possible without the help of many people and, as it's usual, I'd like to start this document thanking all of them for their support. A big “Thank you” to everyone, specially to my entire family for always being there, listening to every word I said, despite they barely understood a thing.

I also want to thank the patience and guidance of my thesis Director, Mr Javier Béjar, who has kindly supervised this pages. Thank you for always replying lightning fast to my questions.

I can't finish this section without thanking my friends, for being there with a welcoming smile despite not being able to spend much time with them. I'm very lucky for having them by my side.

To my fellow coworkers, and specially to my bosses, for being so comprehensive every time I went to work in zombie mode for staying till late the night before working in this thesis.

And finally, to my friends at BEST Barcelona, for all the good memories I keep of this last year as a university student. I'll never forget it.

Contents

Figures

Chapter 1

Introduction

This thesis aims to develop a scalable distributed system that provides reliable information about the trend that a certain security will follow in the Stock Market using Machine Learning techniques.

The core of the research presented in this thesis is the creation of a Multi Agent based system capable of predicting whether the price of a stock will rise, drop or stay. In order to do so, the problem is divided into three parts: Information Retrieval, Data Analysis and Data Visualization.

The first part is focused on retrieving the necessary information from Internet and on transforming this raw data into computer-understandable structures that will allow further analysis. In addition to this, new financial indicators are calculated to provide more meaningful data to the system.

The second part is focused on analyzing this preprocessed data using several Machine Learning methods. The methods that have been selected to execute the analysis are: Artificial Neural Networks, Decision Trees, Support Vector Machines and Reinforcement Learning. The idea behind using different methods besides testing the performance of each in this scenario, is the creation of a team of “data analyzers” like the ones found at investment firms. Following the “Keep it Simple” principle, third party libraries have been used when possible to diminish implementation costs.

Finally, the third part deals with the problem of how to present the data and results to the users in a clear but informative way. Just this part on this own could perfectly be a Final Project in a Media Degree, so here we will present a gentle introduction to the Data Visualization world.

Since this thesis is also a Computer Science Engineering Final Project, emphasis will be made in describing the system architecture and the technologies used to create it. In part because of this, this Thesis aims to create a Proof of Concept for a possible future product instead of realizing just a evaluation of Machine Learning methods applied to predicting stock trends.

1.1 Structure of the Thesis

'Background Information' includes all the information necessary to understand what Stock Markets are, why do they exist, and how do they work. In addition to this, the two principal investment strategies are described, and the key question is presented: Is it possible to predict the Stock Markets?

The ''BROMAS – BROker Multi Agent System" chapter presents this Thesis by defining its Goals and Scope, the Benefits that it will deliver, and the Motivation behind it.

The chapter "State of the Art and Related Work" serves as an introduction to the use of Machine Learning in Financial Forecasting. Describes which methods have been used, and for what purpose. In addition to this, a brief introduction to Trading Agents is realized. The chapter ends with the description of MASST, the most related project found.

"Theoretical Introduction" is the most dense chapter of all the Thesis. We tried to make a gentle introduction to the key concepts related to this Thesis: from Agents to Reinforcement Learning. This chapter is divided into three sections: Agents, Multi Agent Systems and Machine Learning.

In "System Description" BROMAS is finally presented. The system architecture is introduced, and the different methodologies and technologies used for developing it are listed and described.

"Design and Implementation" describes the solution adopted to implement BROMAS. It details the internal design and implementation of the three components of the System: Multi Agent System, Data Management and Client Application. It is the essence of our approach to Stock Trend Prediction using Intelligent Agents.

The chapter "Testing and Execution Results" describes the tests that have been executed and the results that were obtained after. This chapter also includes the BROMAS tuning mechanism and the code testing methodology.

"Conclusions" is where we will present our final thoughts regarding our approach to Stock Trend Prediction. Besides this, the Thesis development process will be described in terms of time and money invested. Finally, we present our plans for future releases of BROMAS.

Chapter 2

Background information

This chapter provides a brief introduction to Stock Markets and Stock Trading. From what a Stock Market is and its historic background to the main investment strategies used nowadays to trade Stocks and other securities.

2.1 The Stock Market

A Market is, by dictionary definition, any structure that allows buyers and sellers to exchange any type of goods, services and information. A Stock Market is a type of market in which the trading of stocks, bonds or other securities takes place. This means that, broadly speaking, the only difference between a Fruit Market and a Stock Market lies in the kind of goods that are exchanged within it.

The origins and purpose of markets that trade with primary goods are common knowledge, but which are the origins of modern Stock Markets? And which is the purpose of its existence? In this section a brief explanation of the history and purposes of Stock Markets is presented.

2.1.1 History of the Stock Market

The roots of modern Stock trading are found at the end of XV century in the medieval fairs celebrated all around Western Europe. Is in these fairs where the practice of trading “financial assets” began, by the exchange of real-state assets, commodities and noble titles.

Despite of this, the nobles weren’t the ones who came up with the idea of dividing a business in shares. In those years of increasing overseas commerce between Europe and its colonies, many well-established merchants wanted to invest in the new opportunities this commerce created. However, the amount of money necessary for doing so could not be raised by a single merchant. Because of this, merchants started to pool their funds together to create businesses as partners. Each partner contribution to the venture was represented through a unit, or a share.

Originally, stock trading began on an informal note. Traders hold meetings at their houses, coffeehouses, plazas... In Bruges, Belgium, trader meetings were hosted at Ter Buerse plaza, named after the wealthy Van Der Buerse family. It is said that this location gave birth to the use of the word Bourse to refer to an Stock Market.

In the course of time, this informal meeting was institutionalized and became the Bruges Buerse. The idea widespread all across Flanders, and new “Buerses” were created in major cities like Ghent and Amsterdam, culminating with the construction in Antwerp of the first building explicitly built to trade stocks in 1531. Some years later, in 1602, the Amsterdam Stock Exchange was founded by the Dutch East India Company, the first company to issue stocks and bonds. Since then, Stock Markets opened in the most important cities in the world: London (1570), New York (1792), Paris (1794).

2.1.2 Purpose

The Stock Market is more than a mere building where assets are exchanged. The activities that take place inside can skyrocket the economy or cause a financial crisis. The main purposes of the stock markets are the following:

Mobilize savings for investment, by putting in contact companies with small investors. Any person can acquire shares and other securities in the Stock Market. This way, money that would have been spent or kept in bank deposits are invested in companies to promote their activity, resulting in its economic growth.

Raising capital for businesses and governments. By selling shares and bonds, companies and governments are able to obtain the capital necessary to fulfill their objectives.

Grant liquidity to investments, allowing investors to quickly and easily sell securities. The investor just has to issue a selling order to obtain the value of his shares in cash.

Facilitating company growth, by realizing a fusion or an acquisition of another company. An acquisition is an opportunity to increase the market share, acquire new products, acquire necessary assets, etc. The mechanisms that the Stock Market provides makes this process much simpler.

Redistribution of wealth, dividends and stock price increases can lead to capital gains. This way, all investors can benefit from the wealth that a profitable business has achieved.

Barometer of the economy. The movement of stock indexes and share prices are commonly used as an indicator of the current trend in the economy.

And the benefits that a company gets for “going public”, besides the previously mentioned capital raise, are detailed next:

Improves the company Image and Prestige, going public guarantees that the company is solvent, since its financial situation and results are public. By sharing ownership with the public, its reputation is spread and that may lead to find new business opportunities.

Provide liquidity to current shareholders, the founder members have the possibility to trade their actions to obtain cash any time they need it. Besides, this allows to diversify the ownership of the company and provides a more objective valuation of it.

Finance company growth, when a company is in grow mode, needs to raise capital to fuel that grow. Issuing shares allows the company to access a substantial source of corporate funding.

Motivate employees, stocks and stock options can be used to attract new and retain key employees. These plans are commonly used by start-up enterprises. An allocation of shares can lead to increased productivity and loyalty.

Acquisitions & Mergers, once a company is public and its shares available to the public, stocks can be as valuable as cash when acquiring of merging with another company.

In conclusion, the Stock Market main purpose is to finance company growth allowing them to raise capital from all kinds of investors in a much easier and safer way.

2.1.3 Goods Traded

In Stock Markets the goods that are traded are called securities, a certificate attesting credit, the ownership of stocks or bonds, or the right of ownership connected with tradable derivatives. It's implicit in the previous definition that different types of securities exist. In fact, securities can be divided into 2 groups: *Debt Securities* and *Equities*.

The investor who holds a *Debt Security* becomes a creditor of the company who has issued it. The holder is typically entitled with the following rights: to receive the payment of a fixed interest, to receive the redeem once the term is over and to be able to transfer the security.

In the Market different types of Debt Securities can be acquired, next there's an introduction to some of the most common:

- Debentures, in Spanish *obligaciones*, represent the debt of commercial or industrial entities. It's a long term investment, typically at least ten years.

- Bonds, or *Bonos* in Spanish, is a product very similar to Debentures but with a shorter maturity, usually between 3 and 10 years.

- Bills, or *Letras*, are short term securities, normally lasting between 3 months and one year. In Spain, the most known are Letras del Tesoro, issued by the Spanish Government.

An Equity Security is a share in the capital stock of a company. When a company *goes public*, its capital is divided in shares that can be traded in stock markets. From that moment on, price is established by the rules of demand and supply. This way, price reflects the expectations that investors have in the company. This is why Stock Markets are said to be good barometers of the Economy.

When an investor buys a share, owns a part of the issuer's capital, and thus, becomes a co-owner of it. The ownership of a share grants the investor the following rights:

- *Dividends*, the profits obtained by a company have to be used first to compensate past losses. Second, to pay taxes. And finally, what is left can be used for keeping some reserves and/or to pay the shareholders. This sum of money is called the Dividend.

- *Transmission*, the shareholder has the right to trade his shares if he finds a buyer. The difference between a higher selling price ad a lower purchase price is called *Capital Gain.*

- *Preferred Subscriptions,* when a company realizes a capital expansion, current shareholders have preference in buying new shares.

- *Voting rights,* usually shareholders have on vote per share owned, grating the right to vote on matters such as elections to the board of directors.

There's one kind of financial product that, although is not traded in the Stock Market, is important enough to be included in this section: *Derivatives*. As its name suggests, *Derivatives* are financial contracts whose values are derived from the value of something else, called the underlying. *Derivatives* are traded in the Derivatives Market, and the most popular products we can trade there are:

- *An Option* is a financial derivative that represents a contract that offers the buyer the right, but not the obligation, to buy *(call)* or sell *(put)* a security or other financial asset at an agreed-upon price, known as *strike price*, during a certain period of time or on a specific date, the *exercise date*.

- *A Future* is a financial contract obligating the buyer to purchase an asset (or the seller to sell an asset) at a predetermined future date and price. Futures contracts detail the quality and quantity of the underlying asset; they are standardized to facilitate trading on a futures exchange.

2.1.4 Spanish Securities Market

To fully understand how a modern Stock Market works, in this section the Spanish Securities Market is used to explain how a Market is organized, how the stock exchange process is realized and what measures are taken to make it secure and reliable.

First of all, we shall take an overview at the Spanish Securities Market structure.

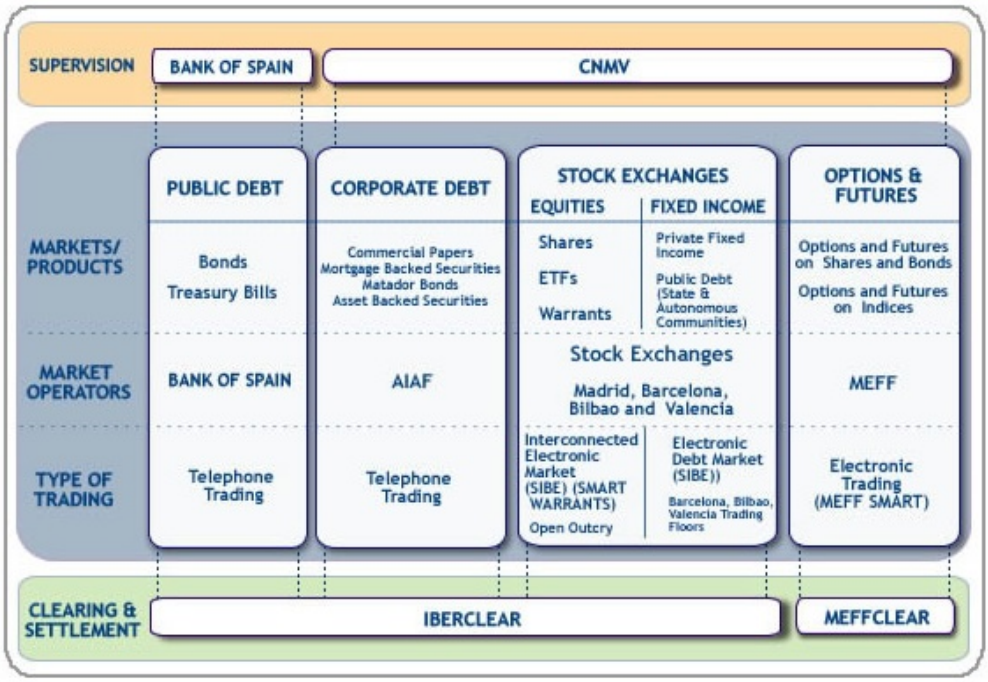


Figure 2.1: Spanish Securities Market

As can be seen, the Securities Market is divided into four different Markets: Public Debt, Corporate Debt, Options & Futures and Stocks.

SIBE is the name of the Spanish Stock Market Interconnection System. This system connects the 4 Spanish Stock Exchanges (Madrid, Barcelona, Bilbao and Valencia) and is where almost the 99% of all trading operations take place.

Spanish Stock Exchanges are open every working day from 8:30 to 17:30. Markets start with an Adjustment Auction from 8:30 to 9:00. During this, bids can be introduced, modified or cancelled but no trading is realized. The auction ends with a random finale, to avoid price manipulation, of 30 seconds that determines the opening price of stocks.

From 9 to 17:30 is the Open Trading session, during which orders to buy or sell stocks are introduced and executed if matched. At 17:30 starts the Closing Auction. It works exactly as the Adjustment Auction, but lasts only 5 minutes. Of those, the last 30 seconds are the Random End, introduced to avoid Scooping, a technique used to manipulate the closing price of stocks.

The price of a stock is established by the rules of demand and supply. If there are more buying than selling orders for a stock, the price will rise. On the other hand, if there are more selling orders, the price will drop. A buying order establishes the maximum price that

the investor is willing to pay and the selling order fixes the minimum price that the holders accepts.

During the Open Trading, prices are updated when there's a match between a buying and a selling order. This means that the stock price is the last negotiated price of a share, as the result of the execution of each order.

The settlement of all the transactions realized in the Spanish Stock Market is the responsibility of Iberclear. This company is in charge of both the Register of Securities, held in book-entry form, and the Clearing & Settlement of all trades. This means that Iberclear is responsible for exchanging securities for cash and vice versa.

The supervision and inspection of the securities markets and of the activity of all physical and legal persons involved therewith is entrusted to the Spanish Stock Exchange Commission (CNMV). This Commission is also responsible for providing extensive and up-to-date information to investors. Its objective is to make the Spanish Stock Market transparent and trustworthy.

2.1.5 Popular Stock Markets and Indices

An introduction to Stock Markets wouldn't be complete without introducing the most important Stock Markets and explaining what are the acronyms that we usually see and hear in the news.

In the United States of America we have the most popular Stock Market in the world, the New York Stock Exchange, commonly known as Wall Street. It is the largest stock exchange in the world by dollar value of its listed companies' securities. NASDAQ, the largest electronic screen-based equity securities trading market in the United States, is also located in New York City.

In Europe the most important Stock Markets are the London Stock Exchange, the Euronext NV (which includes Paris, Amsterdam, Brussels and Lisbon) and the Frankfurt Stock Exchange.

In Asia, we find the Tokyo Stock Exchange. The TSE is the second largest stock exchange market in the world by market value, just behind the NYSE. As of year 2007, the TSE had over 2400 companies listed.

Stock Market Indices are statistic tools used to measure the price evolution of a section of the Stock Market. The Index valuation reflects the gains or losses of the values that compose it, and thus, are used to track the evolution of Stock Markets. Because of this, many have become the baseline for benchmarking the performance of portfolios.

Not all included companies have the same influence over the index price. Depending of the price, the size of the company, the number of shares, etc. each component is weighted to control its influence. Following there's a list including the most influential Indices.

Dow Jones Industrial Average, shows how the stocks of 30 of the largest and most widely held public companies in the USA have traded. The DJIA includes companies of various sectors (the “industrial” tag is kept for historic reasons) such as Coca-Cola, Walt Disney, General Motors and IBM. It’s the most widely followed index, despite criticism for only including 30 stocks, a very small sample of the thousands of companies that are publicly traded.

NASDAQ Composite, covers all common stocks listed in the NASDAQ, meaning that over 3000 companies are included. It’s mainly used as a indicator of the performance of technological companies and growth stocks.

S&P 500, is an index of the prices of 500 large capitalization common stocks actively traded in the USA. Covers large publicly held companies that trade either on the NYSE or on the NASDAQ. The S&P 500 is often used as a baseline level of performance.

FTSE 100, is a share index that covers the 100 most highly capitalized UK companies listed on the LSE.

CAC 40, represents a capitalization-weighted measure of the 40 most significant values among the 100 highest market capitalizations on the Euronext Paris

DAX 30, is an index consisting of the 30 major German companies trading on the FSE.
EuroStoxx 50, the Dow Jones EuroStoxx 50 is a stock index of Eurozone stocks that covers the Supersector leaders in the Eurozone.

IBEX 35, is a market capitalization weighted index comprising of the 35 most liquid Spanish stocks traded in the Madrid Stock Exchange.

Nikkei 225, is the oldest and the most well known asian index. The Nikkei 225 is a price-weighted average index designed to reflect all the overall market and is thus used as the major indicator for the Japanese economy.

2.2 Investment Strategies

There are many different approaches to money investing, classified into two main (and antagonist) groups: Fundamental analysis and Technical analysis.

The *Fundamental analysis* consists in estimating the ‘real’ value of a company by deeply examining its financial situation, market position, competitive advantages and the competitors. It’s performed on historical and present data with the goal of making financial forecasts, and more specifically, to predict its probable price evolution.

On the other hand, *Technical analysis* completely ignores the company situation to concentrate in chart analysis. Technical analysis considers primarily the price and the volume behavior, in addition to indicators such as relative strength index, moving averages, etc. With this past and present information, technical analysts attempt to identify price trends and patterns to forecast price evolution.

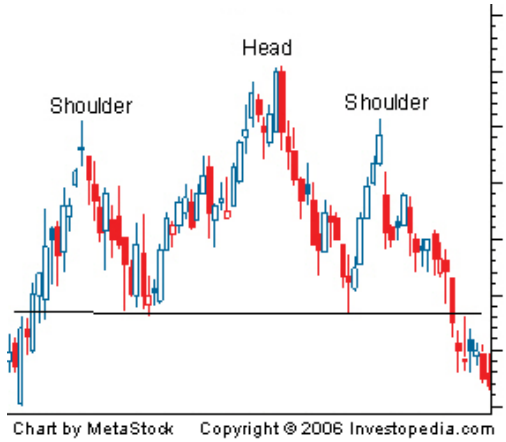


Figure 2.2: The Shoulder-Head-Shoulder pattern

There's a lot of controversy regarding Technical analysis and if it actually works or not. For example, the Efficient Market Hypothesis (EMH) asserts that Financial Markets are "informationally efficient", and because of this, prices already reflect all known information. It's impossible to consistently outperform the markets by using any information that the markets already know, except through luck. Besides the EMH, Technical analysts also have to cope with negative comments coming from Fundamental analysts, such as Warren Buffett: "I realized technical analysis didn't work when I turned the charts upside down and didn't get a different answer".

Despite of this, technical analysts claim that they experience continuous positive returns by using chart analysis methods. They say that EMH ignores the way markets work, stating that people are not entirely rational actors, and that many base their expectations in past results or track record, for instance. This irrational human behavior influences stock prices, and thus, leads to predictable outcomes. Social dynamics are a key factor, and EMH underestimates it.

Chapter 3

BROMAS – BROker Multi Agent System

3.1 Goals and Scope

The BROMAS project aims to create a decision support system for people interesting in investing in stock markets. This project targets not only to professional traders but to anyone attracted to this exciting field. I aim to create a trustworthy and objective system that will help others to decide where and when to invest. Thus, this thesis goal is not only to analyze the stock market using Artificial Intelligence methods but to present the users all the information they need to make their own decisions. Unfortunately (specially for me), the system can only give a recommendation without any kind of formal guarantee. It's almost impossible to beat the market in a consistent way.

In order to achieve this goal, work has to be done in the following areas:

- retrieve the data necessary to realize an exhaustive analysis
- preprocess the data to maximize the quality of the obtained results
- design a multi agent system open enough to be easily extensible yet closes enough to keep things under control
- analyze the data using several machine learning methods
- present all the gathered and generated data to the user in a simple yet meaningful way

These 5 subgoals are the starting point to define the scope of this thesis. In order to do so, I have to further detail each of them.

First of all, we have to find out which data is required in order to do a correct analysis, basic knowledge in stock analysis is compulsory. After we have the data requirements defined, we have to find from where we are going to retrieve this data (from a public source if possible). Finally, we have to transform this data to make it processable by software agents.

The design of the multi agent system is one of the key parts of this thesis. We have to define which agents will conform the system, the interaction protocols between them and the data necessities of each. Design phase will try to build a reliable multi agent system able to be easily extended in the future by adding new agents or functionalities. Standard methodologies will be used to try to guarantee all this.

The data analysis using machine learning techniques is the most decisive part of this thesis. If the analysis is not valid nor productive the system will be totally pointless. We have, thus, to carefully choose which techniques to use, tune their parameters precisely and test them exhaustively to assure a minimum level of quality in their results.

And finally, we have to design a way to present all this data to the users so they can make better decisions when investing their savings. Unfortunately, my designing skills are quite lacking at the moment, so no multi-touch 3D visualizations (yet).

3.2 Benefits

There are several benefits that this thesis provides in comparison to other similar systems. The first benefit is that this system is planned to aggregate data from different sources to help the users make more informed decisions, we want to provide users with all the relevant information that is available. In addition to this and to avoid death from excessive information, data will be presented using visualization tools to increase its understandability.

Unlike others, this system will be totally neutral. I'm not part of an investment company nor of a bank, I have no interest in influencing the decisions users make. All the information gathered from users will be used to further improve the system. Besides this, he only “opinion” the system will give is a mere recommendation of what to do in the immediate future.

Another important feature is that this system is designed to be available from any device with internet connection, using a web interface or some kind of native application depending of the device. Because of this, the design of the system has been done following a distributed approach using a multi agent system. This will allow the system to run across several machines in order to increase its performance and solve scalability issues.

And finally, self improvement. Like other systems, BROMAS will be in constant development to add new features and correct errors to make it more precise and, consequently, more useful. In addition to this, the use of machine learning techniques allows the system to self improve using its past decisions and data gathered from users.

3.3 Motivation

In previous sections, we saw that not even the stock market specialists are able to come to an agreement regarding wether or not there are strategies which are able to consistently beat the market. The current economic situation has made this lack of forecasting precision quite clear. Not even the top financial analysts were abel to predict such a serious economic crisis. Financial markets are too sensitive, they're affected by so many elements that it's quite impossible to realize predictions with enough confidence.

With this, I don't pretend to say that this project was designed in order to find a solution to this problem, I'm not that pretentious. Truth is, what I really wanted was too see how well (or bad) would the techniques that I had learned during this Master in Artificial Intelligence perform in such a volatile environment.

I chose to apply these techniques to the study of financial markets because I thought it was an area in which the use of these methods could provide an innovative yet handful approach. It was also the perfect excuse to increment my knowledge in Economics, one of my favorite topics. During High School, there was a time in which I couldn't choose between studying Economics or Computer Science. Guess my childhood dream of building intelligent robots is the cause of me being here writing a Master Thesis in Artificial Intelligence instead of studying Business plans. This project was a great way to fill the gap between these two exciting fields. Besides, due to the amount of money that's being “gamed”, there's a huge necessity of up-to-date information. Information that is freely available in the Internet waiting to be gathered and used. My thesis aims to use this publicly available information to satisfy this hunger for valuable economic information.

Another area in which I am also interested is the “startup world”, everything that has to do with entrepreneurship, technological or not. Because of this, I wanted also to create “something”, not just a mere experiment on data. And if this something could eventually become the base layer of my own technological firm, the better.

Chapter 4

State of the Art and Related Work

The idea of applying Artificial Intelligence techniques to Financial Analysis is not new. We have systems from the late 80s that perform Bankruptcy predictions using techniques such as Artificial Neural Networks. From that point, more and more financial institutions and firms have used Artificial Intelligence to improve their systems.

In Finance, small improvements can generate a huge impact. We all know the history of how American Airlines saved $40,000 in 1987 by eliminating one olive from each salad served in first class. Because of this, the amount of research produced in this are is quite extensive.

In this section, we are going to focus in intelligent agents that realize financial analysis/forecasting and in decision support systems. Truth is, it was quite surprising to find only one similar system when searching for “multi agent stock” in Google Scholar. One would expect this field of study to be quite crowded with systems of similar characteristics, mainly because of the huge amount of money that is involved in it. It was disturbing, how come nobody else had thought about his? We haven't found an answer yet. Maybe there are patents that prevent research, or maybe research ended becoming a commercial product and thus decided to keep their philosopher's stone secret. Anyway, we were decided to study this, so we left behind all our concerns and started reading some papers for inspiration.

4.1 Machine Learning and Finance

Financial time series forecasting is regarded as one of the most challenging applications of modern time series forecasting. Financial time series are inherently noisy, non-stationary and deterministically chaotic.

The first source of information had to be how Artificial Intelligence is applied to Stock Markets, which methods are used and for what purpose. We found several reviews of the state of the art of AI applications in Business, including one from a former teacher of the Master, Alfredo Vellido.

What we noticed is that the most popular technique by far is Artificial Neural Networks. It has been used mainly to minimize risks in banking and insurances, like realizing credit evaluation, mortgage risk assessment or insolvency prediction. Regarding stock markets, ANNs have been used to forecast financial time series, detection of regularities in price movements, and price trend prediction.

Until the disruption of ANNs, most finance analysis systems were based in Expert Systems or linear statistics methods such as ARIMA. The use of ANNs provided many benefits. First, they're data driven and thus require few a priori assumptions about the models for problems under study. In contrast, Expert Systems requires a knowledge base composed by a series of if-then rules that represent the knowledge of an expert in the field. Moreover, Expert Systems are unable to perform effectively when the knowledge is incomplete, ambiguous or partially erroneous. In situations were the knowledge is hard to specify, the creation and maintenance of the knowledge base is extremely difficult.

In addition to this, ANNs ability to generalize allows them to correctly tackle unseen situations, even if they include noisy data. Finally, linear statistics methods assume that the financial time series under study is generated from linear processes, ANNs don't. Unfortunately, there's no perfect method, and ANNs have their defects too. The biggest one is that Neural Networks act as a blackbox system. We get outputs, but we there's no way to find out how that conclusion was reached. The information provided by the Network, the connection weights, can't be translated into an intelligible explanation. In addition, ANNs also have the danger of overfitting the training data, and the possibility of getting stuck in a suboptimal solution. Despite of this, ANNs have been extensively applied to forecasting financial time series.

Other Machine Learning methods haven't been so extensively applied to Finance Data. Not because of performance, but of being more “research oriented”. Thankfully, more and more research is done to apply these methods to real world problems. We found interesting information regarding Support Vector Machines, Decision Trees, Fuzzy systems, and Reinforcement Learning.

In terms of popularity, Support Vector Machines are right behind Neural Networks. They are starting to gain more recognizement due to the fact that SVMs implement the structural risk minimization which searches to minimize an upper bound of generalization error rather than minimize the training error implemented by traditional neural networks. In addition to this, SVMs solution is unique, optimal and absent of local minima. Despite each SVM type has its own characteristics, SVMs usually have less parameters to tune than ANN. All this make SVMs a very good option to realize financial forecasting.

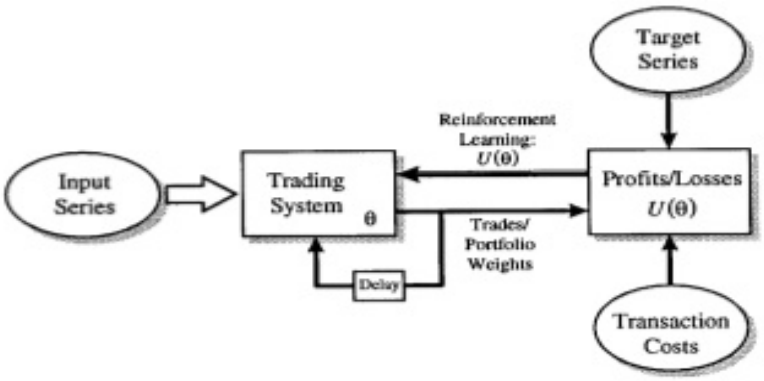


Figure 4.1: Recurrent Reinforcement Learner

Regarding other Machine Learning methods, very few literature is available about applying them to financial data. For Reinforcement Learning, we were able to find several documents describing how to use this method for price forecasting and stock trading. We had already decided to use Reinforcement Learning in BROMAS, but these documents encouraged us even more. One of special interest is [MOODY], in which a trading system is optimized using recurrent reinforcement learning.

4.2 Trading Agents

Another important source of information used to develop BROMAS comes from research about trading agents. Even though BROMAS agents don't trade, it's interesting to see what techniques are these systems using to improve the profit.

Regarding general trading, there are several trading competitions for autonomous agents. One of those is the Trading Agent Competition, in which agents play the role of travel agents who strive to arrange itineraries for a group of clients who wish to travel to a common destination and home again during a five-day period. Travel agents have to bid for Flights, Hotel rooms, and Entertainment activities in order to “construct” the most satisfying trip for the clients. The “Autonomous Bidding Agents” describes the strategies and lessons learned from this competition. Trading competitions related to stocks and financial markets are also hold, but very few information is made available.

Regarding stock trading agents, what all these research documents have in common is the use of the Sharpe ratio to base its performance evaluations. The Sharpe ratio is a reliable measure of the statistical significance of earnings and the trade-off between risk and return. It's one of the most widely-used measures of risk-adjusted return. Denoting the trading system returns for period *t* (including transaction costs) as R_t, the Sharpe ratio is defined to be:

$$S_t = \frac{Average(R_t)}{Standard\ deviation(R_t)}$$

Despite BROMAS is not a trading system but a decision support system, many of the techniques used for trading can be directly applied to improve our decision making process. In fact, one could say that recommending is a simplified version of trading, in which agents don't have to concern about the “positions” that are holding in the market.

4.3 MASST

The Multi Agent System for Stock Trading (MASST) is the only decision support system we found that resembles our BROMAS.

MASST is a middle-layer agent system between the demand side of information and the supply side of information. MASST realizes the following tasks:

- Stock information retrieval
- Stock status monitoring and risk management processing
- Buying and selling shares decision support process

From this point of view, the system looks very similar to what we had planned to do in BROMAS, plus the status monitoring functionality that we may include in future releases. The information retrieval process seems very complete, since it integrates various sources of information. MASST retrieves technical information (such as trading history and technical indicators), fundamental information (including new management, new products,...), market statistics information like the top 10 shares of maximum price upward, and even stock charts. Unfortunately, no information is provided about how this information is extracted from the internet nor how is it stored (besides saying that it's stored using Microsoft Access).

The stock status monitoring function reports any abnormal status to users, including price fluctuations, trading volumes, and price chart patterns. To provide Risk Management processing, MASST calculates profit/risk ratios based on the market status and the user's current investments.

The decision support process is, besides the information retrieval, the most interesting aspect of MASST for us. MASST goals are the same as ours, provide a support tool to help investors decide what to do and when. MASST uses a combination of user defined business rules and machine knowledge to provide buying and selling decision support.

MASST agent architecture is very interesting, mainly because they choose a different approach to the problem. Instead of grouping around functionalities (coordinator, information retriever, analyst,...) MASST has decided to mix together information retrieval and information analysis into the same agent. The overall agent system is depicted next:

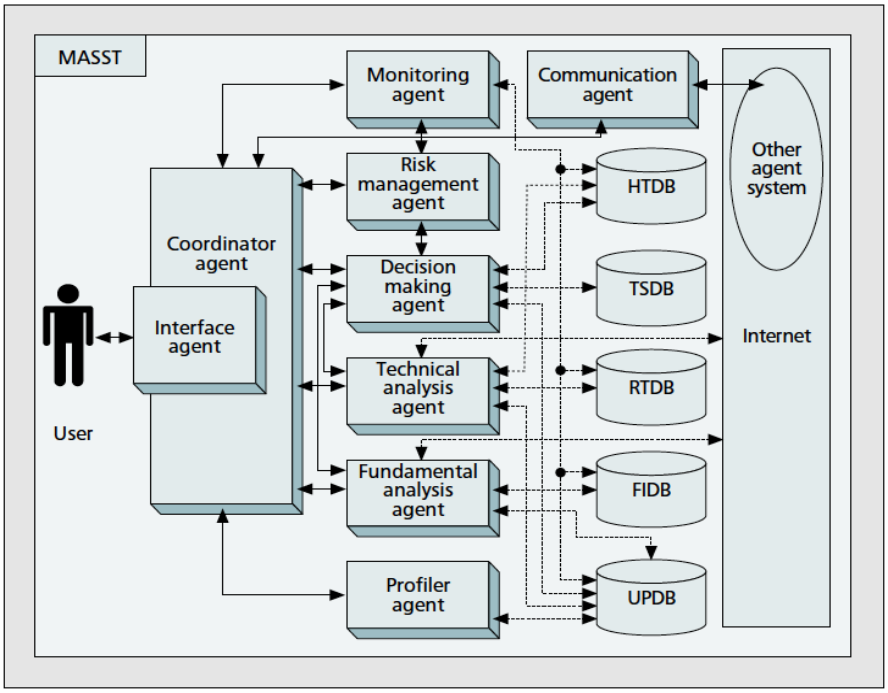


Figure 4.2: MASST Architecture

We're going to briefly explain next some of the aspects of the previous diagram that need some clarification: the data sources and the agents' tasks. As we can see in the previous figure, MASST uses a wide variety of Data Sources to realize its tasks.

The UPBD stands for User Profile Database. This data source is dynamic and shared among agents within the system. The UPDB includes information such as the user login information, the list of stocks he possesses, the list of stocks in which is interested, monitoring instructions, planned tasks, preferences, and privacy settings. As we can see, the system possesses a very exhaustive information about its users. Besides of this, each user also has its own personalized interface agent and the Trading Strategy Database (TSDB) which stores the user's private trading strategies. The History Trading Database (HTBD), the Real-time Trading Database (RTBD), and the Fundamental Information Database (FIDB) are shared among all agents. The exact information these Databases hold is not detailed.

Now, it's time to detail some of the agents present at MASST system. The profiler agent provides the mechanism by which a user's profile and TSDB are generated and maintained. What it's interesting is that profile agents uses information coming from the user and the environment to determine the interests of the user. The Technical Analysis and Fundamental Analysis agents are responsible for gathering the information from the Internet, process it into the HTDB/RTBD and FIDB databases respectively, and make recommendation to the Decision Making agent, who will combine these outputs with the investment strategies set in the TSDB to give a list of recommended stocks to buy, and suggest for each stock owned by the user wether to hold or sell. Finally, the Coordination agent is the responsible for planning and managing the rest of the agents. The Coordinator agent knows the capabilities of each agent present in MASST, decomposes a given task into subtasks, and dispatches each to the corresponding agent.

The communication architecture is quite complex, making use of what they call a dynamic blackboard system. This dynamic blackboard is the combination of a standard blackboard system, the data sources described previously and the coordinator agent. Communication between agents occurs within a private message area in the blackboard, using a specific communication language called MASST-ACL, which mixes the standard FIPA-ACL and XML.

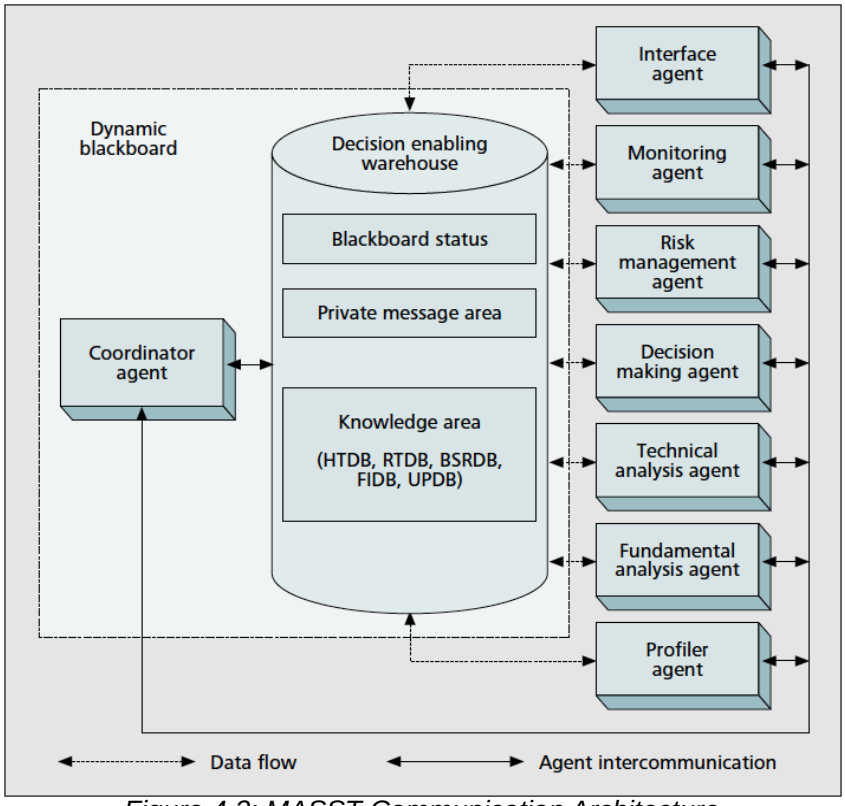


Figure 4.3: MASST Communication Architecture

Following I have included an example scenario of how MASST realizes a decision support task for buying shares. Almost all the agents in the system participate in this process. First, the Decision Making agent (DMA) asks the Risk Management agent (RMA) to check the risk for all the shares. After that, the DMA asks the Technical Analysis agent (TAA) checks the price and volume trend, rejecting those that are uninteresting. When the Technical Analyst has finished, the DMA asks the Fundamental Analysis agent (FAA) to check if the share price is correctly set. The FAA rejects all the shares that are overvalued. Finally, the DMA has a set of shares whose trend seems to be positive and whose price seems to be undervaluated.

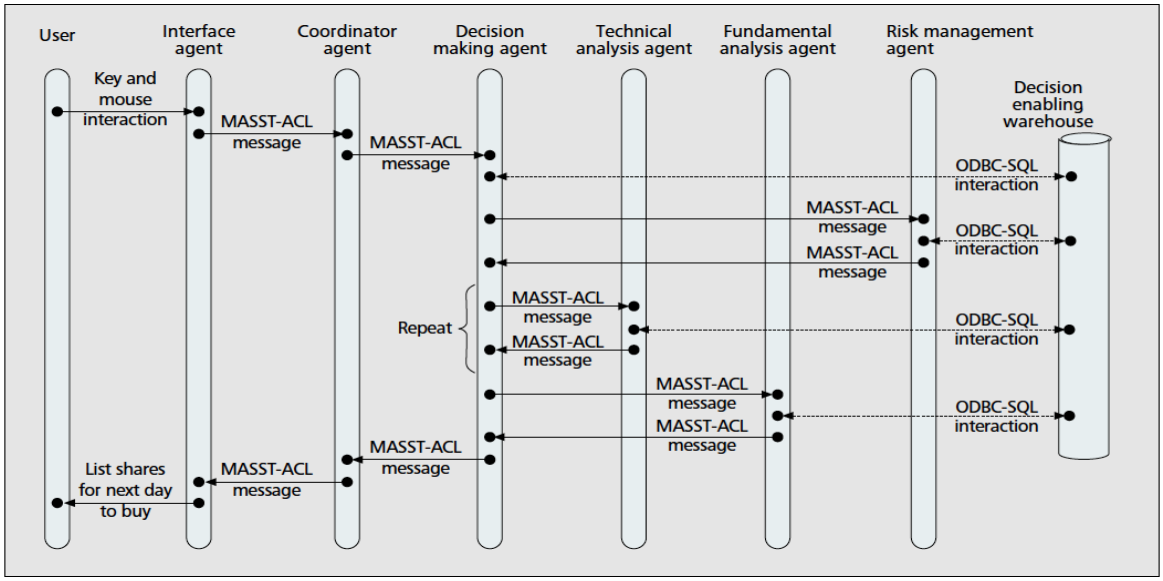


Figure 4.4: MASST Protocol for Recommending Shares

Chapter 5

Theoretical Introduction

This chapter comprises a brief overview of Agent Theory, with special emphasis on Agent-based Systems. First we introduce the concept of Agent from various points of view. Later, we will focus on Intelligent Agents and describe the different possible classes. Finally, we address their social dimension to create fault tolerant Multi Agent Systems.

However, before we start explaining what an Agent is, we must first try to define the term “Artificial Intelligence”. And before that, we must define what “real” Intelligence is. The problem is that there are several valid definitions, with some discrepancies between them. The term intelligence comes from the Latin *intellegere*, which means “to understand”. We could say then, that being intelligent is not only being able to adapt to one's environment, but to fully understand it too. From the *Mainstream Science on Intelligence*, an opinion piece signed by 52 Intelligence researchers, Intelligence can be defined as:

“A very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience.”

And now that we have, more or less, defined what Intelligence is, we can define what Artificial Intelligence is. And who better than John McCarthy, responsible for coining the term at the Dartmouth Conferences, to do so:

“It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.”

The next question to arise is: what does it mean to make intelligent machines/programs? Truth is, there are 4 possible answers to this question, depending on how success is measured and on what is the focus: mental processes or conduct. The 4 answers are:

- Systems that act like humans

The Turing Test, proposed by Alan Turing in his 1950 paper “Computer Machinery and Intelligence”, defines the abilities that a machine must have in order to be indistinguishable from a human being. Natural Language Processing, Knowledge Representation, Automatic Reasoning and Machine Learning are the abilities necessary to pass the test.

- Systems that think like humans

Before testing if a system thinks like a human, we must learn first how a human thinks. Cognitive science is a interdisciplinary field of research that mixes computational AI models with techniques from Psychology in order to study the nature of intelligence. Cognitive science tries to elaborate valid theories about how the human mind works.

- Systems that think rationally

The first approach to how to think rationally is found in Aristotle's syllogisms, a reasoning process that obtains a valid conclusion from valid premises. This was the beginning of Logic, a branch of Philosophy/Mathematics that studies the principles of valid demonstration and inference.

- Systems that act rationally

Acting rationally means to direct your activity towards achieving your goals. Just what Agents do.

5.1 What's an Agent?

The origin of the word comes from the Latin *agere,* which means “to do”. Because of this, we can define, in a very simplistic way, an Agent as an entity capable of action. But since this definition doesn't comply the standards of what a definition should be, we have to further extend it.

The term agent is used in numerous fields, from Law to Computer Science. Nut all this possible views of what an Agent is have some interesting points in common. The first shared property is that the term is used to refer to indermediaries such as Sports Agents or Administrative Agents. The Agent is thus an entity authorized to act on behalf of others.

Another shared property of all these “possible” agents is that they are, at least to a certain degree, autonomous entities. For example, a broker agent has autonomy to decide how and where to invest his clients' funds. Autonomy means that agents have to act without needing the intervention of humans or other agents. This implies that an autonomous Agent's behavior has to rely more on its experience with the environment than on its initial knowledge.

In addition to this, there are two additional features that are worth mentioning: *reactivity* and *proactivity.* Reactivity is, as its name suggests, how an agent perceives and reacts to changes in its environment. But just reacting to changes is not enough to achieve your own goals. Because of this, an agent's behavior can't be only dictated by its environment and by its interactions. An agent has to take the initiative and *proactively* pursue its objectives.

But all these properties alone are not enough for making the agent *intelligent*. There's one property missing: *rationality.* As we said earlier, *rationality* means to act in order to fulfill your objectives. Thus, a rational agent realizes the action that maximizes its performance, that is, the one that satisfies its goals.

Learning is also another important feature that an intelligent agent may possess. This feature allows the Agent to become even more Autonomous by improving its adaptability to new/changing environments.

Other important characteristics attributed to agents are mobility and cooperation. Mobility allows the agent to move to other environments, from a Computer Science point of view, this could be understood as moving through a network, from one machine to another. Real world is a Multi Agent environment: we can't achieve our goals without taking others into account. In addition, there are complex tasks that can only be solved by cooperating with other agents. Because of this, agents have to be social aware, they must be able to interact with other agents regardless of the communication channel used.

In this thesis, we will focus in developing intelligent software agents, defined in [Green97] as “a computational entity which:

- acts on behalf of other entities in an autonomous fashion
- performs its actions with some level of proactivity and/or reactiveness
- exhibits some level of the key attributes of learning, cooperation and mobility”

5.2 Agent Types

According to [RussellNorvig] there are 4 basic types of agents which embodies the principles that underlie in almost all intelligent systems. They are, in order of increasing complexity:

- simple reflex agents
- model-based reflex agents
- goal-based agents
- utility-based agents

All of which can be transformed into learning agents.

5.2.1 Simple Reflex Agents

The simple reflex agent is the most simple type of agent. This kind of agents choose actions based solely on actual perceptions of the environment, ignoring the historical perceptions.

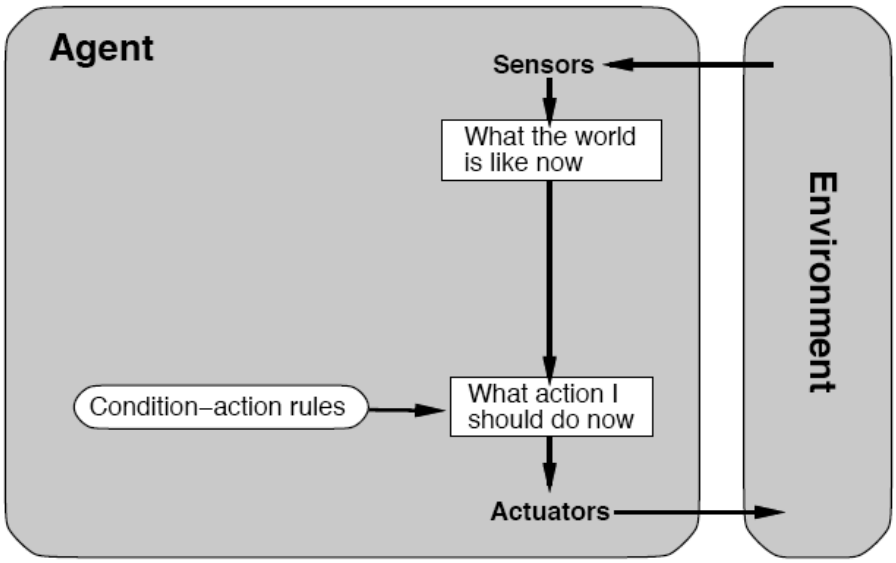


Figure 5.1: Simple Reflex Agent

5.2.2 Model-based Reflex Agents

The most effective way agents can deal with partial visibility is to store information about the parts of the world they can't see. That is, the agent has to store some kind of internal state that depends on the history of perceptions in order to reflect some of the non observable aspects of the actual state.

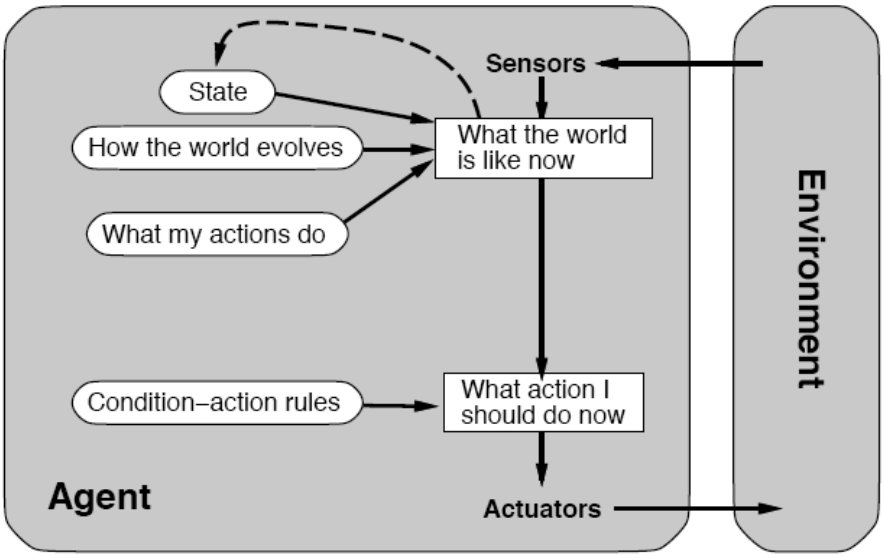


Figure 5.2: Model-based Reflex Agent

5.2.3 Goal-based Agents

Knowledge about the current state of the world is not always enough to decide what to do next. In addition to the description of the current state, the agent needs some kind of information about its goal that describes which are the desirable situations.

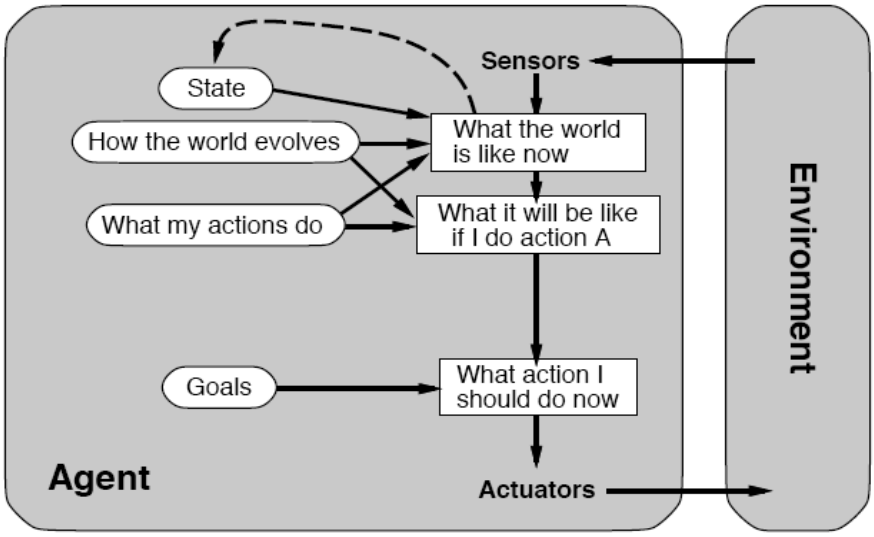


Figure 5.3: Goal-based Agent

5.2.4 Utility-based Agents

Goals on their own are not really enough to generate high quality behavior in the majority of environments. Goals only provide a binary distinction between the states of “happiness” and “sadness”, when a more general efficiency measure should allow a comparison between different states of the world accordingly to the exact “happiness” value that the agent reaches when it's in one state or another. This “happiness” value is known as **utility**.

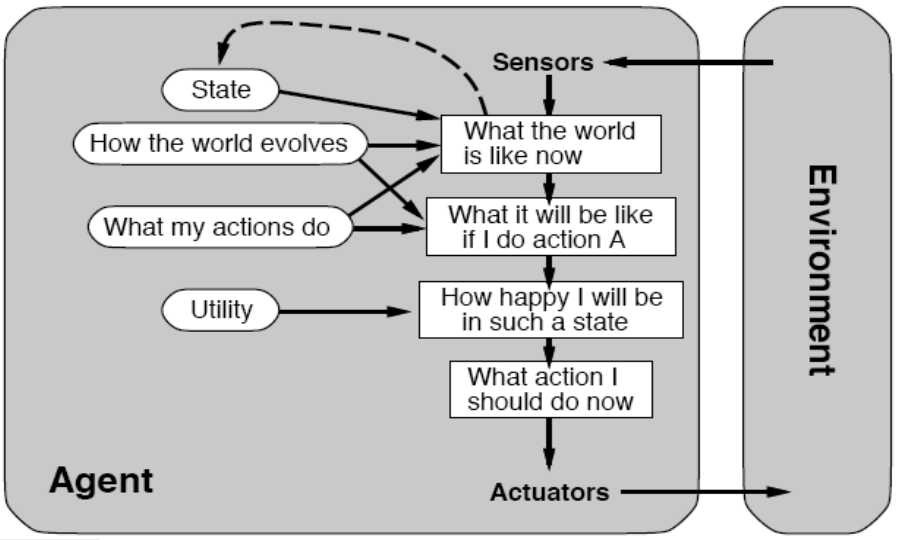


Figure 5.4: Utility-based Agent

An utility function associates every environment state into a real number that represents the level of “happiness” that the agent reaches in that state. This function allows the agent to solve situations when there are conflicting goals or situations where the agent has to choose which goal to pursue.

5.2.5 Learning Agents

In order to construct intelligent machines, Turing proposed building machines that are able to learn and teach them afterwards. This method is much faster than coding an intelligent machine from scratch. Besides this, learning has other advantages: it allows the agent to operate in initially unknown environments and it makes the agent more reliable in comparison to just using initial knowledge. A learning agent can be divided into 4 conceptual components:

- Learning element, responsible for improving the agent behavior.
- Performance element, responsible for selecting external actions.
- Critic, responsible for telling the learning element how good the agent decisions are.
- Problem generator, responsible for suggesting actions that will lead the agent to new and informative experiences.

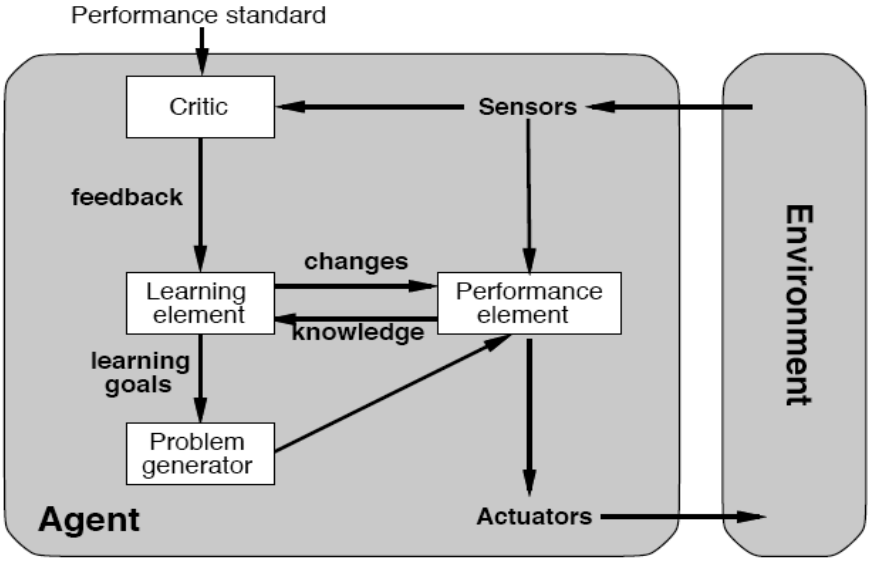


Figure 5.5: Learning Agent

5.3 Practical Reasoning Agent – The BDI Model

Practical reasoning is reasoning directed towards actions, it's the process of figuring out the next action to take:

"Practical reasoning is a matter of weighing conflicting considerations for and agains competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes" Bratman

In humans, practical reasoning consists of two activities:

- Deliberation, when we decide what state we want to achieve.
- Means-end reasoning, when we decide how to achieve these objective states.

When we deliberate, we "create" intentions that are what moves us to act towards achieving our goals.

The Belief-Desire-Intention Model is a model of practical reasoning developed by Michael Bratman as a way of explaining future-directed intention. Nowadays, it's widely used for programming intelligent agents.

A rational agent has bounded resources, limited understanding and incomplete knowledge of what happens in the environment it lives. Such an agent has beliefs about the world and desires to satisfy, driving it to form intentions to act. An intention is a commitment to perform a plan. In general, a plan is only partially specified at the time of its formulation since the exact steps to be performed may depend on the state of the environment when they are eventually executed. The activity of a rational agent consists of performing the actions that it intended to execute without any further reasoning, until it is forced to a revision of its own intentions by changes to its beliefs or desires. Beliefs, desires and intentions are called mental attitudes (or mental states) of an agent.

BDI agents depart from purely deductive systems and other traditional AI models because of the concept of intentionality, which significantly reduces the extent of deliberation required partially by eliminating choices inconsistent with current intentions. BDI has demonstrated to be well suited to model certain types of behavior, such as the application of standard operational procedures by trained staff. It has been successfully adopted in fields as diverse as simulation of military tactics, application of business rules in workflows, and diagnostics in telecommunication networks.

Based on previous research and practical application, Rao and Georgeff have described a computational model for a generic software system implementing a BDI agent. Such a system is an example of event-driven programs. In reaction to an event, for instance a change in the environment or its own beliefs, a BDI agent adopts a plan as one of its intentions. Plans are precompiled procedures that depend on a set of conditions for being applicable. The process of adopting a plan as one of the agent's intentions may require a selection among multiple candidates.

The agent executes the steps of the plans that it has adopted as intentions until further deliberation is required; this may happen because of new events or the failure or successful conclusion of existing intentions. A step of a plan can consist of adding a goal (that is, a desire to achieve a certain objective) to the agent itself, changing its beliefs, interacting with other agents, and any other atomic action on the agent's own state or the external world.

To summarize, the BDI model consists of:

- Beliefs: represent the information state of the agent, in other words, its beliefs about the world including itself and other agents. Beliefs can also include inference rules, allowing forward chaining to lead to new beliefs. Using the term belief, rather than knowledge or similar, recognizes that what an agent believes may not necessarily be true and, in fact, may change in the future.

- Desires: represent the motivational state of the agent, also referred to as goals. They represent objectives or situations that the agent would like to accomplish or bring about. Usage of the term goals adds the further restriction that the set of goals must be consistent. One should not have concurrent conflicting goals even though they could both be desirable.

- Intentions: represent the deliberative state of the agent. In other words, what the agent has chosen to do. Intentions are desires to which the agent has to some extent committed, which in implemented systems means that the agent has begun executing the related plan.

- Plans: sequences of actions that an agent can perform in order to achieve one or more of its intentions. In Bratman's model, plans are initially partially conceived, with details being filled in as they progress. Plans can include other plans. For example, a plan to go for a ride with my car may include a subplan to find the keys.

5.4 Multi Agent Systems

A Multi Agent System is a system composed by a set of agents who interact with each other to perform complex tasks. Agents are defined as computer programs capable of taking their own decisions with no external control, based on their perceptions of the environment and the objectives they aim to satisfy. Multi Agent Systems are an interesting field of research due to their ability to:

- solve problems that are too large for a centralized single agent to do due to resource limitations or the sheer risk of having one centralized system
- allow for the interconnecting and interoperation of multiple existing legacy systems
- provide solutions to inherently distributed problems
- provide solutions which draw from distributed information sources
- provide solutions where the expertise is distributed
- enhance speed, reliability, extensibility and the ability to cope with uncertainty
- offer conceptual clarity and simplicity of design

Because of this, online trading is an example of problem that is appropriate to be solved by using a Multi Agent System.

5.4.1 Open and Closed Systems

In Multi Agent Systems there are two main design approaches: *top-down* and *bottom-up.*

In the *top-down* approach, the system designer decomposes the original problem into smaller and simpler sub-problems until it is possible for an agent (or group of coordinating agents) to solve it. This relies on the *benevolent agent assumption* that, as its name suggests, only applies when agents have common or non-conflicting goals. It's in this scenario where the fulfillment of the agents' individual goals will indirectly lead to the resolution of the complex task addressed by the whole system. Since there is no conflict between agents' and system's goals, the sum of individual efforts results in the solution of the complex task. Agents carry out a cooperative problem solving.

Truth is, this approach has some important drawbacks when applied to Multi Agent Systems. This design requires to have everything (or almost everything) accounted for at design time. Possible states, transitions, interactions,... everything has to be defined. This results in a *closed system*, where an element outside the ones that were defined at design can't enter the system.

Since the interaction space is known a priori, the system doesn't allow interactions outside the original domain. As a result of this, the social level of the system is weak and agents' autonomy is reduced. Agents are bounded by the decisions made at design time. This is the cost for having the correctness, stability, efficiency... of the system anticipated and enforced at design time.

For agents to exploit their full potential, they must be part of an *open system*, a system were not all factors are known at design time. It's in this scenario where the properties of the agent paradigm distinguish it from other software paradigms.

Unfortunately, the design of an open system is not trivial, as one would suspect. In the bottom-up approach designers must place the emphasis on building agents rather than building a system. This approach does not resort to a recursive decomposition of the original problem into subproblems, but an aggregation of agents with certain skills or services instead. Through interaction between these “specialized” agents, the complex task can be attained. Thus, the goals of the agents are not subproblems of the complex task. For example, this design approach can be used to solve routing and load balancing in telecommunication networks with a myriad of very simple agents following very simple rules.

This approach is well suited for open and distributed domains, where agents can interact with other agents that where not previously known. Ironically, this openness is the main responsible for the downsides of the bottom-up approach:

- How can an agent know about the services that other agents offer?
- How can an agent interact with agents who are not designed for interacting with it?
- What are the guarantees that another agent will do its task properly?

These questions arise due to the fact that in open systems the *benevolent agent assumption* can't be applied.

5.4.2 Social Mechanisms

It was stated in the last section that in open domains, behavior of the entities escape the control of the designers of the system. There's no guarantee that one agent is acting as it should (from a designers' point of view). Due to this, we concluded saying that in open systems the *benevolence assumption* doesn't apply. A system that assumes that agents will always act according to the common good but has no means to enforce proper behaviors will certainly fail. This is true both in real and artificial scenarios. Agents introduced by the designers are expected to do what they where designed to do, but this can't be ensured from alien agents. This “outsiders” escape from the designers' control.

In summary, agents will only look for the common good as long as it serves their particular interest. So, how do we enforce beneficial behaviors without “closing” an open system? Using the same mechanisms that work for humans: social mechanisms. We humans do act collaboratively, not because of altruism, but because of social mechanisms that bound us to do so. We have created these mechanisms to reduce the complexity and uncertainty that exists in human societies. With the growing complexity of Multi Agent Systems and other artificial societies, some of this social mechanisms have been ported to be applied to computational entities. Among these mechanisms we find: institutions, norms, conventions, trust and reputation. In the following sections, we'll describe some of these social mechanisms in detail.

Ironically, porting real world social mechanisms into artificial societies such as Multi Agent Systems can contribute to a better understanding of the processes occurring in complex social systems. With the use of Agent-based simulations, new insights and findings can provide better understanding of the social processes that take place in society. And all this knowledge, will be brought back to the Agent field to create more stable and efficient Multi Agent Systems.

5.4.3 Trust and Reputation

Social mechanisms such as *trust* and *reputation* help agents in their decision making process, and are intended to help the system run smoothly with agents willing to cooperate. *Trust* and *reputation* have been extensively studied in the field of social sciences, here we'll introduce the “computation version” of these two concepts.

Trust

Trust is a firm belief in the reliability, truth, ability,... of someone or something. A *prediction* of reliance on an action, based on what a party knows about the other party. The degree to which one party trusts another is a measure of belief in the honesty, benevolence and competence of the other party. Don't confuse benevolence with good character, vices or morals, trust can be created between criminals for instance.

Gambetta defined trust as a particular level of subjective probability with which an agent will perform a particular action before the action is performed. Trust between two agents is built on the outcome of interactions, usually positive interactions in which both agents benefit from the interaction. Certain level of trust between agents is required to maintain a cooperative regime in open systems. Trust minimizes the amount of uncertainty an agent faces when interacting with another.

Reputation

Reputation is a social evaluation of the public toward a person, group of people or organization. It is known to be a ubiquitous, spontaneous and highly efficient mechanism of social control in natural societies. Because of this, *reputation* is one of the social mechanisms that has been “ported” to artificial societies.

Reputation helps manage the complexity of societies by reducing the number of agents worth interacting with, ensuring (to a certain extent) a positive interaction and promoting cooperation under the menace of getting a negative review (and thus, lowering its reputation value).

All mechanisms have drawbacks, including reputation. The filtering induced by reputation can lead to a suboptimal state of the system where latecomers, despite offering better reliability/services, have no visibility. Designers have to assure that all agents have a fair starting point and that malicious agents are unable to change their identity for a new one too easily.

In addition to this, the measure of reputation could be tampered, specially when the reputation of an agent is based on explicit feedback given by other agents. A malicious agent could undermine a competitor's reputation or artificially increase its reputation with the help of cooperator agents.

5.4.4 Communication

At the beginning of this section, it was stated that Multi Agent Systems are made to solve complex tasks using a set of agents. In order to do this, agents need to coordinate (cooperate, compete) with others. And to do that, agents need to communicate. Providing infrastructure to communicate allows the agents to:

- request actions or services to other agents
- ask for information
- share beliefs with other agents
- coordinate with other agents

Communication protocols can be split into several levels. The lower levels define how the messages are sent and received. The middle levels specify how the message is structured and expressed. Finally, the upper levels specify the semantics, or meaning, of messages. The meaning of a message is given not only by its content, but also by the type of message. In agent communication, levels are defined in the following way:

- Message semantics, composed by the message type (which gives the intentionality), the message content (which contains the information) and the ontology used (which tells what the message is referring to).
- Message syntax, composed by the message structure and the content language used to codify its content.
- Interaction protocol, composed by the agent protocols, which define how conversations/dialogues between agents are structured.
- Transport protocol, which defines how messages are actually sent and received.

In general, agents can neither force other agents to perform some action, nor modify the internal state of other agents. What they can do is perform communicative actions in an attempt to influence other agents appropriately.

Speech acts

The Speech Act Theory treats communication as action. Speech actions are performed by agents just like other actions, in the furtherance of their intentions. This theory is generally recognized to have begun with the work of Austin. He noted that certain class of natural language utterances, or *speech acts*, had the characteristics of actions, in the sense that they change the state of the world. For example, saying “I now pronounce you man and wife” under the appropriate circumstances clearly changes the state of the world.

Austin identified a number of *performative verbs*, which correspond to various different types of *speech acts*, like *request, inform, promise,...* In addition, Austin distinguished three different aspects of speech acts:

- Locution, the act of making an utterance. For example, saying “Please make some tea”.
- Illocution, the action performed in saying something. E.g. “He requested me to make some tea”.
- Perlocution, the effect of the act on those who receive the utterance. E.g. “He got me to make tea”

Speech act theory helps define the type of message by using the illocutionary force, which constraints the semantics of the communication act itself. The sender's intended communication act is clearly defined, and the receiver has no doubt as to the type of message sent, independently of the message contained. Doesn't matter if the message is ambiguous, doesn't have a simple response,... the communication protocol has to clearly identify the type of message. This constraint clearly simplifies the design of communicative software agents.

Agent Communication Languages: FIPA-ACL

The Speech act theories have directly informed and influenced a number of languages that have been developed specifically for agent communication. Agents use a set of predefined performatives in order to communicate their intentions, allowing the receiving agent to interpret its content in a proper way. There are two predefined sets used in Multi Agent Systems: the Knowledge Query and Manipulation Language (KQML) and the FIPA-ACL. We'll focus in the latter.

The FIPA-ACL is an Agent Communication Language created by FIPA, an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. The Foundation for Intelligent Physical Agents has played a crucial role in the development of agent standards and has promoted a number of initiatives and events to contribute to the development and rise of agent technologies.

The FIPA-ACL is the centerpiece of FIPA's work in developing standards for agent systems. FIPA-ACL defines an 'outer' language for messages, with 20 defined performatives for defining the intended interpretation of messages and without a mandatory language for message content. This allows to use almost any content language, like KIF, RDF,... or FIPA-SL, a content language designed for agents with BDI architecture.

Interaction Protocols

Performatives are not enough to establish a correct communication between agents, and so, they're part of a protocol specification. A protocol is a conversation between agents which follows some rules defining which performatives to use and when in order to achieve a given goal. Each protocol defines the sequencing of messages in a given dialogue as a finite-state diagram, making easy for agents to keep the current state of a dialogue and to know which utterances follow. Each protocol is designed for a specific type of dialogue, it's necessary to choose carefully which one to use.

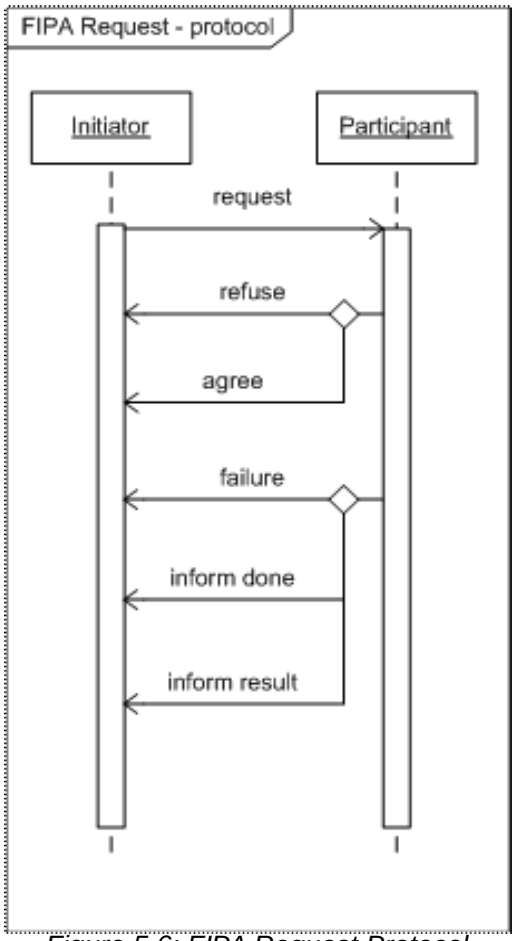


Figure 5.6: FIPA Request Protocol

5.5 Machine Learning

Machine Learning is a discipline within Artificial Intelligence that is concerned with the design and development of algorithms that allow computers to learn based on data. This definition brings out two important questions: What do we understand as “learning”? What does it mean that a machine learns?

We will first answer the first question to try to find an answer for the second. The dictionary defines “to learn” as follows:

- to get knowledge of by study, experience of being taught
- to become aware by information or from observation
- to commit to memory
- to be informed of, ascertain
- to receive instruction

These meanings though have some shortcomings when it comes to talking about computers. For the first two definitions, it is virtually impossible to test whether learning has been achieved or not. How can we test if a machine has got knowledge of something? If we tried to solve this dilemma by asking questions to the machine, we wouldn't be testing its ability to learn but its ability to answer our questions. The second definition is great for starting a philosophical debate: can machines be aware, or conscious? How can we know if a machine has become aware of something? And well, the last three fall short of what machine learning might be. They're trivial tasks for a computer. So, what kind of learning is Machine Learning? A suitable answer to this would be:

> “Things learn when they change their behavior in a way that makes them perform better in the future.”

Machine Learning algorithms are organized into a taxonomy, based on the type of problem they have to face. The three main types are the following:

- Supervised Learning, in which the algorithm generates a function that maps inputs to desired outputs. The training data comprises examples of the input vectors along with their corresponding target vectors. If the task is to assign each input vector to one of a finite number of discrete categories, it is called *classification.* If the desired output consists of one or more continuous variables, then the task is called *regression.*

- Unsupervised Learning, in which the training data consists of a set of input vectors without any corresponding target values. The goal in these kind of problems may be to discover groups of similar examples within the data, called *clustering*, or to determine the distribution of data within the input space, known as *density estimation*, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *data visualization.*

- Reinforcement Learning, in which the algorithm learns a policy of how to act given an observation of the world in order to maximize a reward. The learning algorithm is not given examples of optimal outputs, in contrast to *supervised learning,* but must instead discover them by a process of trial and error.

In the following subsections we'll introduce the techniques that are currently being used in the BROMAS system. There are 3 *supervised learning* algorithms and one *reinforcement learning algorithm.* In the future, new techniques will be added to try to provide more insightful information to the system.

5.5.1 Artificial Neural Networks

The term 'neural network' has its origins in attempts to find mathematical representations of information processing in biological systems. The study of Artificial Neural Networks (ANNs) has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons. In rough analogy, ANNS are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs and produces a single value output. Since there are several classes of Artificial Neural Networks, I have chosen to focus in the one that will be used in this system, and also the one that has proven to be of greatest practical value, the *multilayer perceptron.*

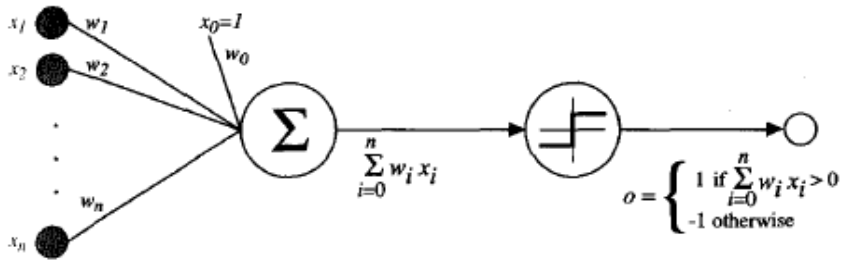


Figure 5.7: A Perceptron

A *perceptron* takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. We can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances. The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for those instances lying on the other side. A perceptron is, thus, a linear classifier.

The w_i elements visible in the image are the weights, a real-valued constant that determines the contribution of the input x_i to the perceptron output. Learning a perceptron consists in choosing values for the weights . Therefore, the space of candidate hypothesis is the set of all possible real-valued weight vectors. To solve this learning problem there are several algorithms, here we'll be describing two of them: the perceptron rule and the delta rule.

The Perceptron Training Rule

The algorithm begins with random weights. The perceptron is then applied iteratively to each training example, modifying the weights whenever the perceptron misclassifies an example. The process is repeated iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the *perceptron training rule*, which updates the weight w_i according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta (t - o) x_i$$

As we can see, the *perceptron training rule* is an error correcting rule. Here *t* is the target output for the current training example, *o* is the output generated by the perceptron and is a positive constant called the learning rate. This constant moderates how much the weight value is changed at each step. Sometimes is made to decay as the number of iterations increases. This method is proven to converge within a finite number of iterations provided that the training examples are linearly separable and provided that the learning rate is sufficiently small.

The Delta Rule

As seen in the previous section, the *perceptron training rule* fails to converge if the examples are not linearly separable. To overcome this difficulty, the *delta rule* was designed. If the examples are not linearly separable, the rule converges toward a best-fit approximation to the target concept.

The idea behind the *delta rule* is to use *gradient descent* to search the hypothesis space of possible weight vectors to find the ones that best fit the training examples. The algorithm will attempt to minimize the error in the output of the perceptron, which is:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where D is the set of training examples td ,is the target output for training example d, and o_d is the obtained output for training example d. As we can see, the output error is simply half the squared difference between the target output and the linear unit output, summed over all training examples.

The *gradient descent* search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps. At each step , the weight vector is altered in the direction that produces the steepest descent along the error surface. This process continues until the global minimum error is reached.

In order to do that, we calculate the partial derivative of the error with respect to each weight. Making the training rule as follows

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

The negative sign is present because we want to move the weight vector in the direction that decreases E. If we differentiate E from the previous equation we can rewrite the weight update rule as

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

where x_{id} denotes the single input component x_i for training example d .

Since the error surface contains only a single global minimum, the algorithm will converge to a weight vector with minimum error, independently of the training examples used, and given a sufficiently small learning rate.

After this, it's time to introduce the *Multi Layer* Perceptron. A *Multi Layer Perceptron* is a feed-forward artificial neural network that uses three or more layers of perceptrons with nonlinear activation functions. The combination of layers of neurons allows the MLP to distinguish data that is not linearly separable. For being able to do this, we must use a differentiable threshold unit, one whose output is a nonlinear function of its inputs, but whose output is also a differentiable function of its inputs. One solution is the sigmoid unit, similar to the perceptron, but based on a smoothed, differentiable threshold function. This unit computes its output *o* as

$$o = \sigma(\vec{w} \cdot \vec{x})$$

where

$$\sigma(y) = \frac{1}{(1 + e^{-y})}$$

is the sigmoid or logistic function.

To train such a network the *backpropagation algorithm* is used to learn the weights given a network with a fixed set of units and interconnections. It employs a *gradient descent* to attempt to minimize the squared error between the output and the target values. Since we are considering networks with multiple output units we have to redefine E:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

where outputs is the set of output units. As in the Delta rule, the learning problem is to search in a large hypothesis space defined by all possible weight values for a hypothesis that minimizes E. Unlike the Delta rule, the error surface can have multiple local minima, meaning that gradient descent is guaranteed only to converge toward some local minimum. For each training example, we propagate the input forward through the network. Then we calculate the error for each network output unit and propagate the errors backward through the network, calculating the local error for each neuron.

Once we have the local errors, we update each network weight using

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

where δn denotes the error term associated with unit n. This will be repeated until a certain termination condition is met. Once may choose to halt after a fixed number of iterations, or once the error falls below some threshold,... The choice is a very important one, since too few iterations can fail to reduce error sufficiently, and too many can lead to overfitting the training data.

5.5.2 Support Vector Machines

A Support Vector Machine is a supervised learning method used for solving problems in classification, regression and novelty detection. An important property of SVMs is that the determination of the model parameters corresponds to a convex optimization problem, and thus any local solution is also a global optimum. SVMs tries to find the solution which classifies the training data set exactly with the smallest generalization error. This problem is approached through the concept of the *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples. In SVMs, the decision boundary is chosen to be the one for which the margin is maximized.

Given a linearly separable data set, we wish to find the "best" of all possible hyperplanes that separates the data. We would like there to exist weights *w* and bias *b* such that:

$$\begin{aligned} w \cdot x_i + b &\geq 1 \qquad \forall\, x_i \in Class\,1 \\ w \cdot x_i + b &\leq -1 \qquad \forall\, x_i \in Class\,2 \end{aligned}$$

To find the plane furthest from both sets, we will maximize the margin between two parallel supporting planes. A plane supports a class if all points in that class are on one side of that plane. The idea is that the support planes are "pushed" apart until they "bump" into a small number of data points, called support vectors, from each class. The support vectors are the only needed points to solve the problem. If we deleted all points except the support vectors, the result would be the same. The distance or margin between these supporting planes $w \cdot x + b = 1$ and $w \cdot x + b = -1$ is $1/\|w\|$. The optimization problem then simply requires that we maximize $1/\|w\|$, which is equivalent to minimizing

$$\frac{1}{2}\|w\|^2$$

subject to the constraints given by $y_i(w \cdot x_i + b) \geq 1, \qquad 1 \leq i \leq N$

This is an example of quadratic programming problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints. In order to solve this constrained optimization problem, we introduce Lagrange multipliers to obtain a dual representation of the maximum margin problem in which:

$$\min_{\alpha} \quad \tfrac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j x_i \cdot x_j - \sum_{i=1}^{m} \alpha_i$$

$$s.t. \quad \sum_{i=1}^{m} y_i \alpha_i = 0$$

$$\alpha_i \geq 0 \quad i = 1,..,m$$

the vectors for which $a_i > 0$ are the Support Vectors.

If we have to solve a non linearly separable problem, the decision boundary in the input space won't be lineal. In order to separate the data correctly, we have to map the data into a space of higher dimensionality called the feature space. The non linear functions used to do this mapping are the kernel functions. The resulting algorithm is formally similar, except we substitute in the objective the original dot product with a kernel evaluation. This way, by changing kernels we can get different highly nonlinear classifiers. All the benefits of the original linear SVM method are maintained.

Sometimes, even in the features space the data is not linearly separable. To solve this situation, SVMs have been extended with *soft margin*. Soft Margin softens the restrictions of the original SVM by allowing mislabeled examples. A nonnegative slack or error variable ξ_i is added to each constraint and then added as a weighted penalty in the objective:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i$$

subject to $\quad y_i(w \cdot x_i + b) + \xi_i \geq 1, \qquad 1 \leq i \leq N$

$$\xi_i \geq 0$$

where C is a positive constant that controls the trade-off between the slack variable penalty and the margin. In the limit, $C \to \infty$, we will recover the earlier SVM for separable data. The key advantage of this approach is that the slack variables vanish from the dual problem, with the constant C appearing only as an additional constraint on the Lagrange multipliers.

5.5.3 Decision Tree

Decision Tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. It's robust to noisy data and capable of learning disjunctive expressions.

Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. Decision Trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. At each node, the attribute specified by it is tested and the branch corresponding to the value of the attribute in the given example is chosen.

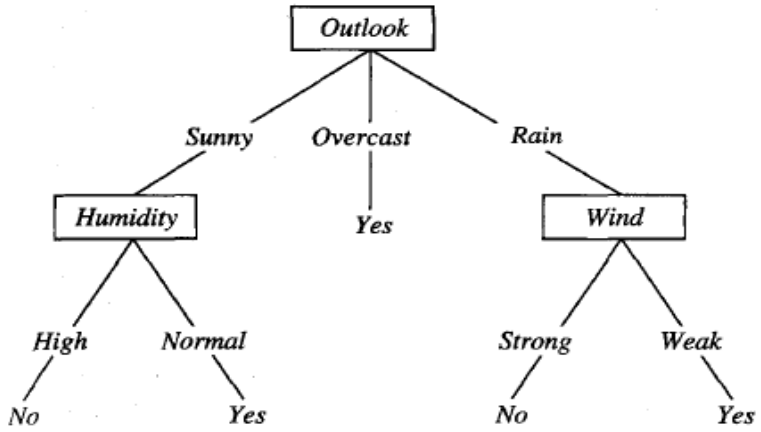


Figure 5.8: A decision tree for the concept PlayTennis

In general, Decision Trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions. For example, the decision tree shown corresponds to the expression:

(Outlook = Sunny Humidity = Normal)
V (Outlook = Overcast)
V (Outlook = Rain Wind = Weak)

All these features make Decision Trees readily interpretable by humans because they correspond to a sequence of binary decisions applied to the individual input variables.

In order to learn such a model from a training set, we have to determine the structure of the tree, including which input variable is chosen at each node to form the split criterion as well as the value of the threshold parameter for the split. It's also necessary to determine the values of the predictive variable within each region. Most algorithms developed for learning Decision Trees come from a base algorithm that employs a top-down, greedy search through the space of possible decision trees. In this section we'll briefly explain the ID3 Algorithm (Quinlan 1986).

The ID3 algorithm makes a Hill-climbing search through the space of decision trees. ID3 performs a simple-to-complex search, beginning with the empty tree, then considering progressively more elaborate hypothesis in search of a decision tree that correctly classifies the training data.

The central choice in ID3 is selecting which attribute to test at each node in the tree. We want to select the attribute that is most useful for classifying examples. The attribute is chosen using an heuristic function. In this case, a measure commonly used in Information Theory: *information gain*. In order to understand what *information gain* is, we have to briefly introduce the information theory.

Information theory studies the quantification of information. This includes, among others, mechanisms to codify messages and the study of the transmission costs. Given a set of messages $M = [m_1, m_2, m_3,..., m_n]$, each with a probability $P(m_i)$, we can define the *quantity of information* contained in a message as:

$$I(M) = \sum_{i=1}^{n} -P(m_i)\log(P(m_i))$$

This can be interpreted as the information needed to distinguish between the messages in M.

If we treat the classes in which we want to classify the data as the messages and the proportion of each class in the data as its probability, we can see a Decision Tree as a codification that distinguishes between different classes. The ID3 algorithm searches for the minimum codification that is able to classify the data.

To do so, we must evaluate the attributes at each level of the tree to find which is the one that minimizes the codification, the one that generates a smaller tree. This attribute will be the one which minimizes the impurity of the subgroups of data that it generates, or the one that minimizes the quantity of information that is still "missing". And that is what the *entropy* is. The expected reduction in entropy caused by partitioning the examples according to an attribute is the *information gain*, the measure that ID3 uses.

First of all, ID3 chooses the decision attribute for the root node and creates branches below the root for each of its possible values. Training examples are sorted to each new descendant node. If all the examples of a node have the same target attribute value (i.e. their entropy is zero), then the node is a leaf. In contrast, the descendants that have non zero entropy will be further elaborated using the same process as with the root node, this time using only the training examples associated with that node. Attributes that have been incorporated higher in the tree are excluded, making that any given attribute can appear at most once along any path through the tree. This process continues for each new node until either of two conditions is met:

- Every attribute has already been included
- The training examples associated all have the same target attribute value

All this is directly applicable when the training data only contains qualitative attributes. If the data includes quantitative attributes, another step is needed: we have to discretize the attribute into two or more values. For each attribute, we evaluate all possible splits using the *information* gain measure. The best partition will be used during the attribute selection step to compare its performance with other attributes. This additional step increases the temporal cost of the algorithm.

ID3 in its pure form performs no backtracking in its search. Once it selects an attribute to test at a particular level in the tree, this choice is never reconsidered. Therefore, it is susceptible of converging to a locally optimal solutions that are not globally optimal. In order to avoid this, ID3 was extended to do *pruning*. *Pruning* is a set of techniques that search (and eliminate) those nodes that, when deleted, reduce the error of the tree. *Pruning* can be done during the construction of the tree (*prepruning)* or after the construction (*postpruning*). Prepruning consists in applying statistical tests to establish if its beneficial to make a new partition. A new partition won't be made if it doesn't improve the classes' distinguishability. In postpruning, the prediction error of each node is estimated in order to calculate the improvement of deleting a certain node. A node will be deleted if its prediction error is lower than the weighted sum of its descendants' errors.

5.5.4 Reinforcement Learning

Reinforcement Learning is a sub-area of Machine Learning that provides a computational approach to understanding and automating goal-directed learning and decision-making. The most important distinguishing feature of Reinforcement Learning is its emphasis on learning by the Agent from direct interaction with its environment, without relying on supervision or complete models of the environment. That is, the Agent learns what to do – a mapping from situations to actions – by maximizing a numerical reward signal. The Agent is not told which action has to take, but instead has to find the most suitable action by realizing a trial-and-error search among all possibilities. These actions affect the environment, and thus the next situation, meaning that the action chosen will affect not only the immediate reward but also all the subsequent rewards from that point onwards. Taking in consideration this aspect, the delayed reward, is another distinguishing feature of this approach.

Reinforcement Learning uses a formal framework defining the interaction between a Learning Agent and its environment in terms of states, actions and rewards. The environment comprises everything that is outside the Agent. Both entities interact continually, the Agent selects actions and the environment responds to those actions by presenting new situations to the agent. The Environment is also responsible for the rewards that the agent receives after realizing an action. The interaction is produced during a sequence of discrete time steps. At each step, the agent receives a representation of the environment’s current situation, the *state*, and on that basis selects an *action* among all the actions available in that state. In the next time step, the agent receives the reward, mainly as a consequence of the action taken previously, and receives the environment’s new state.

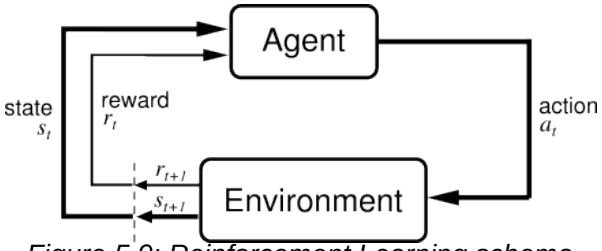


Figure 5.9: Reinforcement Learning schema

Since the agent makes its decisions as a function of the environment's state, the state representation chosen should contain more than the immediate sensations retrieved from the current situation. It should be able to summarize past sensations in a way that retains all relevant information for the agent. A state signal that succeeds in doing this is said to be *Markov*, or to have *the Markov property*. Next I define more formally this property.

In a general environment, the response at time t+1 to the action taken at time t may depend on everything that has happened previously. The dynamics of this environment can be described by the following transition probabilities:

$$Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\},$$

On the other hand, if the state signal has the Markov property the environment's response at time t+1depends only on the state and action representation at time t. Thus, transition probabilities can be defined by:

$$Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\},$$

To summarize, a state signal has the Markov property if and only if, **FIG** is equal to **FIG**. In this case, the environment and the task as a whole are also said to be Markovian. This allows us to predict next state and expected next reward given the current state and action. That is, there's independence of the path that has been followed. Even if the state is not fully Markovian it is still appropriate for a Reinforcement Learning task to think of the state at each time step as an approximation to a Markov state, since that provides a good basis for predicting future rewards and for selecting actions and subsequent states. The reward probabilities in a Markovian environment can be expressed using the following expression:

$$\mathcal{R}^a_{ss'} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\}.$$

A Reinforcement Learning task that satisfies the Markov property is called a Markov Decision Process, or MDP. If the state and action spaces are finite, then it is called a finite MDP.

Besides the environment, the agent and the actions, there are other elements of the Reinforcement Learning framework: a *policy*, a *reward function* and a *value function*. Next are briefly described these new elements.

The policy is a mapping from perceived states of the environment to probabilities of selecting each possible action of that state. The policy is the core of the learning agent, it alone is sufficient to determine its behavior.

The reward function defines the goal in a Reinforcement Learning problem. It maps each state (or state-action pair) of the environment to a numeric value, the *reward*, which indicates the desirability of that situation. The learning agent goal is to maximize the total reward it receives in the long run. The reward function allows the designer to define what are the good and bad events for the agent, and thus define the features of the problem faced by the agent. Because of this, the reward function must be unalterable by the agent.

Previously, I have stated that the agent's goal is to maximize the total reward it receives in the long run. That's a bit imprecise, what the agent has to maximize is the *expected return,* where the *return*, is defined as some specific function of the reward sequence. The simplest case is the sum of the rewards:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T,$$

where T is the final step. This approach makes sense in episodic tasks, that is when the Reinforcement Learning task naturally breaks into subsequences, called episodes. Each episode ends in a special state called the *terminal state*, followed by a reset to the start state. On the other hand, if we were facing a continuing task, that is when the task doesn't naturally break into identifiable episodes and goes on continually without limit, this return formulation would be troublesome. In a continuing task the final time step would be T = , and thus the return could easily be infinite too.
To avoid this, the discount rate is used to define the *discounted return:*

$$\sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where is the discount rate. This parameter determines the present value of future rewards and thus is used to define "how farsighted" the learning agent will be.

The value function specifies the *value* of a given state. A *value* indicates the long-term desirability of a state after taking into consideration the states that are likely to follow, and the rewards available in those states. In other words, the *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Value functions are defined with respect to particular policies. The 2 most common value functions are the *state-value function for policy* (V) and the *action-value function for policy* (Q). We can define V and Q formally as

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s \right\},$$

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right\}.$$

Since the learning agent's goal is to maximize the expected return, we may end up with a greedy agent stuck with a non optimal policy because some states have never been visited. To avoid this situation, we must find a balance between exploitation (of acquired knowledge) and exploration (of new states/actions). The agent *explores* when it selects one of the nongreedy actions, allowing it to improve its estimate of the nongreedy action's value. When the agent selects the action defined by its policy, we say it's *exploiting* its current knowledge of the values of the actions.

Exploitation is the right thing to do to maximize the expected reward on the one play, but exploration may produce the greater total reward in the long run. There are several methods for balancing exploitation and exploration. In this particular case, I've decided to make use of the *e-greedy action selection* method. The *e-greedy* method selects the greedy action with probability (1-e) and a random action with probability e. I've decided to use this method because it provides a good balance between performance and difficulty of implementation. In addition to this, the fact that it has only one parameter (the e value) makes it easier to find a good set up.

There are three fundamental classes of methods for solving the Reinforcement Learning problem: Dynamic programming, Monte Carlo and Temporal Difference. I'll briefly introduce the three classes and then explain in more detail the method selected for this system: Watkins-Q( ).

- Dynamic programming

DP is a set of algorithms that can be used to compute optimal policies. These methods require a perfect model of the environment as a Markov Decision Process. The assumption of a perfect model and a high computational expense limits the utility of Dynamic programming methods in Reinforcement Learning. One special feature of DP is that performs bootstrapping, that is, updates estimates of the values of states based on estimates of the values of successor states.

- Monte Carlo

Monte Carlo methods are ways of solving Reinforcement Learning problems based on averaging sample results. Because of this, these methods are defined only for episodic tasks, where experience can be divided into episodes and where all episodes eventually terminate. When an episode is completed, estimates and policies are changed. Monte Carlo methods learn optimal behavior from online or simulated interaction with the environment. They do not require a perfect model of the environment to work.

- Temporal Difference Learning

Temporal Difference Learning combines ideas from the previous methods. Like DP, TD Learning performs bootstrapping: updates estimates using, in part, other learned estimates. Like Monte Carlo methods, TD Learning learns directly from interacting with the environment (raw experience) without needing a perfect model of the environment's dynamics. If TD Learning methods are augmented with eligibility traces we obtain TD( ) methods, a "new family" of methods that are more general and may learn more efficiently. These new methods span from Monte Carlo to one-step TD Learning, thus acting as a bridge between those 2 methods.

Now, it's time to explain the method I've chosen for my Analyst Agent: Watkins Q( ).Watkins Q( ) is an off-policy TD( ) control algorithm, meaning that the learned action-value function Q directly approximates the optimal action-value function, independently of the policy that is being followed. In this particular case, values are learned for the greedy policy, and not for the e-greedy policy used during the learning process. This doesn't mean that the policy is ignored since it determines which state-action pairs are visited and updated. All that is required for assuring convergence is that all pairs continue to be updated. Unlike other TD( ) methods, Watkins Q( ) doesn't look ahead all the way to the end of the episode in its backup since an exploratory action might be taken. In learning about the value of the greedy policy at <St,at> subsequent experience can be used only as long as the greedy policy is being followed. Thus, Watkins Q( ) only looks ahead as far as the next exploratory action is taken. Following, I'll present the Watkins Q( ) complete algorithm in pseudocode:

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $a^* \leftarrow \arg\max_b Q(s',b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s',a^*) - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                  else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

Figure 5.10: Q Learning Algorithm

All these algorithms store their estimates of value functions as a table with one entry per each state or state-action pair. This approach is limited to those tasks where the state space is small. It’s not only because of the memory that is needed for large tables, but because of the data and time needed to fill such a table accurately. In addition to this, in a task with a big state space, most states that the agent encounters will never have been experienced exactly before. The only way to learn something in this situation is to generalize from previously experienced states to the new ones that have never been visited. The use of generalization thus solves two key issues: allows learning in a reasonable time and space, and allows generalizing from old states to new ones.

In this system, I’ve decided to implement the Tile Coding method. Tile Coding is a Linear method, a Gradient-descent function approximation in which the approximate function, Vt, is a linear function of the parameter vector. In Tile Coding the receptive fields of the features are grouped into exhaustive partitions of the input space called tilings. Each element of the partition is called a tile, which is the receptive field for one binary feature. The use of binary features makes the weighted sum making up the approximate value function easy to compute. I’ve used grid-like tilings to ease the computation of the indices of the present features. The number of tilings, and thus the density of tiles, defines the accuracy of the function approximation. The width used for the tiles should match the width of generalization required for the task. In this case we've decided to use a dynamic approach based on the data set characteristics to assure a good balance between accuracy and computational costs.

Chapter 6

System Description

The goals and scope of BROMAS have already been detailed in previous chapters. Here, I we'll introduce the architecture we designed to achieve these goals. In addition to this, we'll also detail the technologies and methodologies used during the system implementation, explaining for each why was it chosen and the function/role they play in the whole system. Implementation details will be explained in chapter 7.

6.1 System Architecture

The BROMAS system has three main tasks: data retrieval, data analysis and data visualization. In order to fully realize these tasks, the system has been divided into three independent sections:

- the multi agent system
- the data storage
- the web/client application

The following image provides a good overview of how these three components are tied up:

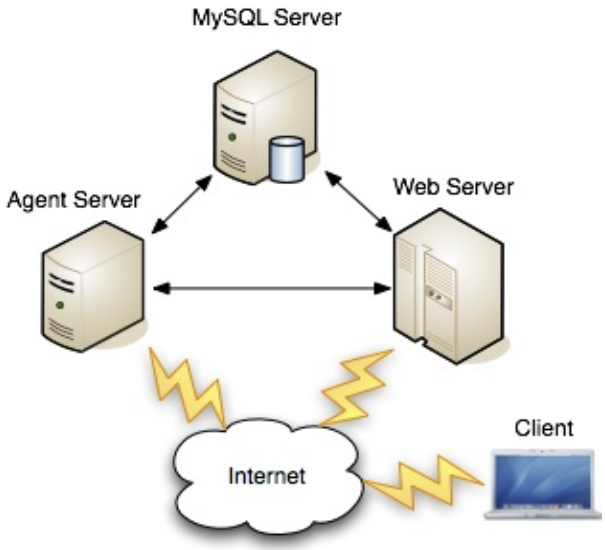


Figure 6.1: BROMAS Hardware Overview

The Multi Agent System is the most important component of the system. Although in the previous image is depicted as a single machine, the truth is that it can be (and *will* be) deployed in a set of machines sharing the same agent platform. This distribution across several machines will be of great usefulness since, as we will see, the multi agent system can be very resource hungry. The Multi Agent System will be responsible for:

- retrieve the data
- process the data
- analyze using Machine Learning methods
- make a decision

As can be seen, almost all the system's logic will be located in this module.

The data storage is currently a Data Base Management System that will be used to store not only the stock data, but the results and statistics of the system. The Data Storage will allow a more efficient data management and will ease the information sharing between agents and between system modules.

The last module will be responsible for presenting the data and the results to the user. To do so two different approaches have been considered: create a web application and create a native client for a given platform. Both are planned to be implemented, but we decided first to focus on the web application since this will allow the system to reach a broader audience.

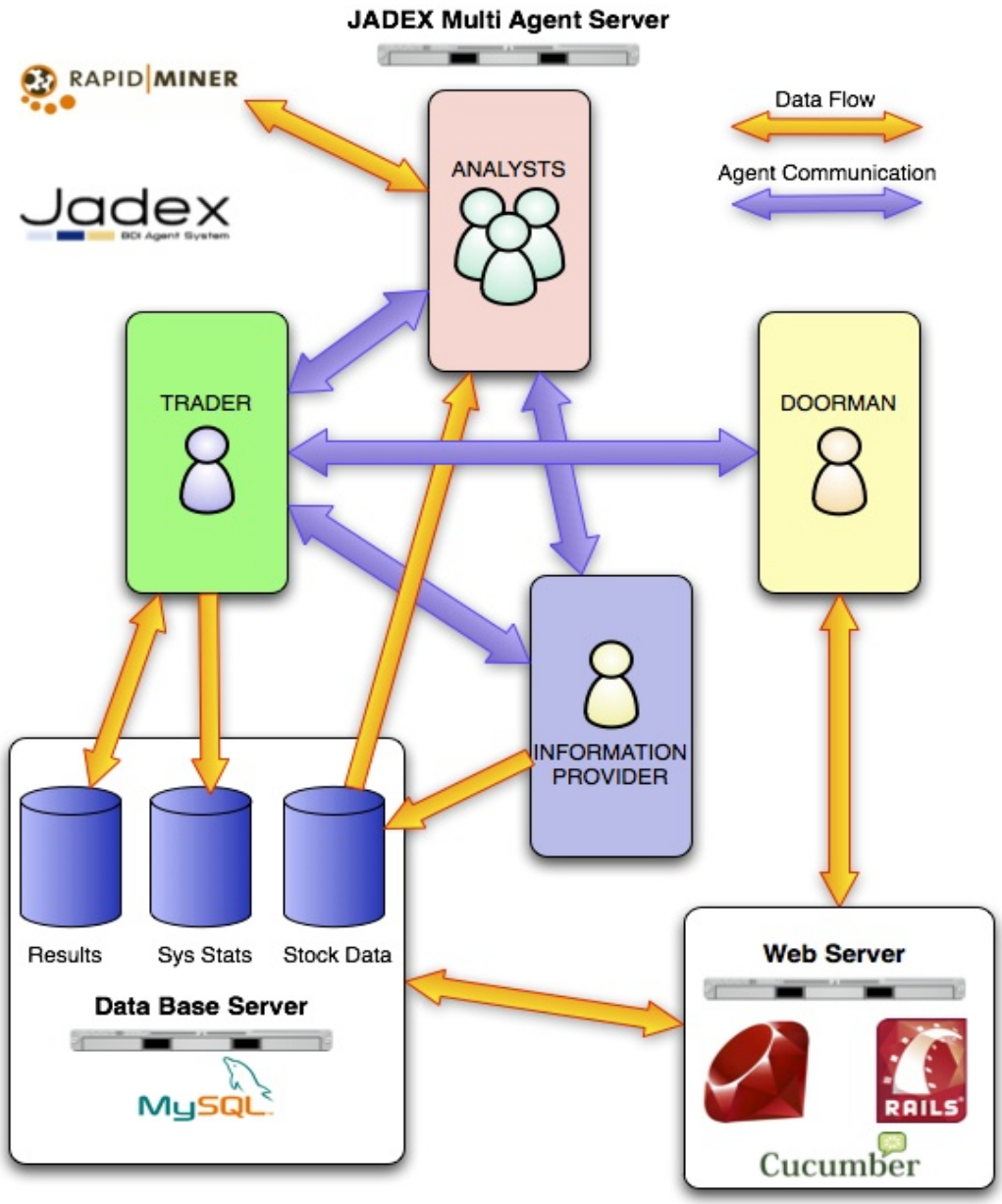


Figure 6.2: BROMAS Architecture

As it can be seen in the previous figure, the multi agent system consists of four different types of agent, each with their own task. The idea behind this organization is to imitate a real world team of financial analysts, which is formed by a group of analysts and a supervisor who is also the team responsible. The decision of using different methods for each analyst has also its roots in the necessity of modeling different points of view. Teamwork is useful when there's divergence between the team members' opinions, a team composed by clonic agents is meaningless. In addition to this, this approach allows us to check the performance of each agent individually and of the team a a whole.

In BROMAS, we have decided to add two additional type of agents: the provider and the doorman. Provider agents will be responsible for gathering and processing of the data used by analysts. Currently, the only Provider agent implemented gathers the historic price data of any stock, although it's planned to add non numeric data to the system too. The doorman type of agents will allow to intercommunicate the multi agent systems with client applications like web or mobile applications, in addition to serve as a monitor of the system and agents status.

6.2 Technologies

This system is a clear example of technology integration, since we're mixing a Multi Agent System with a Web Application and a shared Data Base. In this section, we'll briefly introduce the key technologies used in the BROMAS system.

JADE

The Java Agent DEvelopment Framework is an open source software framework designed to simplify the implementation of multi agent systems through a FIPA compliant middle -ware and a set of graphical tools, to ease the deployment and debug of the system. The agent platform can be distributed across several machines and its configuration can be controlled via a remote GUI.

The JADE framework was initially developed by Telecom Italia Labs and later released under the open source LGPL v2 License, allowing its use in commercial applications. The JADE project is under the management of a board composed by 5 members: Telecom Italia, Motorola, Whitestein Technologies AG, Profactor GmbH and France Telecom R&D.

JADEX

Jadex is a software framework for the creation of goal-oriented agents following the belief-desire-intention (BDI) model. JADEX objective is to fill the gap between middleware and reasoning-centered systems. JADEX is, thus, a reasoning engine that sits on top of a middleware agent infrastructure and allows for intelligent agent development using software engineering foundations. JADEX follows an Object oriented approach backed up with technologies such as Java and XML to ease the development of multi agent systems populated by BDI compliant agents.

JADEX is a research project conducted by the Distributed Systems and Information Systems Group at the University of Hamburg. Like JADE, it has been released under the commercial-friendly LGPL license.

Despite there are other agent platforms/frameworks, we decided to use the combo of JADE+JADEX for several reasons. First, and most important, is our previous experience with both frameworks. We have used both for a couple of university projects and were very satisfied with the obtained results. Besides this, there's another important feature, and it's that both are open source projects that are well documented and widely used.

RapidMiner

RapidMiner (formerly YALE-Yet Another Learning Environment) is an environment designed to conduct machine learning and data mining experiments. Experiments are made up of a large number of nestable operators and stored as XML files which can easily be created with a feature rich Graphical User Interface. What makes RapidMiner interesting is its ability to work as an external Java library, making its integration with an existing Java system almost straightforward.

The initial version was developed by the Artificial Intelligence Unit of University of Dortmund and is distributed under the open source GPL license.

The use of RapidMiner for this project may seem overkill, but the truth is that it has been really handful. Integrating RapidMiner allows to use all its “operators”, which range from data preprocess to visualizations. This means that once you learn how the API works, it's very easy to add new analysts using other machine learning methods. Instead of fighting to integrate several different libraries, RapidMiner allows you to access all its methods with standardized JAVA calls. The minor penalty of consuming more memory is worth taking into consideration the obtained benefits. We're glad to have discovered this tool when coursing Data Mining 2.

Ruby on Rails

Ruby on Rails is an open source web application framework for the Ruby programming language. It follows a Model-View-Controller architecture pattern to organize application programming. It is intended to be used with an Agile development methodology to allow rapid development of web applications.

Ruby on Rails allows the developers to be really productive. The framework leverages many of the tedious aspects of creating a web application, meaning that the developers can focus in what it's really important: the functionalities. Besides this, Rails is Agile, meaning that the cycles between change and test are really short, making development faster. The fact that Rails is powered by Ruby also helps, since it allowed us to create the glue code necessary to integrate all the components of the system in that language.

Processing

Processing is an open source programming language and environment for people who want to program images, animations, and interactions. The language builds on the graphical capabilities of Java, simplifying features and creating new ones. Processing was initiated by Casey Reas and Benjamin Fry, former members of the Aesthetics and Computation Group at the MIT Media Lab.

Although we didn't manage to complete on time a full User Interface using Processing, we've decided to include it here because we consider that this project deserves more public spread. There are a lot of truly amazing projects made with it, and we hope that for the next release we'll be able to include a little visualization tool written based on it.

6.3 Methodologies

“One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems”

A software development methodology is a framework that is used to structure, plan and control the process of developing a software product. Software methodologies are a must when developing systems of high complexity, since they provide good guidelines for developers to follow.

To develop a system like this thesis, a convenient use of proven methodologies is required in order to fulfill the objectives that were set at the beginning. Since there are many different methodologies available, it's of key importance choosing the ones that suit the project that is gonna be developed. Following I present the ones we've used during this thesis development.

Prometheus

Prometheus is an iterative methodology that covers the complete software engineering process, providing an “start-to-end” support (from analysis to implementation) along with design artifacts constructed and steps for deriving artifacts. Unlike others, Prometheus aims specifically at the development of intelligent agents, in particular those who implement a BDI model by using goals, beliefs, plans and events. The resulting specification can be implemented in any agent framework/platform that covers the BDI approach, although it offers a better out of the box integration with JACK.

Prometheus has been developed at Melbourne University over 7~8 years, in collaboration with an industry partner: Agent Software. During development, feedback from students and industry partner clients was incorporated. It is focused on detailed guidance and structure to facilitate the development of multi agent systems to non experts in the field. The specification is done using a mixture of graphical notation for overview and structured text notation for details. In addition to this, an external editor called the Prometheus Design Tool was created to ease the creation and manipulation of agent specifications.

Prometheus results quite similar to classic object oriented methodologies, making it really easy to use for newcomers to the agent world. Its focus on developing intelligent agents based on the BDI model makes it a very good choice to be used along with JADEX+JADE, the platform we were planning to use. Another option was using the Gaia methodology, but since we had prior experience with Prometheus we decided to stick with it.

TDD/BDD

Test Driven Development is a technique that uses short development iterations based on previously written test cases. These test cases define using tests the desired improvements or new functionalities. Each iteration produces the code necessary to pass that iteration's tests. Once the tests pass correctly, it's time to refactor the code and repeat the process. TDD is not a mere testing method, but a software design method. Writing previously the tests forces the developers to think how this functionalities will be used by the users, and thus helps the design and implementation phase.

Behavior Driven Development is a software development technique that encourages collaboration between developers and non technical participants in a software project. To do so, BDD uses a subset of the native language to describe the behavioral aspects of the system. This helps bridging the gap between users and developers, and serves as documentation explaining how the system behaves. BDD can be seen as an evolution of TDD.

This two methodologies have been used to develop part of the Multi Agent system and the entire web application. Software development is an iterative process where code is always evolving. Having exhaustive tests gives us the confidence to change the code as much as we need, without having to concern about unexpected behaviors.

Chapter 7

Design and Implementation

In the previous chapter, we described broadly the main components of the system and how they interact. In this chapter, we'll detail how we have designed and implemented the system in order to fulfill the goals stated in Chapter 3. This chapter will be divided into three sections, one per component. In the first one, the BROMAS agent system will be detailed using Prometheus methodology artifacts. After this, the Data Management Process, which goes from data gathering to class balancing, will be fully explained. And finally, the Ruby On Rails based application and the glue code needed to integrate the 3 components together will be specified.

7.1 Multi Agent System

In previous chapters, we've seen the goals of the multi agent system, the agents that compose it and the tasks they have to realize. In this subsection, we will present the design of BROMAS agent system and its agents: Trader, Analyst, Provider and Doorman following the Prometheus methodology, explained in the following diagram:

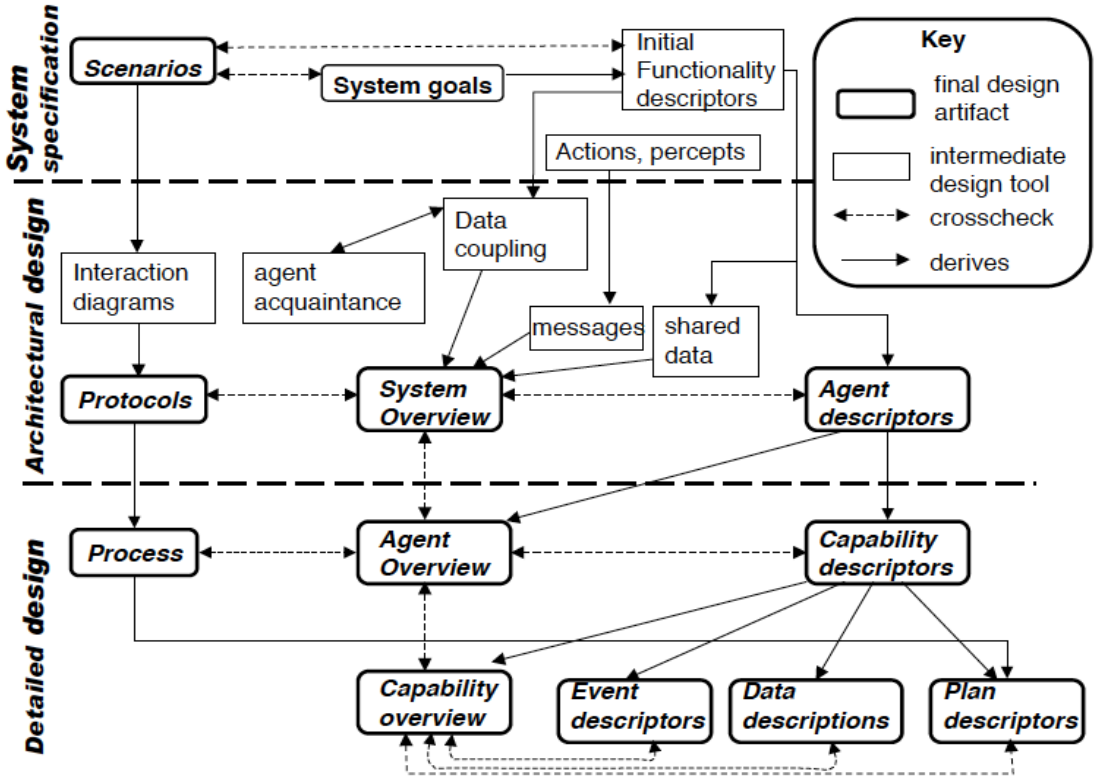


Figure 7.1: Prometheus Methodology

7.1.1 System Specification

This phase provides a general overview of the multi agent system, specifying the global goals and scenarios. Moreover, it identifies the basic roles of the system, along with the actions and perceptions related to these roles.

Goal Overview

Goals are identified from the functionality required by the system, that is, the system's use cases. This system has only one functionality, and thus, only one system goal: Predicting Stock Trends. This main goal has been decomposed in several sub goals, which cover all the subtasks that the agents will have to realize. The two optional goals included are system goals that are still unimplemented: Select Portfolio and Manage Portfolio. With this two new goals the system will be able to recommend stock portfolios to users and manage them based on their performance.

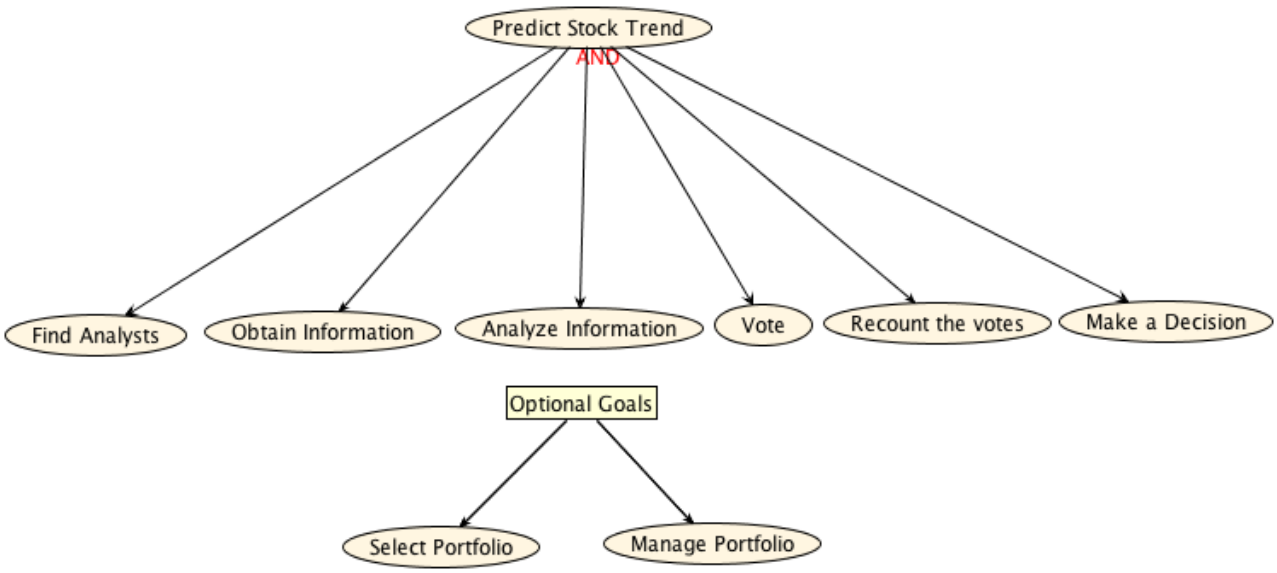


Figure 7.2: BROMAS Goal Overview

Scenarios

Following system goals, we have to define the use case scenarios. Use case scenarios are a detailed description of one particular example sequence of events associated with achieving a particular goal, or with responding to a certain event.

Scenarios are described using a name, description and a triggering event. Scenarios are composed by a sequence of steps, each consisting of the functionality that it performs. There ara several type of steps available: Action, Percept, Goal, Scenario or Other. Each step can be described with a name and the information used and produced by it.

Since the system has one use case implemented and two optional, the scenarios are the following.

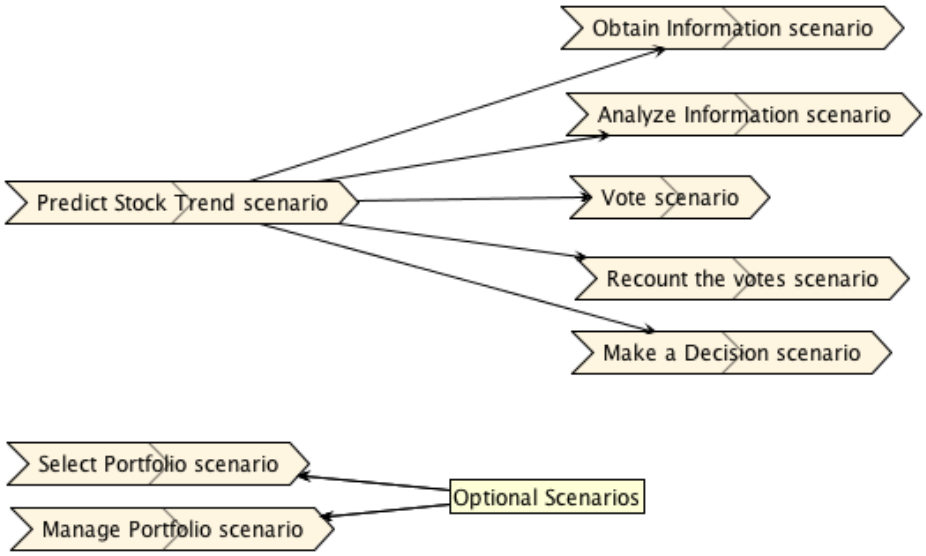


Figure 7.3: BROMAS Scenarios

System Roles

After defining the scenarios and goals, it's time to define the environment within which the agent system will be situated. This means describing the Percepts available to the system, the Actions that it will be able to perform, and define the Roles that will be assigned to the agents. In the BROMAS agent system we have identified seven roles:

- Environment management: a management role. The agent playing this role has to find the available analysts, notify them the stock to analyze, and control their behavior.

- Information acquisition: when the “GetInformation” message is received, the agent playing this role will start the information retrieval process.

- Information analysis: when the AnalyzeInfo message is received, the agent has to analyze the available data to emit a vote with its decision.

- Vote management: checks that the Votes received are valid, stores them in the shared DataBase, and checks if the Voting session has finished.

- Decision making: once the Voting has finished, the agent associated to this role has to analyze the votes and take a decision wether the stock price will rise, drop or stay. This decision will be notified to the user and later be compared with the correct result to realize some updates to the agent's logic if necessary.

- Portfolio generation: once the user asks for a portfolio recommendation, the agent entrusted with this role will have to generate a portfolio adapted to the user.

- Portfolio management: given a defined portfolio, the agent will have to monitor its performance, make changes in its components if necessary, and notify the user of any relevant information

- Client API

This is better illustrated in the System Roles diagram shown next.

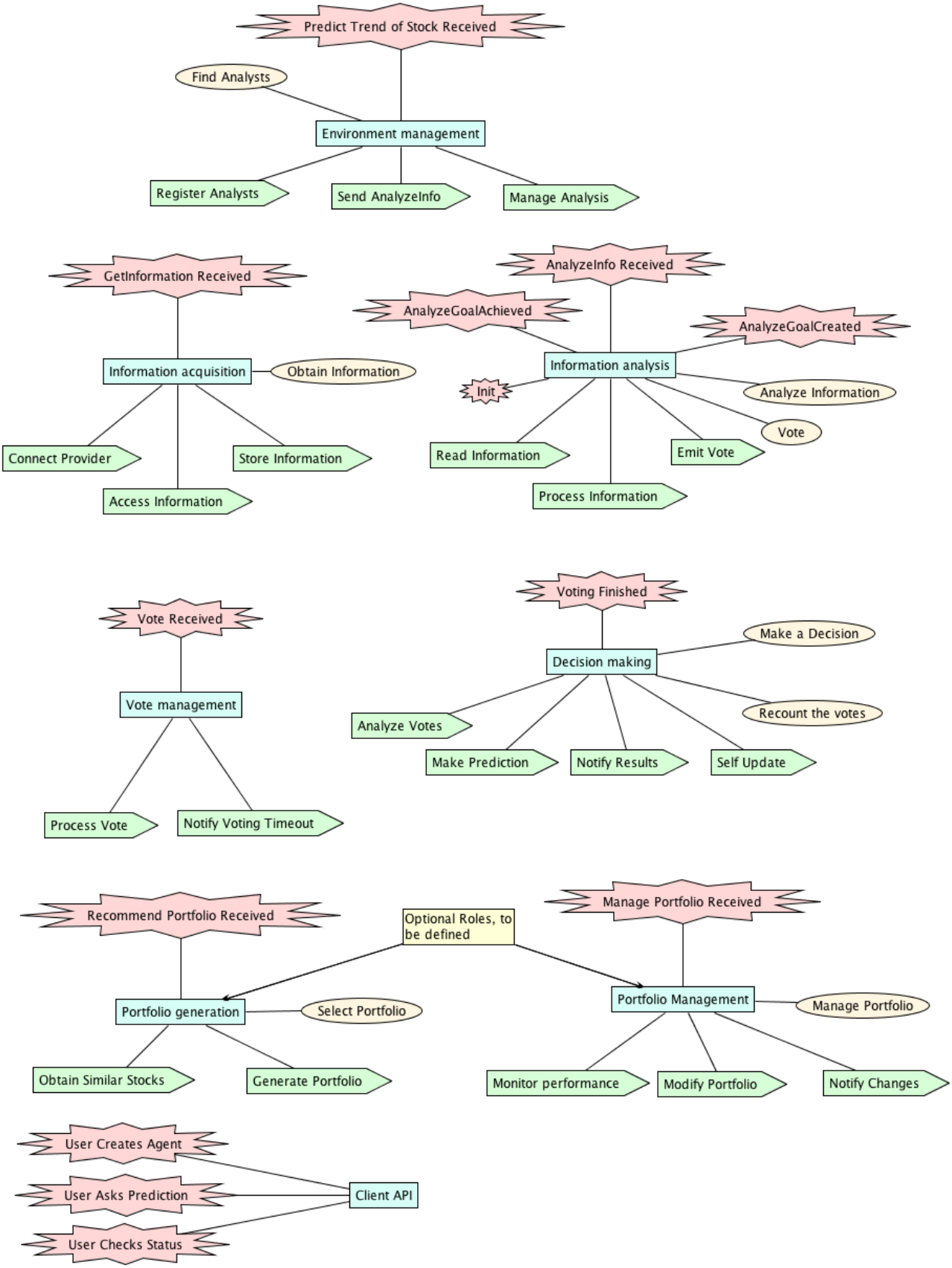


Figure 7.4: BROMAS System Roles

7.1.2 Architectural Design

In the Architectural Design phase the focus is on deciding the different agent types in the system, describing the interactions between agents using Interaction diagrams and Interaction protocols, and designing the overall system structure.

Deciding on the agent types that will exist in the system is the most important decision that is made during this phase. The idea is to obtain a set of agent types, where each type is cohesive. To obtain such a set, we combine functionalities that are related or that make use of the same data. Each agent type should be cohesive, meaning that we shouldn't group clearly unrelated functionalities, despite they use the same data.

These functionality groups will become the agent types of the system. To help with the creation of these combinations, Prometheus offers two tools: the Data Coupling Diagram, the Agent-Role Grouping Diagram, and the Agent Acquaintance Diagram which are introduced next.

Data Coupling

The data coupling diagram depicts the functionalities and data resources of the system showing how functionalities read and write these data resources. BROMAS system uses a noticeable amount of information, divided into several data resources:

- Technical data: the provider gathers the historic price data of the company from the internet, calculates the necessary technical indicators, and stores the resultant data in this data source.

- Voting data: when the trader receives a vote from an analyst, stores its predictions in this data source to use it during the Decision Making scenario, and to make it available to the Doorman agent or client applications.

- Agents data: in this data source the trader agent stores some statistics of each analyst, too keep track of which is the most trustworthy of them.

- Textual data: the provider gathers the necessary textual data to do a fundamental analysis of the stock from the internet, processes it, and stores the processed data in this data source.

- Portfolios: contains the different stocks that compose the portfolio and its performance.

The Data Coupling Diagram allows the developers to define which data is required by the system and who and how will access it. Following, the diagram for the BROMAS system:

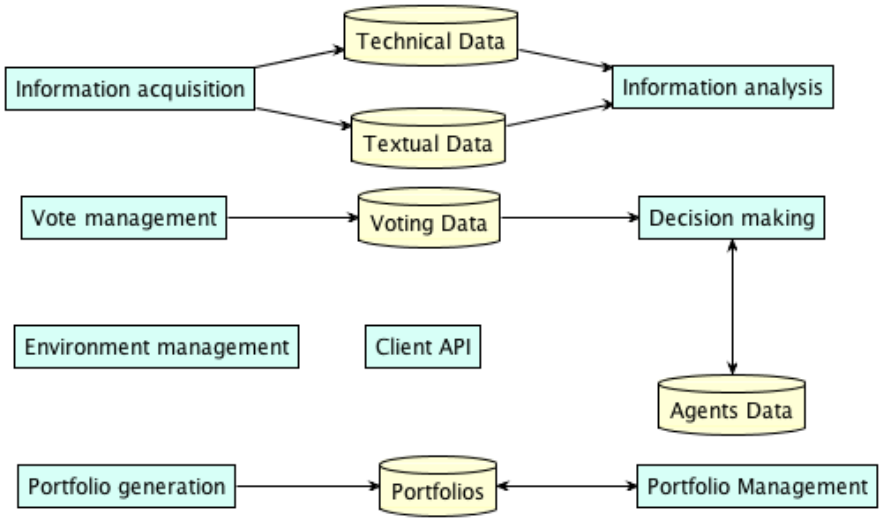


Figure 7.5: BROMAS Data Coupling

showing how the roles previously defined access and manipulate the data.

Agent-Role Grouping

The Agent-Role Grouping Diagram describes which agent type is the responsible for each of the roles that the system provides. Since roles define the different functionalities of the system, this diagram portrays the functionalities each agent is able to do.

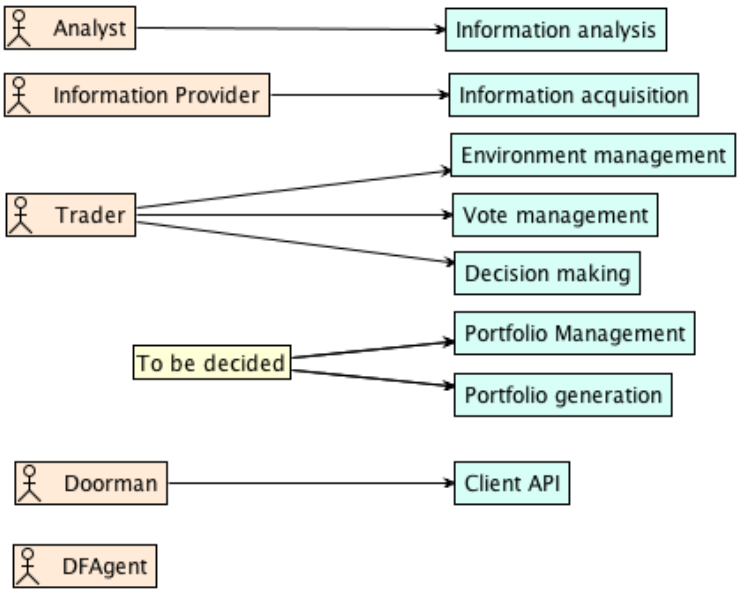


Figure 7.6: BROMAS Agent-Role Grouping

Agent Acquaintance

The Data Acquaintance Diagram shows how coupled the different agent types are

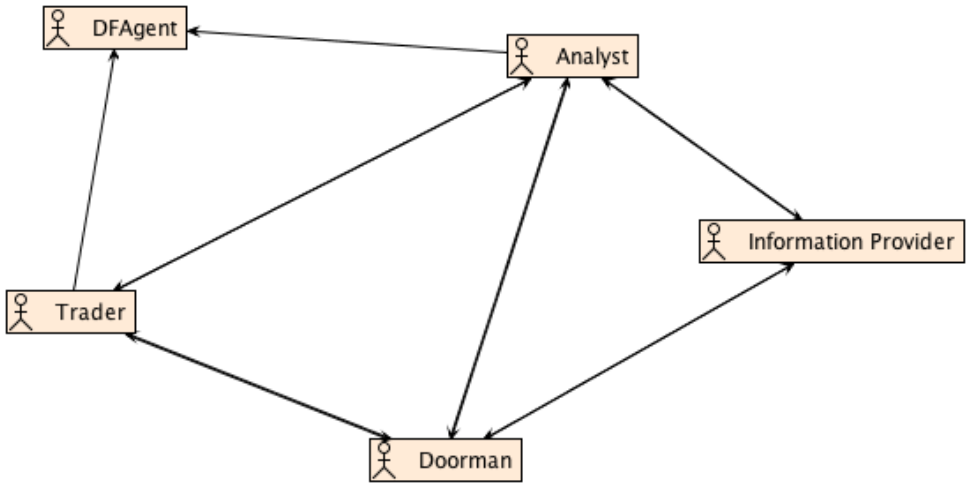


Figure 7.7: BROMAS Agent Acquaintance

System Overview

Finally, the System Overview Diagram summarizes all the Architectural Design Phase in one diagram. This diagram ties together the agents, events and shared data objects. By viewing a system overview diagram, we obtain a general understanding of how the system as a whole will function, including interactions between agents, depicted as interaction protocols. Interaction protocols and Agent descriptors provide additional detail needed to understand the high level functioning of the system.

To fully specify the interaction between the agents, each Interaction protocol has an Interaction Diagram linked that can be accessed from the System Overview Diagram. These Interaction diagrams are created using aUML, a “version” of UML specially designed for Agent systems.

Because of the amount of information that it contains, the System Overview Diagram is arguably the single most important artifact of the entire design process.

The Interaction protocols between the agents are:

- Register protocol: the analysts registers themselves with the Directory Facilitator Agent; once the trader requires an analysis, asks the Directory Facilitator for the analyst agents available. Next, the trader contacts them, informing that have been registered by the trader as a “team”.

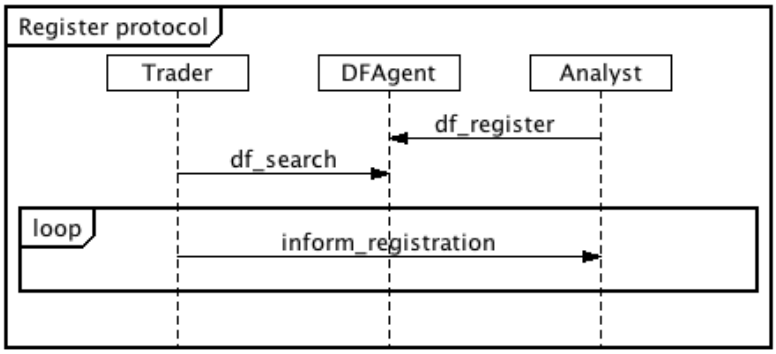


Figure 7.8: Register Protocol

- Analyze protocol: the trader contacts all its analysts to request an analysis of a given stock. When the analysts are informed, ask the providers to update the required data if its out of date.

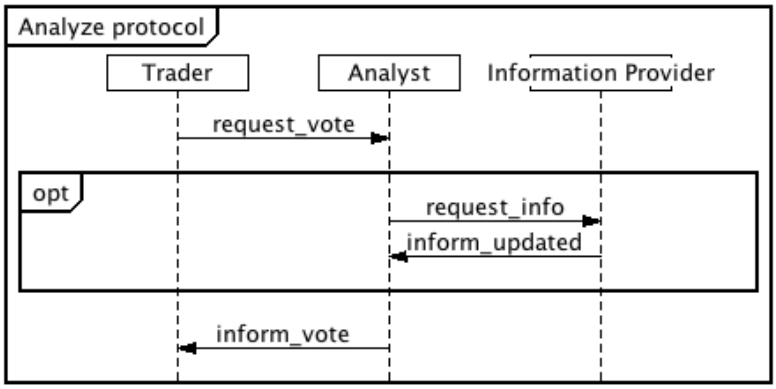


Figure 7.9: Analyze Protocol

- Gateway protocol: when the client application generates an analysis request, the doorman agent contacts the trader and request its prediction. In addition to this, the doorman agent can also request to any agent its status, and can contact the trader to create new agents if needed.

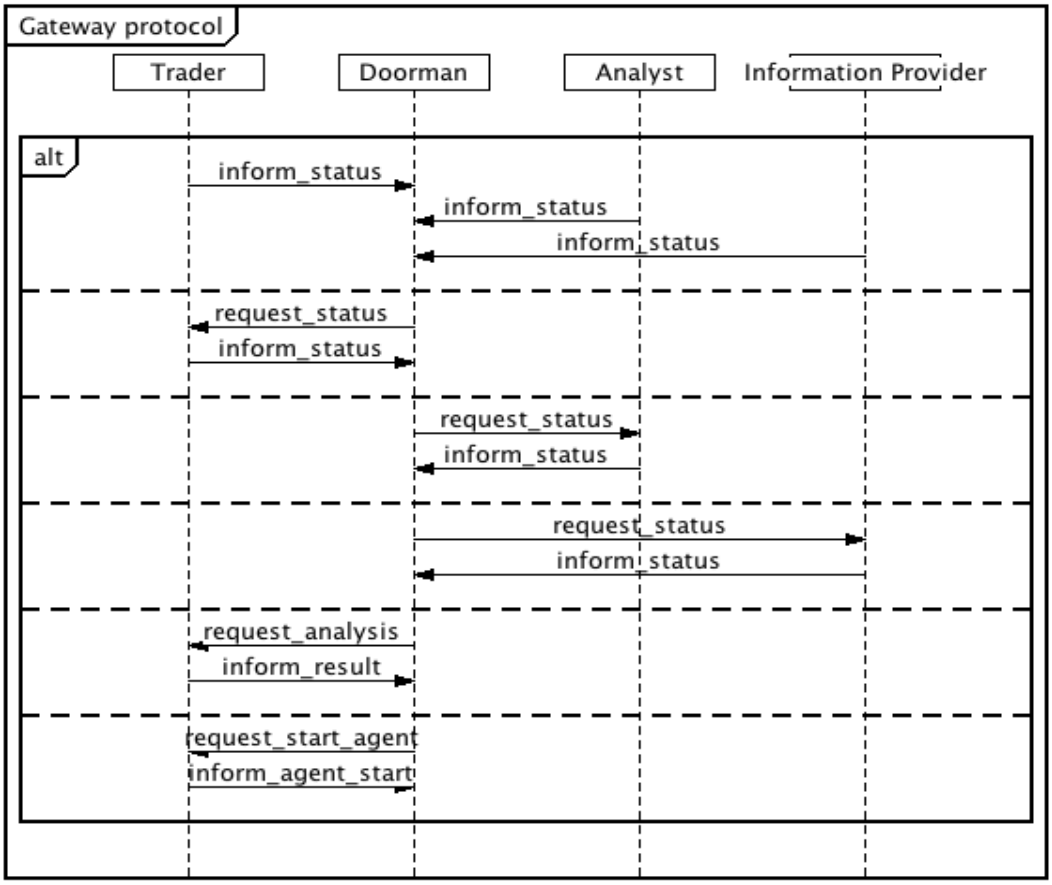


Figure 7.10: Gateway Protocol

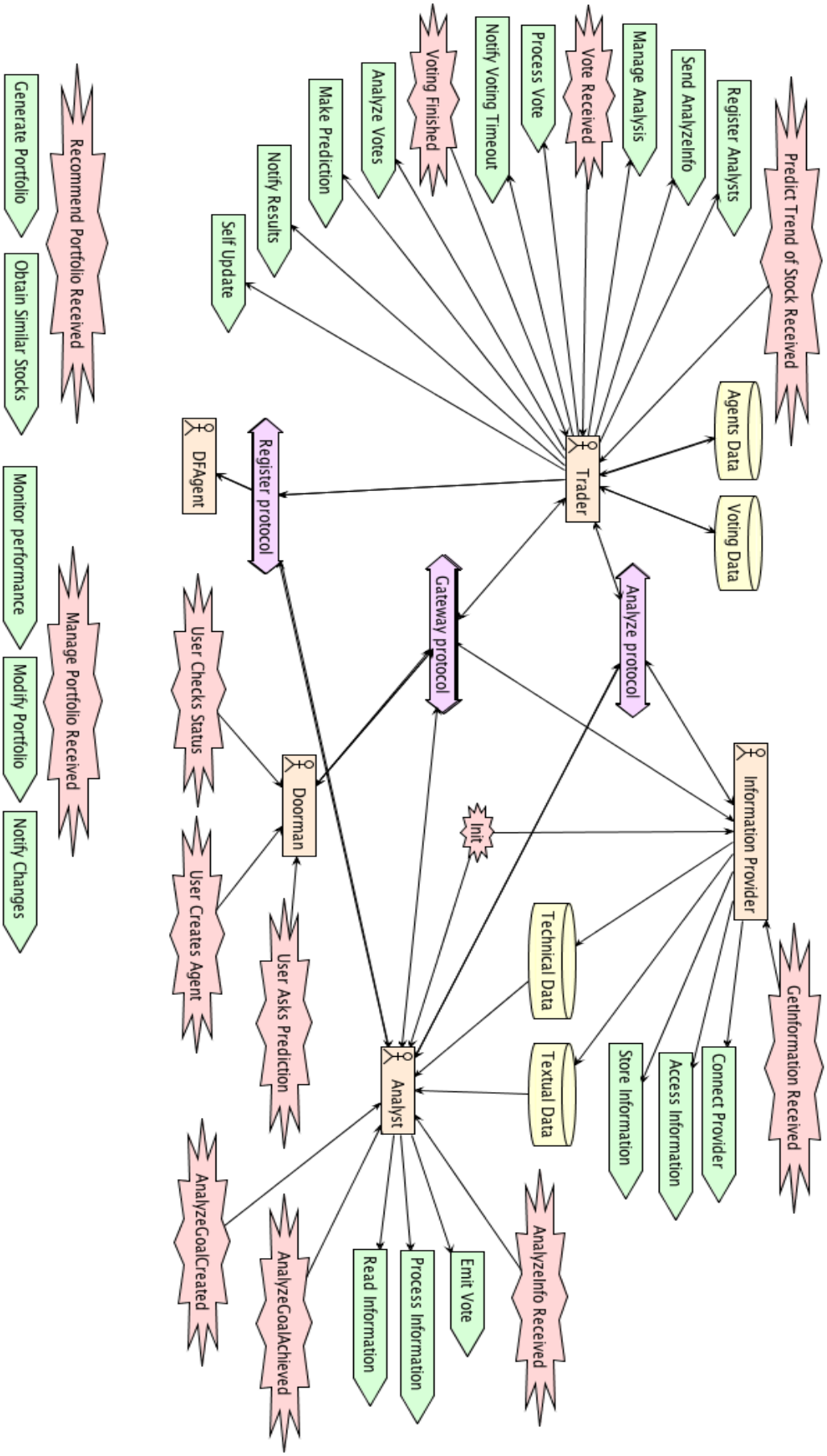


Figure 7.11: BROMAS' System Overview Diagram

7.1.3 Detailed Design

This phase focuses on developing the internal structure of each of the agents and how it will achieve its goals within the system. This is done in terms of capabilities, events, plans, and data. For each agent present in the system, Prometheus generates an Agent Overview Diagram, which will be filled with all this information to provide an overview of the internals of the agent.

Trader

As we said in the previous chapter, the Trader is the supervisor, the manager that controls that everything is working as planned. It is the central piece of the BROMAS agent system, coordinating the actions of all the agents present in the system. Not only does it coordinate the others, but is also responsible for taking the final decision in the analysis process. Using the voting data and the agent data of each analyst, the Trader predicts which trend the stock will follow. In addition to this, the Trader agent is also responsible for realizing the actions that the client application through the Doorman agent requests.

The Trader agent is composed (in addition to the actions, messages, and events that were shown in previous diagrams) by the following plans:

- Manage Analysts: once the Trader receives the Request Analysis event, it has to find and register all the analysts that are available in the system. Once all the analysts are registered, the Trader requests them to perform the analysis.

- Vote Received: every time the Trader receives a vote, it checks that the vote is valid (I.e checks that is coming from a registered analyst), processes it, and stores it in the shared data repository called Voting Data.

- Make Predictions: once the voting period is over, the trader makes a decision using the Agents Data and the Voting Data, and notifies it to the user. This is the most important plan of the trader. By using the Agents Data, the Trader decides how trustworthy each analyst is, and weights the analyst decision using this trust measure.

- Generate Stats: this plan is activated once the Trader has notified its decision, it generates a set of statistics of the current analysis and stores them in the shared data source for further use/analysis.

- Update Self: once the real result is know, the trader compares its decision with the expected value, and updates itself to improve its performance.

- Manage Agents: under request of the Doorman agent, the Trader has to create a certain kind of agent.

- Notify Status: under request of the Doorman, the Trader has to inform its current state.

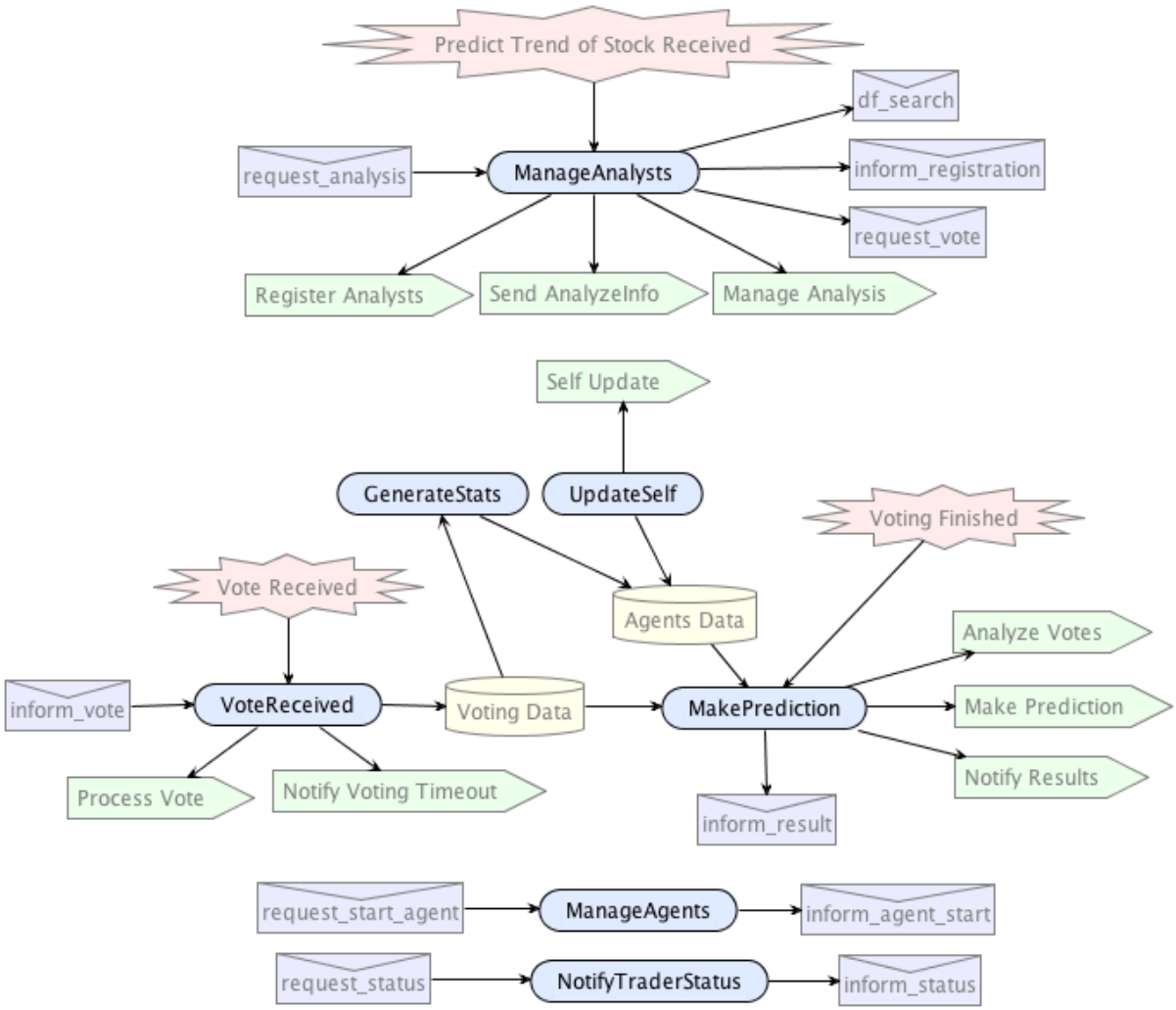


Figure 7.12: Trader Agent Design

Analyst

The Analyst is the responsible of using machine learning methods to predict the future trend of a stock. This information will be delivered to the Trader, who will then decide which prediction is the most feasible.

The Analyst agent is composed of the following plans and capabilities:

- DFRegister: this capability allows the analyst to register itself with the DF Agent, available in the agent platform.
- Registered: this plan is activated once the “inform registration” message is received, allowing the agent to prepare itself for the analysis phase.
- Vote Requested: once the request vote message coming from the Trader is received, the agent updates its *Belief base* with the stock to analyze and creates the analyze goal.

- Analyze: this plan contains all the logic necessary to analyze the stock data and make a prediction. If the necessary data is not present, or out of date, requests it to the providers. Once the analysis is over, the Send Vote plan is activated.

- Send Vote: this plan sends the agent's predictions to the Trader.

- Notify Status: answers a request status message incoming from the Doorman.

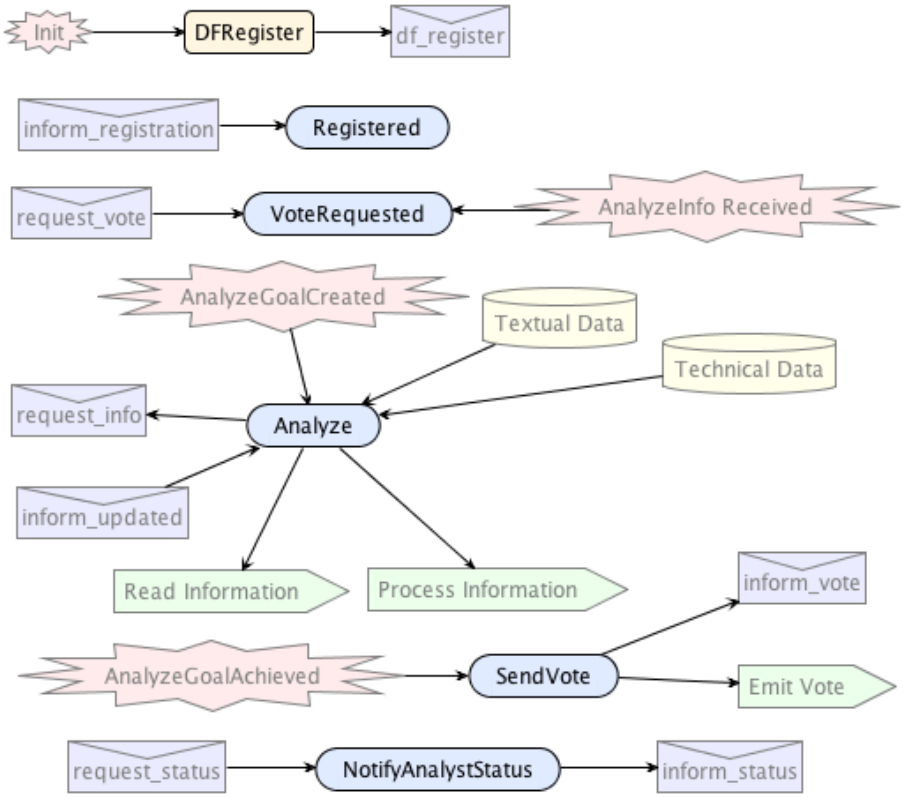


Figure 7.13: Analyst Agent Design

Provider

The Provider agent is the responsible for scraping the necessary data from the internet, transform it to an agent usable format, and store it in the shared data sources. It is composed of the following plans:

- Retrieve Information: this plan is executed when the agent is created, and every certain period of time, to retrieve the necessary information of the stocks.

- Information Requested: when an analyst needs data, request it to the providers using a request info message. Once received, the provider checks the data source, and updates it if necessary.

- Notify Status: this plan is activated when the Doorman's request status message is received.

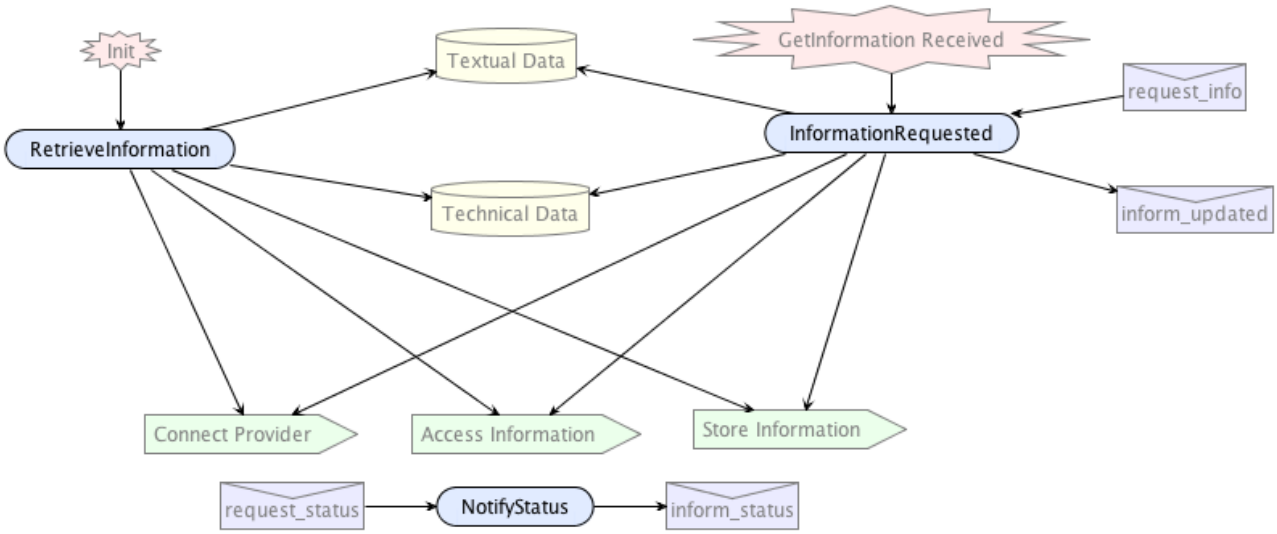


Figure 7.14: Provider Agent Design

Doorman

The Doorman agent, as its name suggests, manages the communication with external applications, providing an “API” to access the system's functionalities. Currently, the Doorman is designed with the following plans:

0ma

- Retrieve Status: if the client application wants to know the current status of an agent, this plan is executed to send a request status message to the agent.

- Get Prediction: if the client application wants to get a stock trend prediction, the Doorman agent executes the Get Prediction plan to start the whole analysis process.

- Start Agent: if the user wants to create a certain type of agent, this plan is executed. Doorman agent will send a request start agent message to the Trader, who will be responsible for initiating a new agent.

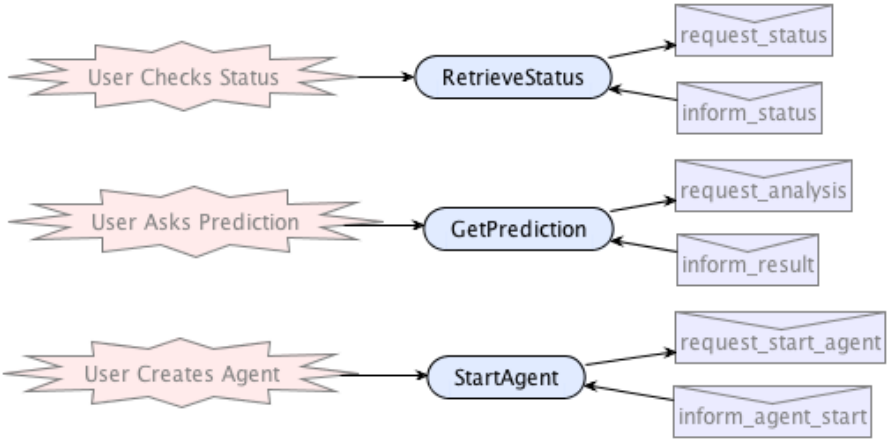


Figure 7.15: Doorman Agent Design

7.2 Data Management

In this section, we will present how the Data is managed in the BROMAS system, from its obtention to its storage.

During the past chapters, we have mentioned several times that the Provider agent transforms the data it obtains from the internet into an agent understandable format. To explain what do we mean by that, we have to introduce a key concept in multi agent systems: the Ontology.

7.2.1 Ontology

The term ontology comes from the Greek “Study of Being” and has its roots in philosophy, where it's the philosophical study of the nature of being, existence or reality in general. It's also the study of the basic categories of being and their relations. The term has been applied in many different ways, including Computer Science and Artificial Intelligence.

In computer science an ontology is the representation of entities, ideas, and events, along with their properties and relations, according to a system of categories. A formal representation of the concepts within a domain and the relationships between these concepts. There are many different possible representations, but ontology engineering is concerned with making representational choices that capture the relevant distinctions of a domain at the highest level of abstraction while still being as clear as possible about the meanings of terms.

Early AI researchers recognized the applicability of the work from mathematical logic and argued that new ontologies could be created as computational models to enable certain kinds of automated reasoning. Later the term was used to refer to both a theory of a modeled world and a component of knowledge systems.

From the point of view of Agent theory, ontologies can be seen as the *vocabulary* that agents need to use in order to talk about a given domain. Agents can send and receive messages without using an ontology, but then there's no guarantee that the receiver agent will understand the message as the emitter does. This can happen since agents may have different, over-lapping and/or mist-matched concepts, structures and methods, leading to this lack of shared understanding. To allow sharing an interpretation of information structure between people/agents is one of the main motivations of ontology development.

Another important motivation for creating ontologies is knowledge reuse. This way, to create our ontology we can reuse parts or entire ontologies instead of writing them from scratch. In the BROMAS case, parts from other projects' ontologies have been used.

There is no single standard methodology to develop ontologies, nor there is a single correct method to model a domain. Each domain is unique in some aspects, and thus the best solution depends on it. Despite of this, there are some phases that are present in most methodologies. Following, each of these phases is detailed.

Determine the domain and coverage for the ontology

The first step when developing an ontology is to define its domain and scope. To do so, we must be able to answer the following basic questions

- What is the domain that the ontology will cover?

In BROMAS, the ontology will cover our particular approach to stock market trading. This includes the way stock data is represented as well as the information that the agents need to share during the analysis process like votes, predictions, …

- For what are we going to use the ontology?

This ontology will be used to provide a suitable format/structure to the data required for doing an analysis, and to provide a shared vocabulary that all the agents accepted in BROMAS will have to understand and use. This way, agents will use concepts like stockdata, windowdata, vote, etc. know what they are, and how to use them.

- For what types of questions the information in the ontology should provide answers?

As stated in the previous question, in BROMAS the ontology will be used as a way to structure the information shared within the system, instead of creating a knowledge base. Because of this, the questions would be more of the type: What is a stock data object? Which components form a Vote? Etcetera.

- Who will use and maintain the ontology?

Well, this ontology will be designed specifically for BROMAS, so it will be used and maintained by me. Despite of this, anybody will be able to use it once I make it public.

Consider to reuse existing ontologies

BROMAS ontology is very simplistic, there aren't many classes, nor deep hierarchies. Because of this, the cost of developing a custom ontology for BROMAS is lower than the cost of finding a suitable ontology and modify it to be usable in the system.

Despite this, we checked old ontologies developed for other projects to get some inspiration and find some ideas that could be reused.

Enumerate the important terms in the ontology

It is useful to write down a list of all terms we would like either to make statements about or to explain to a user.

For BROMAS we considered the following terms:

- Data: StockData, WindowData, Vote
- Messages: Request Info, Request Vote, Inform Vote, Inform Result

Basically all messages with complex content and their respective content data.

Define the classes and their hierarchy

This step can be realized using three different approaches: top-down, bottom-up or a combination of the previous two. None of these methods is better than any of the others. The approach to take depends strongly on the personal view of the domain. For the BROMAS system, we opted for the bottom-up approach, basically because of the few classes this ontology defined. We thought it would be easier to start by defining each of the terms introduced in the previous step, and then group them into more general concepts. This generalization process ended when we reached the root class: the Concept.

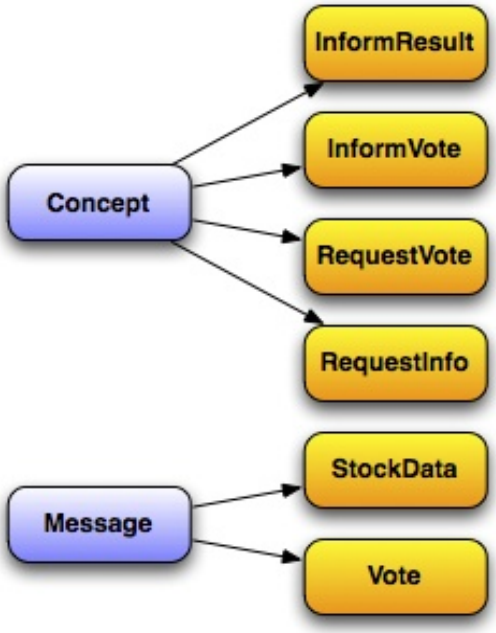


Figure 7.16: Ontology Class Hierarchy

Define the attributes of each class

Classes alone are not enough to answer some of the questions defined at the first step. We have to provide more information. And to do that, we have to describe the internal structure of concepts, its attributes.

The previous step and this one are closely intertwined. It's quite difficult not to define the attributes of each class while we are defining it, specially if we later have to generalize them to create a superclass. Because of this, what we did was first to define the classes, then describe each with attributes, and then generalize.

Figure 7.17: Stock Data Concept

Protégé

To develop the ontology we have just described, we have used the protégé platform. Protégé is a free, open-source platform that provides its users with a suite of tools in order to construct domain models and knowledge-based applications with ontologies. At its core, protégé implements a rich set of knowledge modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representational formats. protégé can be customized for the purpose of providing domain-friendly support in order to create knowledge models and to enter data. Further, protégé can be extended via a plug-in device and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

The Protégé platform supports two main ways of modeling ontologies:

- The Protégé-Frames editor enables users to build and populate ontologies that are *frame-based*, in accordance with the Open Knowledge Base Connectivity Protocol, or OKBC. In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes.

- The Protégé-OWL editor enables users to build ontologies for the *Semantic Web*, in particular in the W3C's Web Ontology Language, or OWL. An OWL ontology may include descriptions of classes, properties and their instances. What makes OWL ontologies different is the OWL formal semantics, which specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms.

For BROMAS, we have chosen the first option, a Frame-based ontology, since we don't need the formal semantics that the OWL offers. Currently, a frame-based ontology satisfies all our needs. We have decided to use protégé because of its integration with the agent platform we use. Both JADE and JADEX provide plugins for developing ontologies with protégé. These plugins, called Beanynizers, generate from a protégé ontology document Java classes or Java Beans, which can be directly imported into our multi agent system. This eases tremendously the amount of effort needed to create an ontology for a multi agent system.

7.2.2 Analysis Data

In the Prometheus specification we showed 4 different data sources: the Technical Data, the Textual Data, the Voting Data, and the Agent Data.

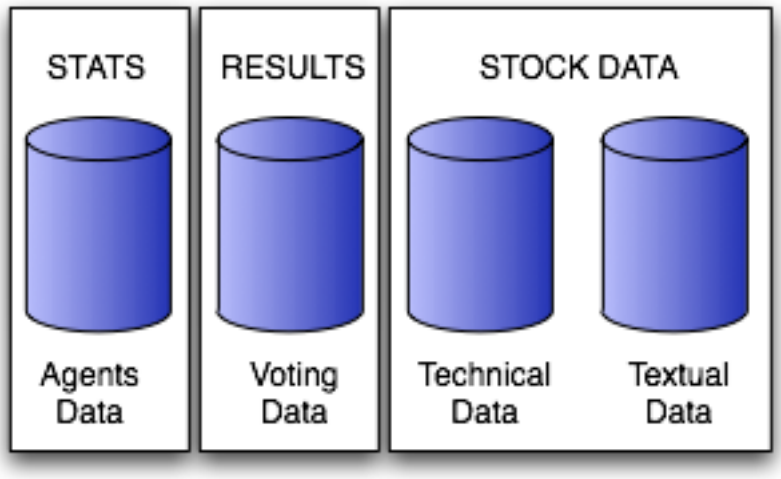


Figure 7.18: BROMAS Data

In this section, we will describe the ones that are currently implemented, that is, the Technical Data, the Voting Data and the Agent Data

Technical Data

This is the most important data source of the system. For each stock, we must do the following tasks:

- Retrieve historical data
- Calculate technical indicators
- Create 10 day window
- Balance data samples

We will be using Yahoo's Finance page to retrieve stock's historical data. Yahoo Finance offers the possibility to export this data to a CSV file. Using a web scrapper, the Provider agent scraps the page to find the “Export to CSV” link. This way, the system won't depend on a particular URL that may change in the future, making it page independent. Once the CSV file has been retrieved, we use a CSV reader to transform each line into a StockData object.

Once we have an array of StockData, filled with all the historic data of a given stock, we start calculating the technical indicators. We chose the following indicators based on the advice given by a professional trader:

- MACD: the Moving Average Convergence Divergence indicator shows the difference between a fast and slow exponential moving average of closing prices. It's a trend following indicator designed to detect trend changes.
- RSI: the Relative Strength Index is a momentum oscillator that measures the velocity and magnitude of directional price movement by comparing upward and downward close-to-close movements. RSI shows when a stock is overbought or oversold.
- STS: the Stochastic Oscillator is also a momentum indicator used to compare the closing price of a stock to its price range over a given time span.

With these three technical indicators we, in words of our consulted Expert, have enough information. In addition to this, we also calculate the percent difference from day-to-day, which will be used later to label examples.

Now that we have for each day all the required information, we create 10-day-window data, instances of WindowData, that is the information that will be analyzed. This means that the system will use the data of 10 consecutive days to try to predict what will happen the 11th. This is done to provide the agent with information of the current trend. Without this information, it's almost impossible to predict the evolution of the stock price trend. We decided to use a 10 day window since it's among the common values used for this purposes: from 5 to 15 days. A window too small wouldn't provide enough historical information, and a window too big would provide too much, making the agent to lose its focus in the short term trend prediction.

There's one exception to this, for the Reinforcement Learning powered analyst we will be using individual data instead of 10-day-windows. We thought that this daily approach suited better a reinforcement learning environment. A Reinforcement Learning state is represented by the adjusted price, the volume, percent variation, technical indicators and the label of that state.

The reward function implemented is quite simple: it gives a positive reward to the agent each time it chooses the correct action, and gives a negative reward (which doubles the value of the positive reward). This way the Reinforcement Learning agent will learn a risk avoidance policy that minimizes the amount of mistakes done.

All this StockData and WindowData objects are stored into an MySQL database. This way the data will be available not only to the multi agent system, but also to RapidMiner and other client applications.

Once we have created all the WindowData objects, we have to check the class distribution of the examples. There are three possible classes:

- Up: the price rises a 5% or more.
- Down: the price drops a 5% or more.
- Stay: the price remains stable between 5%.

When we started to test our system, we found out that the machine learning methods that we were using were always predicting a Stay class. Only the Decision Tree labelled some data as Up or Down examples. This was because the number of examples of the Stay class was overwhelmingly superior to that of the other two. Because of this, even by labeling all the data as Stay, methods were able to achieve an Accuracy over 75%.

The purpose of BROMAS system is to predict stock trends, and more specifically, to detect the turning points, the days in which the stock trend is reversed. An accuracy of 90% it's of no use if the system can't find any turning point, that is, the examples labeled as Up or Down. In order to improve the Precision and Recall of the minority classes, we had to slightly tweak the data.

Minority class examples (Up & Down) combined were less than the 10% of the total examples available. Since (most) learning algorithms' goal is to maximize the accuracy, when presented with imbalanced class distributions they label all examples with the majority class. Based on the underlying assumptions, this is the intelligent thing to do.

In order to avoid this, we decided to implement some data balancing techniques to rebalance the data sets artificially. There are two main methods to do this: down-sampling and up-sampling. Basically, in down-sampling you ignore cases from the majority class, while in up-sampling you replicate examples from the minority.

Since repeating information didn't look too reliable, we decided to try first down sampling the Stay class examples. Our approach was to obtain a data set where the majority class would be represented by the 50% of the examples. By down-sampling the Stay class we

managed to improve the retain and precision of the minority classes while maintaining a good accuracy score. Despite of this improvement, the recall/precision values for the minority classes were still too low, around a 35%.

We have mentioned Accuracy, Recall, and Precision in the previous paragraphs, but we haven't explained what these measures are. Following there's a brief description to each of them.

Accuracy is the proportion of true results (both true positives and true negatives) in the total population. An accuracy of 100% means that the algorithm identifies all examples correctly.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

Recall is defined as the number of true positives divided by the total number of elements that actually belong to the positive class, that is, the sum of true positives and false negatives. Recall tells the amount of elements of one class that we were able to label correctly.

$$\text{Recall} = \frac{tp}{tp + fn}$$

Precision is defined as the number of true positives divided by the total number of elements that were labeled as belonging to the positive class, or the sum of true positives and false positives. Precision tells the amount of labeled elements that truly belong to the predicted class.

$$\text{Precision} = \frac{tp}{tp + fp}$$

For example, if our system always labels data as Stay, the recall value for Stay will be 1, because all Stay examples were labeled correctly. Unfortunately, since all the Up/Down examples were all labeled incorrectly as Stay, the precision value for Stay won't be of 1. As we can see by this, there's an inherent trade-off between recall and precision, meaning that it's hard to improve one without hurting the other.

To overcome this situation, we checked again the class distribution of our data sets. It was true that now the data sets were more well balanced, but there was another issue that we hadn't considered yet: the amount of examples of each class. Minority examples were really scarce in some data sets, and since we were down sampling the Stay class using the amount of minority class examples as a reference, we were generating data sets that were too compact.

In this situation, we decided to also implement up sampling for the Up and Down classes. We were replicating cases, but we thought that this was a better decision than lowering the threshold used to label examples. The new balanced training data sets were stored in a specific table of the DataBase.

After this, we were able to improve a bit more the learning algorithms' performance, and because of this, we switched our focus to tune their parameters correctly to further improve the system's score.

Voting Data

The initial idea regarding the Voting Data was that the Analyst would send an Inform Vote message to the Trader for each example analyzed. There were two reasons for ditching this idea: first, the Trader agent would be totally overwhelmed by the volume of incoming messages; and second, RapidMiner didn't allow us to obtain the results of the analysis on the fly. We had to wait until the whole data set was analyzed and then read a text file where the results for each example were written.

Due to this, we decided that the Inform Vote would include an array of Vote objects. The Trader would then check the validity of the received data, and store it in a specific table of the data base if everything was correct. Each Voting object is composed by the ticker name, the date, the predicted label, and the agent weight in the system, which at this point is the default value: 1. The idea behind including the weight of the agent is to be able to later analyze the weight evolution of each agent during an analysis.

Once the Trader agent had received all the votes or the Voting has timed out, its time for it to fully use the Voting Data. In the Make Prediction plan, the Trader checks the Voting Data of each analyst, and for each example uses a simple majority mechanism with the votes weighted to decide which label assign to the given example. Weight values range from 0.1 to 1, denoting how reliable an analyst is. Weights show how much the Trader trusts every analyst. Weights are updated at every step of the decision making process. When the agent misses two turning points, its weight is reduced in one tenth. On the contrary, when an agent correctly detects a turning point its weight is increased. Once the Trader has evaluated the decision made by each of the analysts, stores its decision in the Data Base, notifies the Doorman agent, and updates the weights of the agents if necessary.

Agents Data

The Agents Data contains all the data necessary to evaluate an agent's performance in the system. For each agent, we record all the values necessary to construct a confusion matrix: the true positives, the true negatives, the false positives, and the false negatives.

A confusion matrix is a visualization tool where each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. Confusion matrixes are used to see if the system is confusing two (or more) classes, for instance, commonly mislabelling one as another.

Alongside this data, we also record the recall, precision, accuracy and root mean squared error obtained by each algorithm. Reinforcement Learning agents record an extra measure called the F2-score for each class. F2-score is a special case of F-score as we will see next.

The F-score is the weighted harmonic mean of precision and recall. It's a measure of a test's accuracy. The traditional F-score evenly weights precision and recall, that's why it's sometimes referred as F1 score:

$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

meaning that it's a special case of the general F_β measure:

$$F_\beta = (1 + \beta^2) \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\beta^2 \cdot \mathrm{precision} + \mathrm{recall}}.$$

For the Reinforcement Learning agent we decided to use the F_2 measure, which weights recall twice as much as precision, since we prioritize improving recall (detecting turning points correctly) rather than precision.

7.3 Web Application

In this section we present our first client application for BROMAS: a Rails-based Web application. As we have stated earlier, Ruby on Rails is a Model-View-Controller framework for agile development of Web applications. Rails is written in Ruby, a dynamic, multi paradigm programming language. We know that such a combination of different technologies can cause big headaches, but the benefits delivered compensate this. We will start by introducing the web application architecture. Afterwards, we will describe the BDD development process and the data visualization tools used.

7.3.1 Architecture

We have repeatedly mentioned the Model View Controller pattern but, what is it exactly?

The Model View Controller is an architectural software engineering pattern. This patters aims to isolate the *business logic* from the *user interface*. By separating software into different layers, we can alter the implementation of one layer without affecting the others. Coupling is reduced, and code is easier to write and maintain.

The model represents the data of the application, it's responsible for maintaining the state of the application while enforcing the business rules that apply to that data. The view is responsible for generating a user interface that contains the application's data. The view doesn't handle any input data, its work is done once it has been displayed. Controllers are the orchestra's director. They receive events from the outside world, interact with the models, and present the corresponding view to the user.

MVC pattern was originally intended for conventional GUI applications, but has been adopted for Web applications too. Ruby on Rails is a great example of how to use the MVC architecture to develop Web applications.

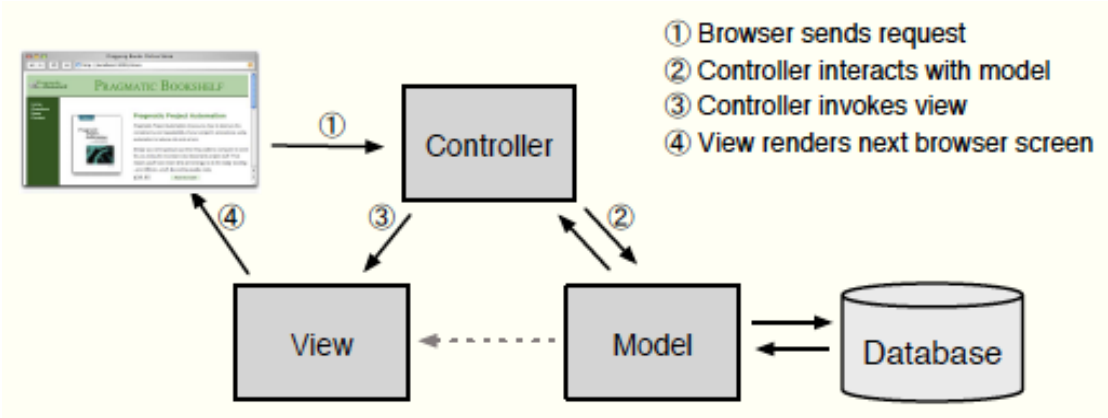


Figure 7.19: MVC in Web applications

Rails applications use an extra element: the router. The router is the responsible for redirecting an incoming event to the corresponding method of a certain controller.

The architecture chosen for BROMAS is a bit special since we're using the shared Data Sources we defined earlier. This means that we won't be able to fully use Rails' ActiveRecord, an Object-Relation mapper responsible for converting Ruby objects to SQL and vice versa. BROMAS web application consists of two models and two controllers.

The Ticker model represents a certain ticker and is responsible of fetching the Stock Data related to its ticker from the BROMAS Data Sources. The other model is called the Execution model. As its name suggests, it represents the executions that BROMAS has completed. It's responsible for fetching from the Stats and Results sources the information related to its Execution.

Both Models are connected through a belongs_to/has_many relationship. This relationship means that each Execution has a related Ticker. This way we can access from the Execution to all the information available of its Ticker. This will be of key importance when we realize the Data Visualization.

As we have explained, the Controllers are responsible for managing the communication between layers and with the external world. We have decided to implement two Controllers, one per Model class, because this way the URL routes generation is more straightforward. By using two controllers, we can easily tell the application router to redirect the requests pointing to http://myserver/executions to the Executions controller. The same happens for the Tickers controller. This way, the URLs of our application are easily remembered by our users, who can access all the information available of a company by opening the URL http://myserver/tickers/company in his browser.

Two views are currently available: Show and Index. Index views present the user the full list of executions or tickers available in BROMAS. The Show view presents all the information available of a given execution or ticker. It's in this view where the Data Visualization will be implemented.

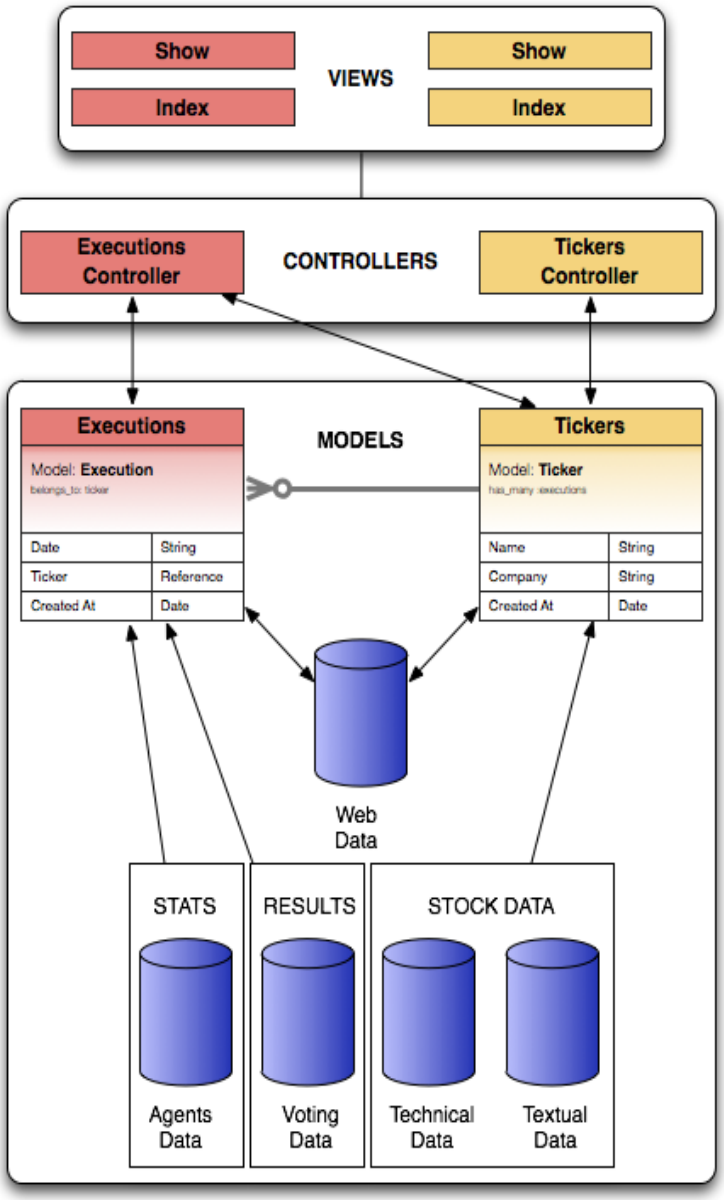


Figure 7.20: BROMAS Rails Application

7.3.2 Visualization

As we stated in chapter 3, one of the main goals of BROMAS is to present data in a meaningful way, while trying not to overwhelm users with too much information. Data Visualization is a very interesting field of research, specially for those that, like us, try to discover new knowledge from data. Designing a Data Visualization system for Stock Analysis could perfectly be a Master Thesis on its own, so we have narrowed this objective a bit.

Our first decision was to entirely develop our Visualization tools. This would allow us to control everything and experiment with new approaches. Unfortunately, to do so in a Web environment requires knowledge in Rich Internet Applications technologies, at least till the arrival of the future HTML5. Despite our dislike of Flash technology, we considered that Adobe's Flex was the best solution in our case. It provides all the necessary elements and integrates very well with Rails applications.

Figure 7.21: Yahoo! Finance Interactive Chart

After several days getting to know Flex we found out two things. One: the decision of choosing Flex was correct, and two: we wouldn't have enough time to develop a full set of visualization tools for the Thesis presentation date. Even giants like Google and Yahoo needed some time to launch their interactive stock chart tool. We were not going to be able to implement a full set of visualization tools from scratch.

Luckily for us, among the vast amount of APIs that Google provides for free to developers, there's one called Google Visualization API. Despite still being a beta, the Visualization API provides a large amount of different visualizations, including annotated time lines, bar charts, pie charts, the popular motion chart, …

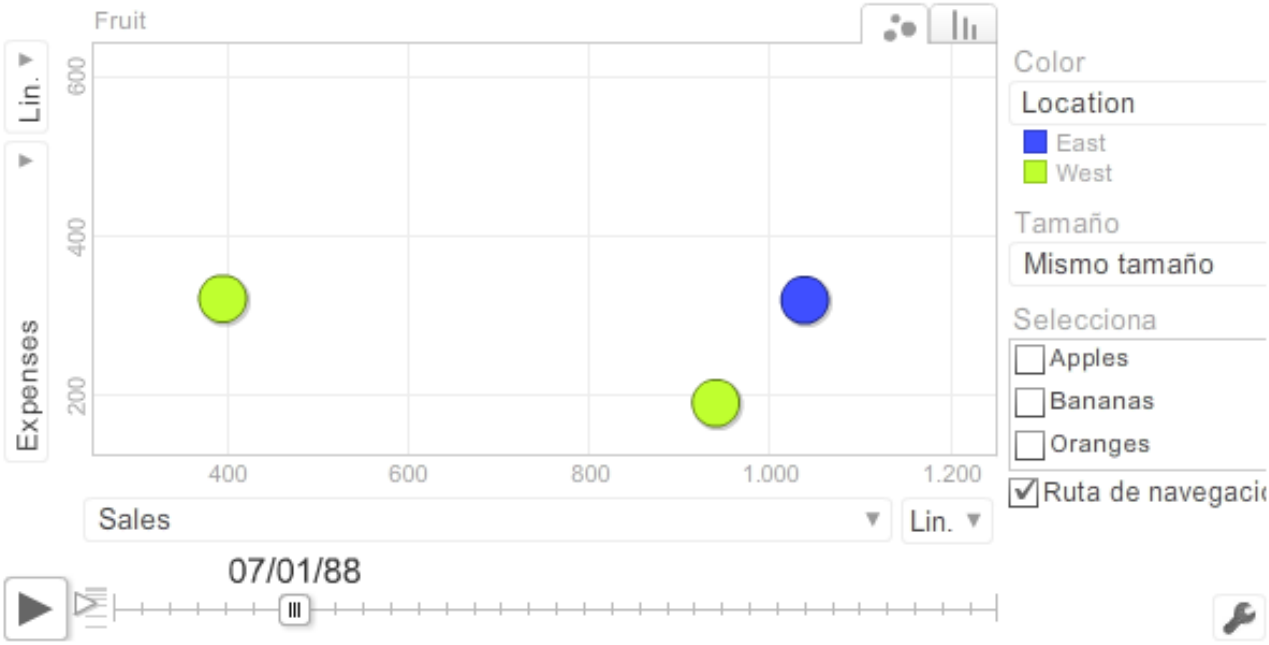


Figure 7.22: Google's Motion Chart

For BROMAS we decided to use the annotated time line, the line chart, and the motion chart. The annotated time line will be used for displaying the historical evolution of price. We have decided to use annotated time line since it provides the ability to “zoom” the data by setting the start and end dates. The simple line chart will be used to display the historic evolution of other indicators such as the volume, the relative strength index,... Finally, the motion chart will be used to display the evolution of agent's performance during the execution.

Although being really good visualizations, this API has two main drawbacks for us: since it's an external tool, we can't fully adapt it to our specific needs. There are some features, that we would like for BROMAS, that currently can't be achieved using Visualization API. Another important drawback is the necessity to transform the data to one of their supported formats. We have done too many data transformations for BROMAS, and we're unwilling to do another one just for Visualization API.

Due to this and to the fact that we don't like to be too dependent of another company, we have decided to continue with our initial plans of developing our own set of visualization tools. Visualization API is a great product, but it doesn't offer all the features we want.

7.3.3 Behavior Driven Development

In chapter 6, we introduced the concepts of Test Driven Development and Behavior Driven Development. This section will present how BROMAS' web application has been implemented using this methodologies.

There are several tools that provide support for Behavior Driven Development, being the most renowned RSpec and Cucumber.

RSpec provides a Domain Specific Language with which developers can express executable examples of the expected behavior of the code. In BROMAS we have used RSpec to define the test cases for the Controllers and Models.
Cucumber is a recently launched BDD tool that can execute plain-text documents as automated functional tests. It's main advantage is the use of a language called Gherkin to describe the application's behavior. Gherkin is a business readable Domain Specific Language that lets the developers describe behaviors without detailing how it is implemented. Another interesting feature is that Gherkin's grammar exists for many different languages (including Spanish), allowing developers to write tests in their preferred language.

Cucumber test files serve two purposes: automated tests and code documentation. Because of all this, Cucumber helps to close the existent gap between developers and non technical participants of the project.

Chapter 8

Testing and Execution Results

In this chapter we will describe the process we have followed to test the BROMAS system. Firstly, we will briefly explain the tests created following the concepts of the Test Driven Development methodology previously introduced. Following, we will focus in the BROMAS tuning and testing processes, including parameter tunings, machine learning methods training, etc. Finally, we will present and comment the results obtained by each agent individually and the system as a whole.

8.1 Testing Environment

Testing software is one of the most tedious activities a software developer has to do. Not only tests have to be written, but they must be ran every time a change is made to the source code. In order to leverage this, several tools are provided to make testing easier.

To develop BROMAS we have used the Eclipse IDE which provides integration with JUnit, a framework for writing and executing automated tests in Java. Following the Test Driven Development methodology, we created test cases for every testable class in the Multi Agent system. We created test cases for the following processes:

- Data Source management
- Artificial Neural Network training
- Decision Tree training
- Support Vector Machines training
- Reinforcement Learning training

In the Data Source management test, we check that the entire process of technical information management is done correctly, from its retrieval from Yahoo Finance! Service to the creation of balanced windowed data, including technical indicators calculation, and data labeling.

Regarding Machine Learning methods, we tested that train and test processes are correctly executed. In the case of RapidMiner-based methods, we also tested that the results of the execution can be correctly imported to BROMAS to create the corresponding Vote message. And in the case of Reinforcement Learning, we also tested that the data split into train and test datasets, and 10-fold cross-validation are done properly.

Since the system testing is performed in a different machine exclusively prepared for it, we had also to develop a mechanism to ease the deployment of BROMAS. To do so, we used a combination of Subversion repository and ANT.

Subversion is a version control system that is used to maintain current and historical versions of files. In our case, we use it to store the BROMAS source code, the necessary libraries, and the documentation. This way not only do we keep everything in a safe place, but we can roll back to a previous version in case some modification went wrong.

Apache ANT is a software tool for automating software build processes. ANT uses XML files to describe the build process and its dependencies, and makes trivial to integrate JUnit tests with the build process. Using ANT, we were able to, with one single command, download the latest version of BROMAS from the SVN repository, compile the code, run all the tests, and start the BROMAS platform.

8.2 BROMAS Tuning and Testing

Machine Learning algorithms require parameter tuning in order to adapt them to the particulars of a training set. Parameter tuning is not a trivial task, requiring expert knowledge in how parameters should be chosen for a given task. Each Machine Learning has its own set of parameters to be tuned, thus requiring us to treat each case individually.

Another advantage of using RapidMiner as a library, is that we could use its parameter tuning operators to tune the Machine Learning algorithms we were using in BROMAS. To do so, RapidMiner provides the GridParamaterOptimization operator. As its name suggests, this operator realizes a grid search to find the optimal values of the specified parameters. A grid search is a simple global optimization method: an utility function is evaluated through all the points of a grid aligned to the coordinate axis of the parameter space. The example with the highest value is an estimation of the global maximum. We have used the GridParameterOptimization to tune the following parameters:

- Artificial Neural Network
  - Learning rate: used in the backpropagation algorithm, moderates how much the weights' value is changed at each step.
  - Epochs: number of total iterations of the backpropagation algorithm.

- Support Vector Machine
  - Kernel: the type of kernel function used.
  - C: the cost parameter.
  - Gamma: the gamma parameter present in some kernel functions.

- Decision Tree
  - Criterion: Specifies the criterion used for selecting attributes and numerical splits.
  - Confidence: The confidence level used for the pessimistic error calculation of pruning.
  - Maximal Depth: sets the upper limit for the generated tree's depth.

In the case of SVMs, we first run the optimization process to select the type of kernel function used, keeping the rest of the parameters at their default value. This was done due to the fact that depending on the type of kernel function used, there are different parameters to optimize. Once the kernel type parameter was fixed, we could then optimize the others. The same happened in the case of Decision Trees: first of all, we had to discover which criterion would fit the data best.

The decision of which parameters to optimize was taken after deliberating with our Thesis Director. As we said in the introduction to this section, parameter tuning is an expert task on its own. And since we weren't an expert, we required some help to do it properly. Our first attempts to optimize ANNs, ended up taking more than 3 weeks of calculation (we don't know exactly since a power shortage stopped the optimization, we bought a SAI system after that incident), meaning that we were trying to optimize too many parameters using value ranges too wide. Since we couldn't afford each calculation to take more than 3 weeks, we had to focus on the key parameters, and so we did with the help of our Director.

Regarding our implementation of Watkins-Q( ), we didn't have any external tool that could do the tuning for us. We had to implement everything, from the data split into train and test sets to the implementation of a utility measure to optimize, also including the 10-fold cross validation method.

In the case of this algorithm, the parameters to optimize where the following:

- Lambda: the eligibility traces
- Alpha: the learning rate
- Gamma: the discount rate
- Epsilon: the trade-off between exploration and exploitation

The parameter optimization process, which is common for all the methods, is clearly explained in the following diagram.

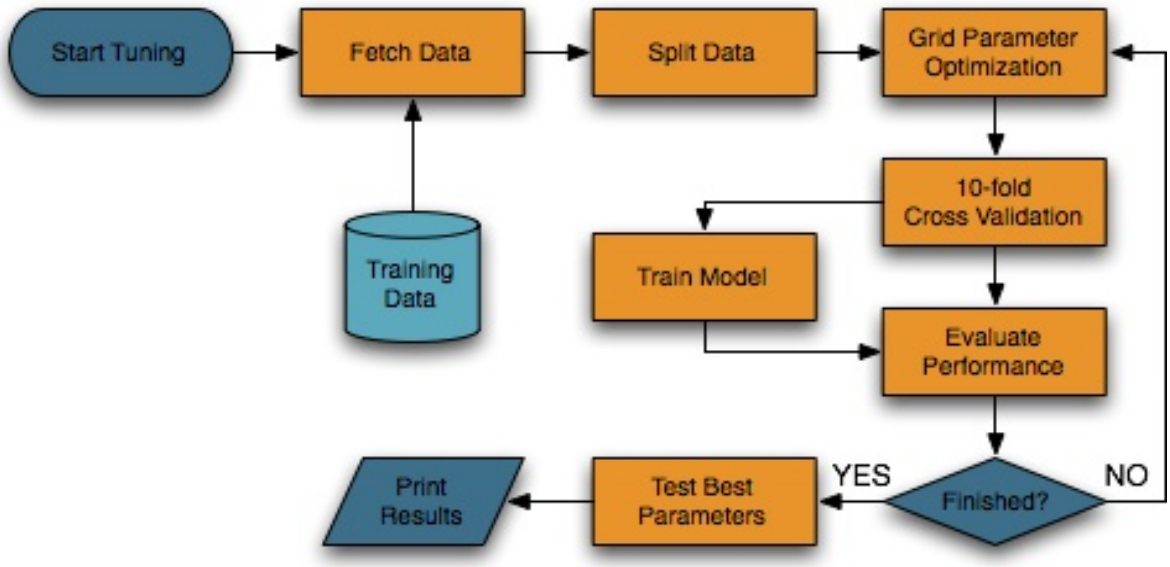


Figure 8.1: Parameter Tuning process

Following, we present the best results we obtained with the technical data of Apple Inc. (AAPL) for all parameter combinations.

Artificial Neural Networks

Accuracy	Absolute Error	Classification Error	Precision	Recall	Rate	Epochs
٠.٨٢٢	٠.٢١٠	٠.١٧٨	٠.٧٥٧	٠.٦٨٩	٠.٢	٦٠٠.٠
٠.٨٢٢	٠.٢٢٥	٠.١٧٨	٠.٨٠٠	٠.٦٨٩	٠.٤	٦٠٠.٠
٠.٧٩٣	٠.٢٤٧	٠.٢٠٧	٠.٧٣٣	٠.٧٠٧	٠.٥	٧٠٠.٠
٠.٧٨٥	٠.٢٥٦	٠.٢١٥	٠.٧١٩	٠.٦٨٢	٠.٢	٥٠٠.٠
٠.٧٦٧	٠.٢٧٣	٠.٢٣٣	٠.٦٧٢	٠.٦٣٢	٠.٢	٤٠٠.٠

Support Vector Machine

First, we ran the process to obtain the most suitable kernel type for our data:

Accuracy	Absolute Error	Classification Error	Precision	Recall	Kernel
0.670	0.330	0.330	NaN	0.341	linear
0.667	0.333	0.333	NaN	0.333	poly
0.874	0.126	0.126	0.947	0.748	rbf
0.667	0.333	0.333	NaN	0.333	sigmoid

Once the kernel type was fixed to use a Radial Basis Function, we had to optimize the C and Gamma parameters:

Accuracy	Absolute Error	Classification Error	Precision	Recall	C	Gamma
0.944	0.056	0.056	0.974	0.889	655.360	0.0
0.941	0.059	0.059	0.973	0.881	501.760	0.0
0.941	0.059	0.059	0.973	0.882	92.160	0.0
0.937	0.063	0.063	0.971	0.875	829.440	0.0

Decision Tree

As in SVMs, we first had to choose which criterion would be used for our decision tree.

Accuracy	Absolute Error	Classification Error	Precision	Recall	Criterion
0.752	0.326	0.248	0.628	0.560	gini_index
0.748	0.328	0.252	0.658	0.584	information_gain
0.730	0.406	0.270	0.620	0.458	gain_ratio
0.670	0.491	0.330	0.391	0.341	accuracy

Since Information Gain obtained better precision and recall results, we decided to use it instead of Gini index. Following, we optimized the Confidence and Maximum Depth parameters.

Accuracy	Absolute Error	Classification Error	Precision	Recall	Confidence	Depth
0.811	0.227	0.189	0.723	0.717	0.150	20.0
0.800	0.228	0.200	0.715	0.724	0.200	17.0
0.796	0.240	0.204	0.700	0.699	0.300	17.0
0.796	0.258	0.204	0.717	0.699	0.400	17.0

Q-Learning

Unfortunately, our home-made optimizer doesn't output the data like RapidMiner's does, so we can only present the chosen values for the parameters.

Alpha	Gamma	Lambda	Epsilon
0.100	0.999	0.975	0.400

8.3 Tests Executed

In this section we will describe the tests that have been executed before commenting the results obtained. In this section we plan to answer the question that has been haunting us since the beginning of this Thesis: is it possible to predict the turning points in stock trends?

We will start this section with the first test we executed: learning with an artificially balanced dataset versus learning with the original dataset. We have already mentioned that due to the fact that our data's class distribution was extremely unbalanced we had implemented a data balancer mechanism that downsampled the main class while upsampling the minority classes, resulting in a class distribution of 50%-25%-25%. Since we were concerned about the performance a model trained with a differently distributed data set would achieve, we decided to compare how our machine learning algorithms performed in both situations.

We decided to test our system with five different companies: Apple (AAPL), Coca Cola (KO), Microsoft (MSFT), Pfizer (PFE), and Yahoo! (YHOO). This way we test the system with companies from different sectors and of different age. We have tested for each the performance of three different algorithms implemented in BROMAS using balanced training data and unbalanced training data. Reinforcement Learning analyst isn't included since it doesn't make use of windowed data.

Yahoo Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	64.52% +/- 17.12%	35.48% +/- 17.12%	58.69%	50.76% +/- 11.66%	0.539 +/- 0.142
ANN Test	5.78%	94.22%	37.10%	32.62%	0.965
Dtree Train	62.00% +/- 10.41%	38.00% +/- 10.41%	57.61%	45.56% +/- 9.81%	0.593 +/- 0.077
Dtree Test	12.47%	87.53%	33.41%	32.44%	0.935
SVM Train	81.37% +/- 14.09%	18.63% +/- 14.09%	75.15%	90.95%	0.397 +/- 0.169
SVM Test	92.49%	7.51%	33.33%	NaN	0.274

Yahoo Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	74.69% +/- 13.83%	25.31% +/- 13.83%	35.97%	39.52%	0.463 +/- 0.150
ANN Test	88.62%	11.38%	39.41%	40.99%	0.324
Dtree Train	63.81% +/- 15.26%	36.19% +/- 15.26%	35.32%	32.85% +/- 2.92%	0.559 +/- 0.157
Dtree Test	90.65%	9.35%	34.27%	34.38%	0.301
SVM Train	78.14% +/- 12.13%	21.86% +/- 12.13%	33.33%	NaN	0.430 +/- 0.185
SVM Test	92.48%	7.52%	33.33%	NaN	0.274

The results clearly show that balancing data in this scenario is counterproductive. Sometimes, the algorithms show some improvement in recalling the minority classes Up and Down, but in other cases the performance of the algorithm is drastically reduced. Decision Trees and Artificial Neural Networks, are the methods that suffer from this performance loss when the class distribution of the training data doesn't match the distribution in test data. Instead of always predicting the majority class, algorithms start to predict basically minority classes, improving their recall, but keeping their precision very low. This causes the algorithm's accuracy to drop to values even lower than 10%. While it's true that we implemented a data balancing system to improve the recall, we wanted to improve the mean recall, which hasn't changed at all. Surprisingly, only Coca Cola escapes from this performance loss, maintaining it's performance stable.

CocaCola Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	74.56% +/- 16.76%	25.44% +/- 16.76%	70.00%	77.82%	0.426 +/- 0.188
ANN Test	71.96%	28.04%	31.41%	33.13%	0.514
Dtree Train	98.69% +/- 1.39%	1.31% +/- 1.39%	99.11%	98.32%	0.084 +/- 0.077
Dtree Test	98.6%	1.4%	33.32%	NaN	0.118
SVM Train	99.98% +/- 0.07%	0.02% +/- 0.07%	99.97%	99.98%	0.005 +/- 0.015
SVM Test	98.63%	1.37%	33.33%	NaN	0.117

CocaCola Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	99.03% +/- 0.84%	0.97% +/- 0.84%	34%	44.13%	0.093 +/- 0.038
ANN Test	98.63%	1.37%	33.33%	NaN	0.122
Dtree Train	98.09% +/- 1.39%	1.91% +/- 1.39%	33.69%	33.53%	0.124 +/- 0.040
Dtree Test	96.14%	3.86%	35.25%	34.29%	195
SVM Train	99.05% +/- 0.85%	0.95% +/- 0.85%	33.33%	NaN	0.090 +/- 0.037
SVM Test	98.63%	1.37%	33.33%	NaN	0.177

In fact, if we closely look at the confusion matrix of the Artificial Neural Network, we have a clear example of what we were planning to obtain from using a data balancer.

KO-ANN-UNBAL-TEST	true Stay	true Down	true Up	class precisior
pred. Stay	3523	21	28	98.63%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

KO-ANN-BAL-TEST	true Stay	true Up	true Down	class precisior
pred. Stay	2566	22	10	98.77%
pred. Up	936	6	11	0.63%
pred. Down	23	0	0	0.00%
class recall	72.79%	21.43%	0.00%	

As we see, we moved from always predicting the majority class (Stay) to predict sometimes the minority class, despite not being too precise. In the following confusion matrix, we see the results obtained using a Decision Tree for predicting Apple. Again, the algorithm suffers from the same issue, it predicts minority classes more, but with a very low precision.

AAPL-DTREE-BAL-TEST	true Stay	true Up	true Down	class precisior
pred. Stay	418	10	11	95.22%
pred. Up	796	33	23	3.87%
pred. Down	539	21	14	2.44%
class recall	23.84%	51.56%	29.17%	

In summary, despite balancing data has forced the algorithms to predict some times the minority classes, the system is still unable to classify them correctly when the class distribution is so unbalanced. For instance, the Support Vector Machine achieves a 98.63% accuracy by predicting Coca Cola always “Stay”.

Another possible solution is reducing the threshold used to label data from 5% to a lower value, in order to decrease the amount of examples of the majority class and increase the number of Up and Down elements. In Yahoo, for instance, by reducing the threshold to a 3% we move from having 257 elements of Down class to having 530. Despite this, performance is not improved as much as we would expect by duplicating the number of examples.

YHOO	Accuracy	Class.Error	Mean Recall	Mean Precision	RMSE
ANN-BAL-TRAIN	66.42% +/- 22.02%	33.58% +/- 22.02%	60.87% +/- 11.63%	62.52% +/- 8.76%	0.511 +/- 0.179
ANN-BAL-TEST	10.43%	89.57%	32.92%	29.35%	0.93
SVM-BAL-TRAIN	60.05% +/- 16.86%	39.95% +/- 16.86%	44.04% +/- 11.92%	85.21%	0.616 +/- 0.141
SVM-BAL-TRAIN	80.97%	19.03%	33.33%	29.99%	0.436
DTREE-BAL-TRAIN	62.00% +/- 10.41%	38.00% +/- 10.41%	57.61%	45.56% +/- 9.81%	0.593 +/- 0.077
DTREE-BAL-TEST	15.28%	84.72%	30.08%	31.45%	0.918 +/- 0.000
ANN-UNBAL-TRAIN	51.81% +/- 15.71%	48.19% +/- 15.71%	34.25% +/- 3.27%	35.81% +/- 5.46%	0.640 +/- 0.114
ANN-UNBAL-TEST	70.05%	29.95%	37.58%	38.01%	0.514
SVM-UNBAL-TRAIN	58.64% +/- 16.33%	41.36% +/- 16.33%	33.33%	19.55%	0.627 +/- 0.145
SVM-UNBAL-TEST	80.91%	19.09%	33.33%	26.97%	0.437
DTREE-UNBAL-TRAIN	47.32% +/- 10.48%	52.68% +/- 10.48%	33.86% +/- 3.05%	34.36% +/- 2.85%	0.701 +/- 0.075
DTREE-UNBAL-TEST	63.76%	36.24%	33.78%	34.07%	0.588

There's a slight improvement, specially when used Balanced data sets. The increase of minority class proportion has “forced” the algorithms to classify more examples as such.

Despite having similar mean recall and mean precision values, precision and recall of minority classes have also improved (at expenses of the majority class, whose classification performance has decreased). From our point of view, the 3% configuration looks promising, and thus, more tests will be realized to verify this initial impressions. Reducing the threshold even further is not considered, since it would be too sensitive to noise.

We think that artificially balancing data is the way to deal with this problem, but we must test the performance obtained by using different class distributions. The current 50-25-25, doesn't work correctly. We think that this is probably due to the fact that we upsampled too much the minority classes and thus, the system was overfitted to classify correctly only those examples correctly. The results show that when using balanced data, algorithms predict more minority classes, but with a very poor precision.

Due to all this, we have decided not to use balanced training data for Production till we improve the precision when predicting minority classes. Another aspect we must check is the data we're using for training and testing. Maybe we're not using the correct attributes despite being recommended by an expert in the field.

After resolving the question of wether using or not artificially balanced data, we focus in how each machine learning algorithm performed. In addition to the result tables previously presented, here we include the confusion matrixes obtained. This way, we can visually check what our models are doing. Here we will show the results obtained for AAPL, for the complete results, please check Appendix B, which includes all the performance measures and confusion matrixes of the tests performed.

AAPL-ANN-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1705	60	46	94.15%
pred. Up	14	2	1	11.76%
pred. Down	33	2	1	2.78%
class recall	97.32%	3.12%	2.08%	

AAPL-SVM-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1752	64	48	93.99%
pred. Up	0	0	0	0%
pred. Down	0	0	0	0%
class recall	100%	0%	0%	

AAPL-DTREE-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1126	41	25	94.46%
pred. Up	186	10	12	4.81%
pred. Down	440	13	11	2.37%
class recall	64.27%	15.62%	22.92%	

As we can see, the most interesting algorithms for this task are Artificial Neural Networks and Decision Trees. Even though data is extremely unbalanced, they try to discern the minority classes. The trained Support Vector Machine prefers not to risk, and opts for always predicting the majority class. We may have to rerun again our parameter tunning process to try to improve SVM's performance.

The most interesting algorithm from our point of view is Reinforcement Learning.

Q-Learning	Accuracy Train	Accuracy Test
Apple	57.44%	70.33%
CocaCola	67.84%	71.00%
Microsoft	66.89%	71.69%
Pfizer	63.35%	70.92%
Yahoo	48.96%	61.07%

As we can see, the Q-Learning agent obtains a very decent performance for every company. And since the accuracy values are far from the 80-90% that we obtained, this means that Reinforcement Learning agent tends to predict more minority classes, in part due to the exploratory actions taken from time to time.

AAPL-QL-TEST	true Stay	true Up	true Down	class precisior	f۲ score
pred. Stay	١٢٨٤	٣٨	٢٧	٩٥.١٨%	٠.٧٦٨
pred. Up	٢٥٠	١٧	١١	٦.١١%	٠.١٥٩
pred. Down	٢١٨	٩	١٠	٤.٢٢%	٠.١١٦
class recall	٧٣.٢٩%	٢٦.٥٦%	٢٠.٨٣%		

With Reinforcement Learning, Apple is the one with the best overall performance. Accuracy is lower than the one obtained for Microsoft or Coca Cola, but the average precision and recall is higher. We can notice from this confusion matrix that we have the same problem as with the other algorithms: a very low precision value for minority classes.

YHOO-QL-TEST	true Stay	true Up	true Down	class precisior	f۲ score
pred. Stay	٥٨٩	٢٣	٢٣	٩٢.٧٦%	٠.٦٨٩
pred. Up	١٥٠	١١	١٠	٦.٤٣%	٠.١٦٦
pred. Down	١٧١	٦	١	٠.٥٦%	٠.٠١٦
class recall	٦٤.٧٢%	٢٧.٥٠%	٢.٩٤%		

Yahoo, on the contrary, is the worst performing stock with Reinforcement Learning, clearly caused by the low results obtained for the Down class: the most difficult class of the three.

But, if the results are more less the same as the ones achieved by other algorithms, why do we say that Q Learning is the most interesting algorithm? Well, we have based this opinion in the fact that Reinforcement Learning has something that the other algorithms don't that can be used for optimize it's results: the reward function.

Currently, the reward function implemented is quite simple: gives a positive reward every time the agent is correct, and gives a negative reward when the agent is wrong. The negative reward value is the double of the positive, meaning that the agent will seek to minimize the possibilities of being wrong. In this scenario, this means that the agent will give preference to the Stay class.

By playing with the reward function, we can force the agent to seek the objective we want to achieve. In this case, we have to translate into a reward function the objective of maximizing the average F-Score.

And finally, it's time to test the entire Decision Making process by using all agents simultaneously. This way we will be able to test if our agents complement each other, as it *should* happen in a real world team of analysts. To do so, we present the overall performance measures of the system plus examples of the decisions made during the execution by all the agents present.

Due to a bug we discovered in JADEX, we were almost unable to perform a global execution for Coca Cola due to the amount of historic data available for this company. After several attempts, we managed to execute a complete test. The results obtained by BROMAS for Apple, Coca Cola, Microsoft, Pfizer, and Yahoo! are summarized in the following table.

	fscore_stay	fscore_up	fscore_down	Accuracy
YHOO-RL	٠.٦٦٤	٠.١٣٤	٠.٠٧٥	٠.٥٨٨
YHOO-SVM	٠.٩٨٤	٠	٠	٠.٩٢٥
YHOO-DTREE	٠.٩٨٤	٠	٠	٠.٩٢٥
YHOO-ANN	٠.٩٤٦	٠.٠٩٦	٠	٠.٨٨٣
YHOO-TRADER	٠.٩٨٢	٠	٠	٠.٩٢٣
MSFT-RL	٠.٧١	٠.٠٢٧	٠.٠٢٣	٠.٦٤٧
MSFT-SVM	٠.٩٩٤	٠	٠	٠.٩٦٩
MSFT-DTREE	٠.٨٤١	٠	٠.٠١٢	٠.٧٩
MSFT-ANN	٠.٩٩٢	٠	٠	٠.٩٦٧
MSFT-TRADER	٠.٩٦٩	٠	٠	٠.٩٣٩
KO-RL	٠.٦٣٩	٠.١٦٩	٠.٠٦٣	٠.٥٦٥
KO-SVM	٠.٩٨٤	٠	٠	٠.٩٢٥
KO-DTREE	٠.٩٨٤	٠	٠	٠.٩٢٥
KO-ANN	٠.٩٤٦	٠.٠٩٦	٠	٠.٨٨٣
KO-TRADER	٠.٩٨١	٠	٠	٠.٩٢٢
PFE-RL	٠.٧٠٢	٠.١١٥	٠.١٤	٠.٦٢٨
PFE-SVM	٠.٩٨٤	٠	٠	٠.٩٢٥
PFE-DTREE	٠.٩٨٤	٠	٠	٠.٩٢٥
PFE-ANN	٠.٩٤٦	٠.٠٩٦	٠	٠.٨٨٣
PFE-TRADER	٠.٩٨٤	٠	٠	٠.٩٢٥
AAPL-RL	٠.٤٨٩	٠.١١٣	٠.٠٨٧	٠.٤٢٧
AAPL-DTREE	٠.٩٨٦	٠	٠	٠.٩٣٩
AAPL-SVM	٠.٩٨٧	٠	٠	٠.٩٤
AAPL-ANN	٠.٩٧٢	٠	٠	٠.٩٢٢
AAPL-TRADER	٠.٩٨٣	٠	٠	٠.٩٣٥

The results, taking in consideration the previous section, were to be expected. If the majority of agents tend to predict always the majority class, so will the Trader. But it's important to note that the Trader doesn't achieve the same scores as, for instance, the SVM powered analyst, the most greedy agent in BROMAS. It seems like Trader is acting as a smooth factor among all analysts. We will have to study this in more depth, once we manage to solve the problems that were detected in the previous section. Once we have a more heterogeneous analyst community, we will be able to see how well the Trader and its voting mechanism works.

Chapter 9

Conclusions

And finally, we reach the end of this Master Thesis. In this last chapter, we will present the learnings / facts we have deduced by creating the BROMAS system. We will analyze the project development process in terms of dedicated time and budget using, among others, a Gantt diagram. And last, we will present what we ware planning to include in future releases of BROMAS.

9.1 Final Conclusions

This was my first individual project in the field of Artificial Intelligence. Many hours have been invested in it, and if I have to be sincere, I'm really glad to finally be able of closing this phase.

BROMAS has been the perfect excuse to put into practice a good percentage of the knowledge that I've obtained during my stay in Facultat Informatica de Barcelona. After developing BROMAS system and obtaining the previously presented results, the following can be deduced:

- The use of a Multi Agent System combined with Machine Learning algorithms is a novel approach for analyzing and predicting Financial Markets.

- The main novelty of BROMAS is the use of a team composed of heterogeneous agents to analyze Stock Trends. Currently we have implemented 4 different analyst agents, but many more can be implemented and integrated.

- Due to the fact that very little research has been done in this area, there's a very big margin for improvement, meaning that creating a reliable Decision Support system is possible.

- Thesis' goals have been achieved thanks to the use of the technologies and methodologies presented during this document. The use of JADEX has allowed us to create this complex distributed system.

- Despite we knew from the very beginning that time series forecasting, and specially finance data, was very difficulty, we can't help but feeling a bit deceived for not been able to sort the issues regarding Minority classes.

- This encourages us to further research this area to try to find a suitable solution for this problem. Some new ideas were born, and we still have to try them.

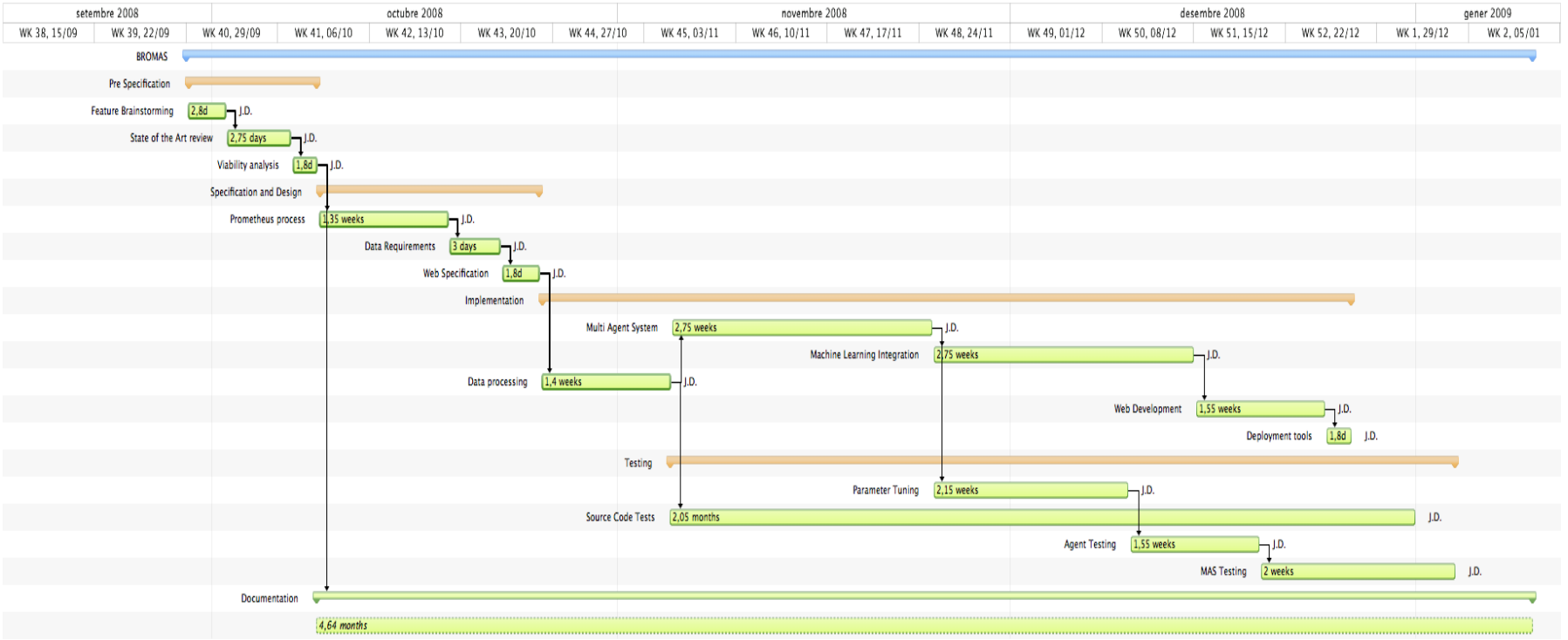
9.2 Planning

This project has been realized in our own and for our own, we depended just in ourselves to do it. Because of this, this project has been considered a full time work, which includes even the weekends.

As a software project, BROMAS follows a typical iterative process consisting of different phases, improving each as the projects goes on. This process start with a Requirements Analysis phase, in which we define the goals and scope of our system. Once we have defined the project's scope and goals, we move on to the Specification and Design phase. In this phase we define the general architecture of the system and the internal details of each component of the system. Once we know *how* to achieve the system's goals, it's time to implement the system we have designed. As soon as we have completed the implementation of the first component, the testing phase is started. During this phase we write and execute the tests required to assure that the system behaves as it was designed.

Another important aspect of every software is the documentation. Software developers are very lazy in everything related to documentation, and because of this we have decided to start writing it as soon as possible. By doing it little by little, documentation is no longer a painful task.

To get an idea of the amount of work required for each step, next we present the Gantt diagram of BROMAS project.


setembre 2008
octubre 2008
novembre 2008
desembre 2008
gener 2009
WK 38, 15/09
WK 39, 22/09
WK 40, 29/09
WK 41, 06/10
WK 42, 13/10
WK 43, 20/10
WK 44, 27/10
WK 45, 03/11
WK 46, 10/11
WK 47, 17/11
WK 48, 24/11
WK 49, 01/12
WK 50, 08/12
WK 51, 15/12
WK 52, 22/12
WK 1, 29/12
WK 2, 05/01
BROMAS
Pre Specification
Feature Brainstorming
2,8d
J.D.
State of the Art review
2,75 days
J.D.
Viability analysis
1,8d
J.D.
Specification and Design
Prometheus process
1,35 weeks
J.D.
Data Requirements
3 days
J.D.
Web Specification
1,8d
J.D.
Implementation
Multi Agent System
2,75 weeks
J.D.
Machine Learning Integration
2,75 weeks
J.D.
Data processing
1,4 weeks
J.D.
Web Development
1,55 weeks
J.D.
Deployment tools
1,8d
J.D.
Testing
Parameter Tuning
2,15 weeks
J.D.
Source Code Tests
2,05 months
J.D.
Agent Testing
1,55 weeks
J.D.
MAS Testing
2 weeks
J.D.
Documentation
4,64 months

As it can be seen, BROMAS was planned to be completed by January 2009. When we look at this diagram, we can't avoid remembering Murphy's Law: “if something can go wrong, it will”. This planning was possible, but in a world with no obstacles. But here, there are, and plans have to have enough margin to be able to cope with this unexpected hindrances. The margin we left was not enough. Since we couldn't finish the BROMAS for the fixed delivery date, we used this extra time to add new features, and test everything more thoroughly.

There are 2 main reasons for this delay. The first one is the amount (and importance) of unexpected complications that we have suffered during BROMAS' development. The second is that this project was planned as a full time work. Ironically, 2 weeks after starting it I was hired for a part time job. BROMAS became my free time work.

Going back to the complications, we think it's worth mentioning the three most important, responsible in a 85% for not delivering this Thesis last February.

The first one is the migration from Joone to RapidMiner. Once we had “completed” the implementation of the Multi Agent System, we started working on integrating Machine Learning libraries to our agent system. We first had decided to use individual libraries for each case, and we started this phase trying to integrate the Neural Network library Joone to BROMAS. After implementing the code necessary to transform data from our format to Joone input's format, we found out that it didn't work. Despite having cautiously followed the user guide, we were unable to make Joone calculate a thing. In addition to this, documentation was very poor and out of date, and the application online forum was filled with unanswered help requests.

After some days trying to make Joone work, we remembered that RapidMiner could be used as a library for any Java project. After checking its documentation, and user forum, we decided to switch from using independent libraries to use an all-in-one solution like RapidMiner. It costed us many lines of code, and changes in the voting protocol, but we managed not only to get Artificial Neural Networks working, but also Support Vector Machines, and Decision Trees.

Another big complication was the unbalanced data issue. At beginning we thought it was a matter of parameter tuning, but after 2 weeks of unsuccessful tests, we found that it was due to the data class distribution. In order to try to solve this issue, we implemented the data balancer that we have already described.

We reserved the best one for the end: a bug in RapidMiner. The same application that had saved us from having to deal with many small undocumented libraries, had a very annoying bug: data labels were mixed up during the testing phase. Stay examples were turned into Up, Stay data into Down, etc... To make things worse, this mixing seemed to depend on the data and algorithm used, making very hard to find a solution to it. After reviewing the code several times, reimplementing it, doing tests from RapidMiner's GUI and from the API, we found in the forums that it was RapidMiner's fault, not ours. It was caused by the use of nominal attributes in our data.

From all this, we extract the following conclusions:

- Leave enough margin to solve any inconvenience: expect the unexpected.
- Rigorously review the documentation and online resources for every third party tool you plan to use.

9.3 Economic Analysis

In this section, we will detail the full economic cost of constructing the BROMAS system. To do so, we have to calculate all the expenses generated during the process, from Personnel to Hardware.

Regarding the Human resources, we have to state that this project has been entirely developed by one person, who has played all the roles required in a software project, from Analyst to Interface “Designer” (sorry dear Designers). We have fixed the salary to the value that our school fixes for internships : 6.5 € per hour.

Concept	Category	Value
Human Resources	All Roles – 1440 hours	9,360.00 €
Hardware	Testing Machine	300.00 €
Materials	Books	45.00 €
Others	Internet Connection	360.00 €
Total		**10,065.00 €**

The total cost of the project is 10065€, not bad taking into account that this project has lated 9 months, from October 2008 to June 2009. As it can be seen, the most costly concept is the human factor, despite fixing the salary to an internship level.

Thanks to the fact that almost all necessary equipment was already available, we have been able to limit the hardware and materials expenses to minimum levels, making this project very competitive.

9.4 Future Work

As we have stated several times through this document, the BROMAS system here introduced is just the beginning, We plan to continuously improve it by correcting its failures and by adding additional features. Following, we present the list of features that we plan to integrate in BROMAS for the future release.

- Improve the performance when predicting minority classes. This includes testing various class distributions using our already built balancing mechanism, fine tune the reward function of the reinforcement learning agent, and improve the voting mechanism used by the trader agent.

- Switch to Online mode: currently BROMAS is an on-demand system, only analyzes stocks when a user requests it. In the future, the system should be totally online, meaning that it has to be more autonomous. This way, when the user requests something the answer will be already available. This will also allow us to “plug” the system into online trading games, or similar.

- Improve the current web application: right now, the web application is more a Proof of Concept than a real application. In order to turn it into an usable product, the following changes will be made:
  - Improve the intercommunication with the MAS: we have to investigate the protocols necessary to guarantee a fluent communication between the web application and the Doorman agents.
  - Implement the User Interface: currently, the web application lacks of a proper and studied design. We have focused in implementing the visualization tools without taking into consideration the user experience.
  - Improve the Data Visualization tools: the tools used at the moment are all provided from Google's Visualization API. Though powerful, this API does not provide the freedom we need for developing our system, specially regarding data sources. Developing our own tools seems to be the wisest choice.

- Implement the Fundamental Data retrieval and analysis: without taking into account different sources of information, we can't fully comprehend market behaviors. In order to increase the reliability of our system, is necessary to include fundamental analysis, the other main trading strategy.

- Implement new analysis techniques, not necessarily relying on Machine Learning algorithms: we are currently studying new strategies to do stock analysis. We are currently solely using Machine Learning algorithms and we think that adding a new agent that doesn't use such techniques may be beneficial for BROMAS. But, since AI is our field of study, we are also considering new AI techniques such as data streams mining, which seems very well suited for the tasks our system realizes.

- Implement a Portfolio Recommendation system: currently, BROMAS only offers recommendations of a given stock. The user tells the system which stock he's interested in, and the system realizes a prediction about it. We want to offer our users more features to help them invest wisely. The first we plan to include is the Portfolio Recommendation. Portfolio Recommendation is not one but two new features: recommendation of possible portfolios and management of a given portfolio.

- Implement a monitoring mechanism: another feature that we think BROMAS should have is the possibility of monitoring a given stock (or portfolio). With Monitoring activated, users will be able to specify triggering rules and the actions to be executed once a rule is activated. These actions could be sending a notification, realizing a new analysis, deleting a stock from the portfolio, etc...

With the addition of all these new features, we will be able to achieve our biggest goal: transform BROMAS into an autonomous trading system.

Appendix A

JADEX

Jadex is a software to create goal-oriented agents using BDI (Belief, Desire and Intention) model. This software framework aims to make the process of developing agent based systems as easy as possible by building a rational agent layer to sit on top of a middleware and provides an intelligent agent construction and taking the advantages of software engineering foundation.

There are a lot of different platforms available such as Jadex, Jason, 2APL, etc. Most of these platforms have focused on a specific aspect of agent technology such as cognitive or infrastructure architecture and this is due to not being possible to focus on all aspects of agent technology in a single agent platform.

In order to increase the applicability of the software for a huge variety of domains, at least three categories of requirements can be considered when building an agent platform: Openness, Middleware and Reasoning. Openness focuses on the vision of interconnected networks of originally unrelated applications. Middleware emphasizes traditional software engineering concerns, i.e. service management, security and persistency aspects. Reasoning, on the other hand, is about agents internal decision making process using natural archetype models such as human or insects.

Based on these aspects, the existing agent platforms can be classified into two groups: FIPA-compliant, which focus on Openness and Middleware, and Reasoning-centered. FIPA-compliant platforms mainly address openness and middleware aspects, while Reasoning-centered platforms are concerned with behavior model of a single agent i.e. trying to achieve rationality and goal-directedness.

The existing gap between middleware and reasoning-centered systems is one of the main reasons to use and understand the Jadex BDI reasoning engine, which tries to bring together both research areas.

A.1 Abstract Architecture

A classic Jadex agent is formed by capabilities and a reasoning engine.
A capability is an encapsulated set of beliefs, goals and plans. Jadex agents have at least one capability and can even contain hierarchical structures of them.

The Reasoning Engine (also called Practical Reasoning Interpreter) consists of 2 interleaved components: the Means End Reasoning module and the Goal Deliberation module. The Means End Reasoning module reacts to messages, internal events and goals by selecting and executing plans. The Goal Deliberation module continuously deliberates about the current goals, to decide about a consistent subset, which should be pursued.

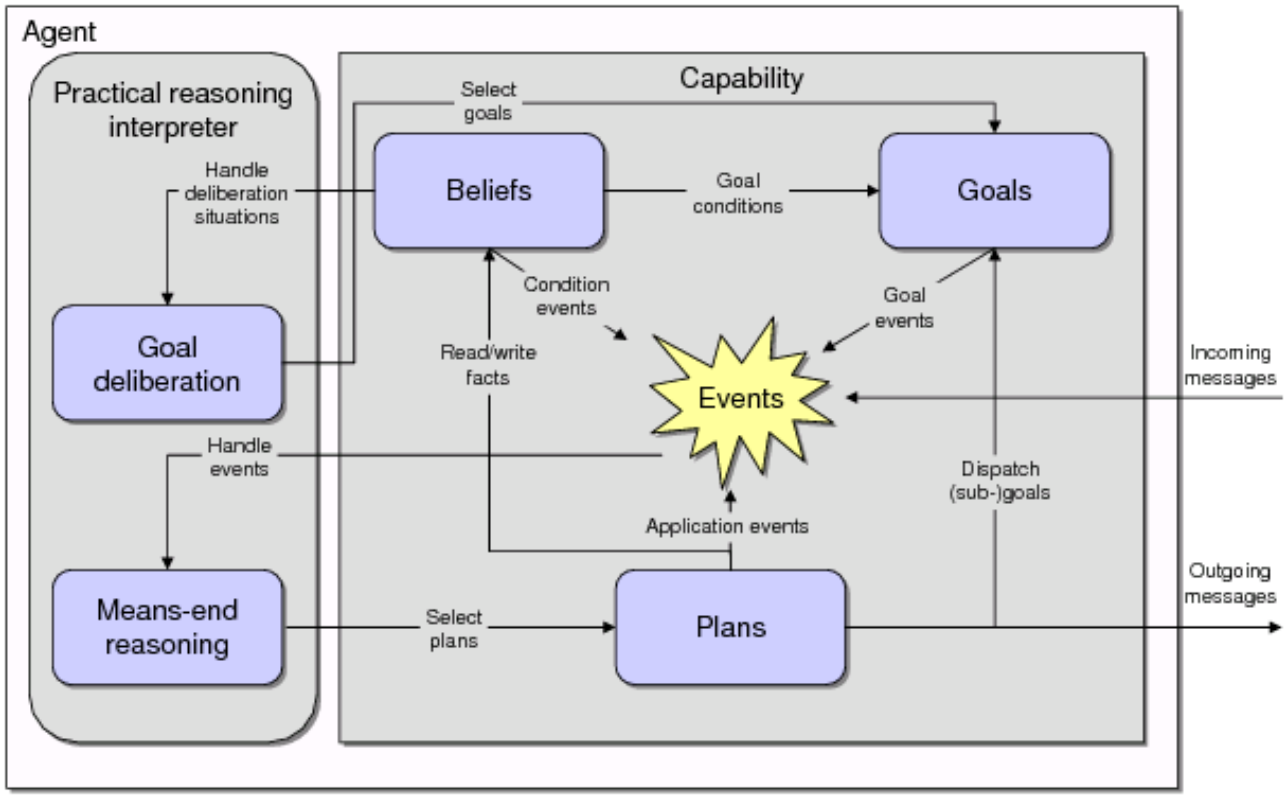


Figure A.1: JADEX Abstract Architecture

A.1.1 BDI Concepts

Jadex is 100% Object Oriented. Due to this, all BDI Concepts are implemented as Objects and no Formal Semantics are available.

Beliefs

Represent the agent's knowledge about its environment and itself. In Jadex the beliefs can be any Java objects. They are stored in a belief base as named facts or named set of facts. The beliefbase is not only a passive data store, but takes an active part in the agents execution, by monitoring belief state conditions. Changes of beliefs may therefore trigger actions such as events being generated or goals being created or dropped. Beliefs can be referenced in expressions using an OQL-like query language, as well as accessed and modified from plans using the beliefbase interface (a Java API).

Goals

Goals make up the agent's motivational stance and are the driving forces for its actions. Therefore, the representation and handling of goals is one of the main features of Jadex. Because goals are represented separately from plans, the system can retain goals that are not currently associated to any plan. As a result, unlike other BDI systems, Jadex does not require that all adopted goals are consistent to each other, as long as only consistent subsets of those goals are pursued at any time. To distinguish between adopted and actively pursued goals a Goal Lifecycle, consisting of the goal states option, active and

suspended, is introduced. Transition between these states is handled by the Goal Deliberation module.

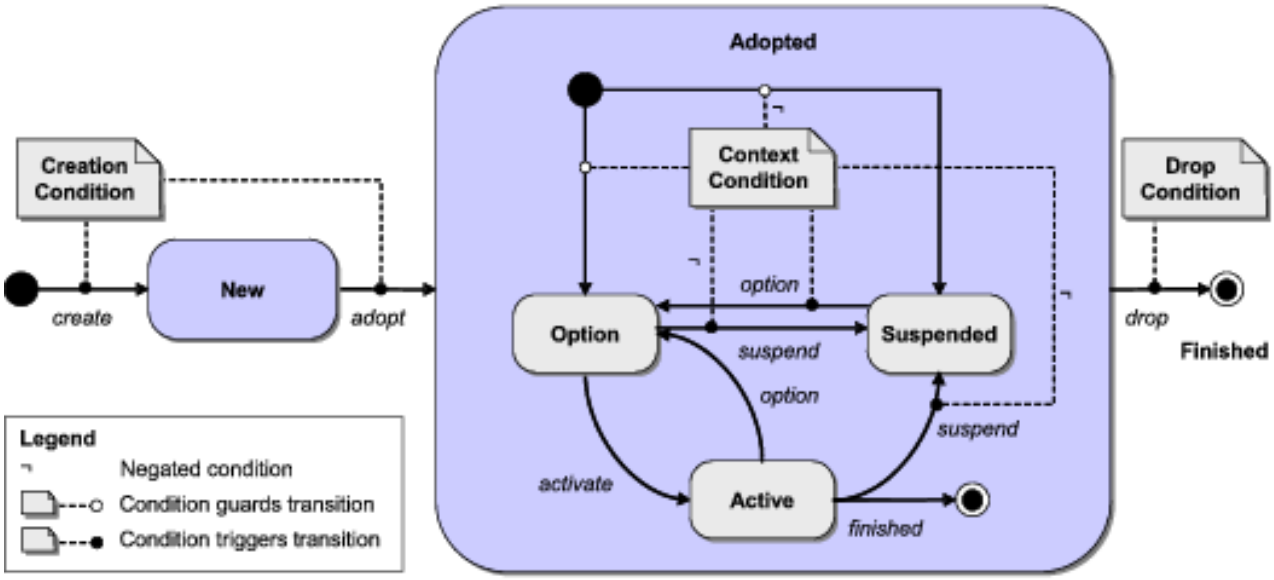


Figure A.2: Goal lifecycle

As can be noticed, all transitions have associated conditions that have to be fulfilled for the transition to take place. All conditions are specified by the user. Once a goal is adopted it is added to the agent's desire structure. If context conditions hold it becomes an option and thus can be activated at any time to be actively pursued. If not, it becomes suspended. A goal may be dropped in 2 situations: when the drop conditions are fulfilled or when it has been reached.

Jadex supports 4 different types of goals:

- Achieve Goal : Specifies a target state that has to be reached.
- Query Goal : Represents a need of information (that the agent has to gather).
- Perform Goal : States that some action should be done.
- Maintain Goal : Specifies a state that should be kept once reached.

And 2 goal levels:

- Top-Level Goal : Independent from any plan. Can be created from the agent/capability configuration files, triggered when the creation condition holds, manually from a plan or from external interactions.
- Subgoal : Created in the context of a plan. If the plan is terminated or aborted all related subgoals are dropped automatically.

Plans

Plans represent the agent's means to act in its environment. Therefore, the plans compose the library of actions the agent can perform. Depending on the current situation, plans are selected in response to occurring events or goals. The selection of plans is done automatically by the Means End Reasoning. In Jadex, plans consist of two parts: A plan head and a corresponding plan body. The plan head defines the Preconditions (conditions that have to be fulfilled for the plan to be selectable), the Context Conditions (conditions that have to be satisfied for the plan to be executed), the triggers (internal or external messages and goal events) for which the plan is applicable and the WaitQueue (where events are collected while the plan is not being executed). Another functionality of Jadex plans is the Priority attribute. Therefore, by combining a high priority value and trigger conditions Jadex agent can react to emergencies and sudden changes in the environment. The plan body encapsulates a recipe of actions written in pure Java. Therefore Jadex plan bodies can access any Java library and can be developed in a common Java IDE.

Meta-Level Reasoning

Jadex supports Meta-Level Reasoning. It consists of MetaGoals, Meta-Plans and MetaActions. Whenever an event or goal is executed and it is determined that meta-level reasoning needs to be done (for example, select an applicable plan for the event/goal) the corresponding MetaGoal of the goal or event is created and dispatched. In order to achieve this MetaGoal, MetaPlans are then created and executed. Each MetaPlan is composed of several MetaActions.

A.2 Execution Model

Jadex employs an agenda based execution scheme as described in [Pokahr05]. The interpreter consists of an agenda component holding the scheduled meta-actions to execute. The basic mode of operation is simple: The agent selects a MetaAction from its agenda and executes it when the the action's preconditions hold. Otherwise the action is simply dropped. The execution of the action may produce further actions that are added to the agenda following a customizable insertion strategy. Currently, the insertion strategy distinguishes between related actions (inserted as child nodes of current action) and unrelated actions (inserted in another agenda slot). Apart from producing new actions, the execution can have further side-effects that are of importance for the agent such as a change in the beliefbase or in the goalbase. These occurrences are captured and computed by a change determination module. This module evaluates affected conditions and, if one is triggered, new agenda entries may be produced and therefore inserted in the agenda component.

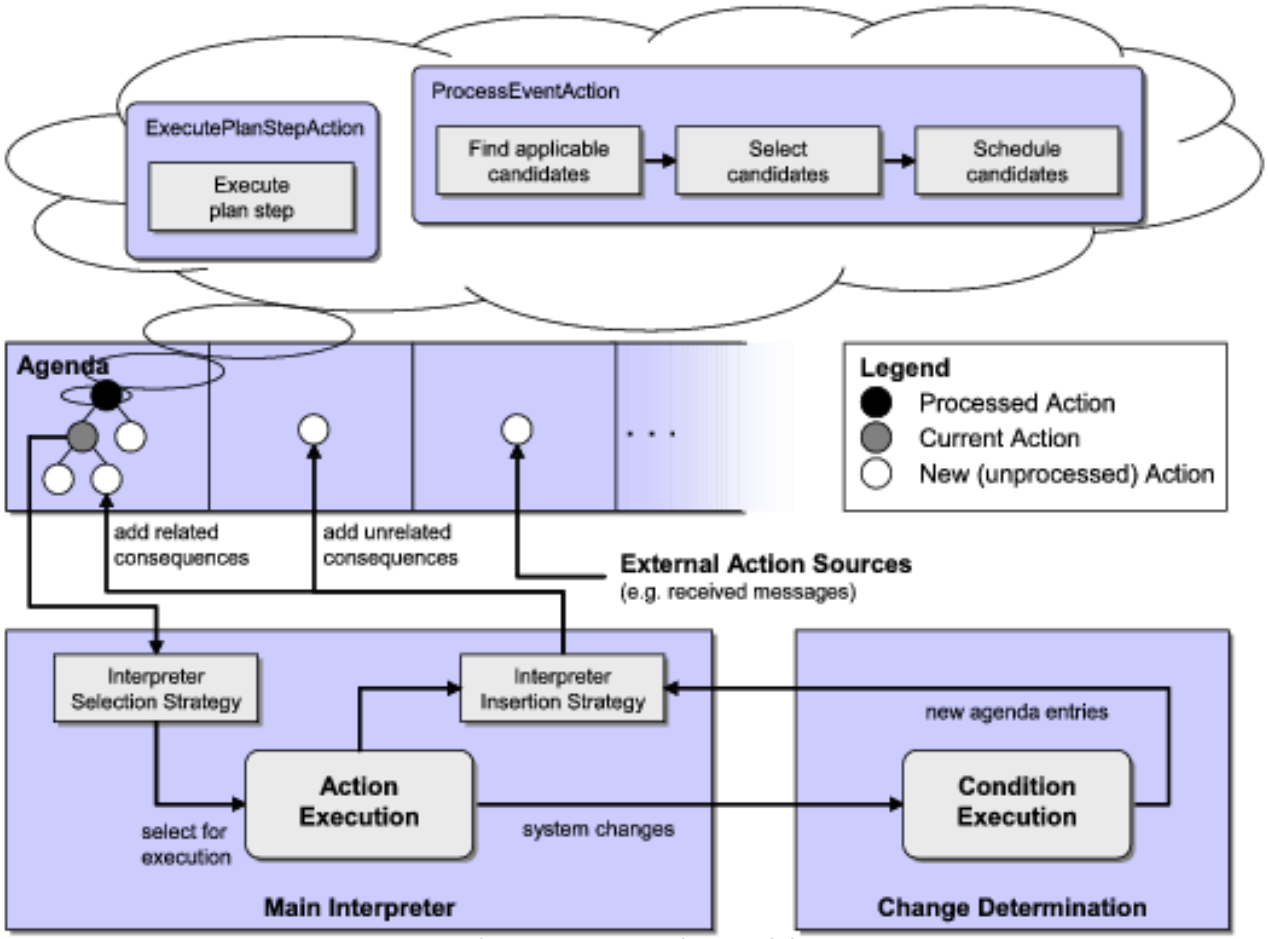


Figure A.3: Execution model.

As it can be seen, the actions contained in the agenda component are not application level actions but Meta-Level actions. Two typical MetaActions are displayed in the image: ExecutePlanStepAction and ProcessEventAction.

- ExecutePlanStepAction: executes one step of the plan and produces a new ExecutePlanStepAction if more plan steps are required.

- ProcessEventAction: encapsulates the plan finding process. Searches for applicable plans matching to an event or goal occurrence, selects candidates from the list and schedules them for execution by creating ExecutePlanStepActions for each candidate.

A.3 Agent Platforms

Jadex is a pure reasoning engine. This means that Jadex agents can potentially run on any middleware platform that fulfills some basic services concerning agent management and messaging. Currently, adapters for Jadex have been realized for the agent platform JADE [JADE] and for a Standalone platform.

A.3.1 JADE Platform

The JADE platform provides sophisticated implementations of all important FIPA specifications. By standing on top of the JADE platform Jadex achieves FIPA-compliance. Apart from this, using the JADE platform allows Jadex to support Ontology and Content Language in messages, agent migration between platforms and agent persistence among others. Because of all this, JADE agents and Jadex agents can coexist in the same Agent platform making JADE and Jadex interoperable.

A.3.2 Standalone Platform

The Standalone Platform is a fast and efficient execution environment for Jadex agents with a small memory footprint. The standalone platform is not a FIPA compliant platform but at least all messages are ACL encoded.

A.4 Tools

Jadex includes various tools for runtime and debugging activities as well as for development and documentation purposes [JTOOL]. All the tools are inserted as plugins in a main graphical interface, called the Jadex Control Center (JCC), which acts as a central access point. The most significant tools that can be accessed from the JCC are:

- Starter : administers the agents on the platform. It can be used to load, start and kill selected agents.
- Introspector: allows to observe the internal state of agents including their beliefs, goals and plans. It also provides a debugger to execute agents stepwise.
- Conversation Center: permits to compose ACL-compliant messages and send them to agents directly.
- DF Browser: the DF (Directory Facilitator) Browser administers the service registrations on the platform. It can be used to view and remove agent service descriptions.
- BDI Tracer: allows to visualize the internal processes of an agent at runtime and show causal dependencies among agent’s beliefs, goals and plans.

In addition, when using Jadex on top of JADE (i.e. when Jadex is run with the JADE adapter explained in Section A.3.1) it is also possible to use the JADE Remote Monitoring Agent (RMA). The RMA is a system agent that offers a GUI to manage agents, containing among others the Sniffer tool to monitor conversations between agents.

A.5 Agent Specification

Agents in Jadex can be specified by providing two sources of information:

- Agent Description File (ADF): it is an XML file containing the definition of all the BDI elements, that is, the beliefs, the goals and the plans of the agent.

- Java classes: they correspond to the implementation in Java of the agent's plans. These classes are referenced from the ADF, and contain the necessary actions to execute the plan. In a plan implementation, subgoals can be created and the belief base can be updated as a result of performing actions.

A.6 Summary

In this chapter we have analyzed Jadex, regarding the motivation to use it, the elements in the language, the operational semantics, the possible environments it can run on, the tools that it provides and the agent specification (illustrated by a simple example).

At the light of this analysis, we can conclude that:

- Jadex is totally object oriented, since all the BDI elements (beliefs, goals and plans) are represented as Java objects. In this sense, it differs from other engines like Jason or 2APL where, for instance, beliefs are represented as first-order predicates.

- For a programmer that is familiar with Java and XML, Jadex is quite easy to learn, since no other previous background is required. However, the difficulty is greater when one does not have this knowledge.

- An important richness of Jadex is the expressivity and dinamicity that the OQL syntax provides, making possible to perform queries at execution time on the belief base.

- A drawback of Jadex is the lack of a tool that facilitates the management of ADF files. Currently, the programmer has to deal directly with XML, either writing the file from scratch or starting from a previous work, which is not desirable.

Appendix B

Test Results

Apple – AAPL

AAPL-ANN-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3853	233	171	90.51%
pred. Up	41	5	5	9.8%
pred. Down	29	3	8	20%
class recall	98.22%	2.07%	4.35%	

AAPL-ANN-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1705	60	46	94.15%
pred. Up	14	2	1	11.76%
pred. Down	33	2	1	2.78%
class recall	97.32%	3.12%	2.08%	

AAPL-SVM-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3923	241	184	90.23%
pred. Up	0	0	0	0%
pred. Down	0	0	0	0%
class recall	100%	0%	0%	

AAPL-SVM-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1752	64	48	93.99%
pred. Up	0	0	0	0%
pred. Down	0	0	0	0%
class recall	100%	0%	0%	

AAPL-DTREE-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3509	207	145	90.88%
pred. Up	236	20	20	7.25%
pred. Down	178	14	19	9%
class recall	89.45%	8.3%	10.33%	

AAPL-DTREE-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1126	41	25	94.46%
pred. Up	186	10	12	4.81%
pred. Down	440	13	11	2.37%
class recall	64.27%	15.62%	22.92%	

AAPL-ANN-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	1767	458	257	71.19%
pred. Up	232	584	68	66.06%
pred. Down	176	45	762	77.52%
class recall	81.24%	53.73%	70.10%	

AAPL-ANN-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1	0	1	50.00%
pred. Up	421	11	10	2.49%
pred. Down	1331	53	37	2.60%
class recall	0.06%	17.19%	77.08%	

AAPL-SVM-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	2175	49	36	96.24%
pred. Up	0	1038	0	100.00%
pred. Down	0	0	1051	100.00%
class recall	100.00%	95.49%	96.69%	

AAPL-SVM-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1753	64	48	93.99%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

AAPL-DTREE-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	1730	161	105	86.67%
pred. Up	262	795	101	68.65%
pred. Down	183	131	881	73.72%
class recall	79.54%	73.14%	81.05%	

AAPL-DTREE-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	418	10	11	95.22%
pred. Up	796	33	23	3.87%
pred. Down	539	21	14	2.44%
class recall	23.84%	51.56%	29.17%	

Apple Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	71.58% +/- 13.92%	28.42% +/- 13.92%	68.36%	51.90% +/- 14.67%	0.470 +/- 0.119
ANN Test	2.63%	97.37%	31.44%	18.36%	0.984
Dtree Train	78.32% +/- 4.88%	21.68% +/- 4.88%	77.91%	45.62% +/- 9.97%	0.443 +/- 0.053
Dtree Test	24.93%	75.07%	34.86%	33.84%	0.864
SVM Train	98.04% +/- 2.82%	1.96% +/- 2.82%	97.39%	98.75%	0.101 +/- 0.097
SVM Test	93.99%	6.01%	33.33%	31.33%	0.245

Apple Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	88.92% +/- 6.69%	11.08% +/- 6.69%	33.88% +/- 1.43%	40.1%	0.315 +/- 0.135
ANN Test	91.63%	8.37%	34.18%	36.23%	0.291
Dtree Train	81.60% +/- 9.05%	18.40% +/- 9.05%	34.41% +/- 2.46%	34.60% +/- 2.25%	0.405 +/- 0.095
Dtree Test	61.53%	38.47%	34.27%	33.88%	0.611
SVM Train	90.23% +/- 5.70%	9.77% +/- 5.70%	33.33%	30.08%	0.298 +/- 0.094
SVM Test	93.99%	6.01%	33.33%	31.33%	0.245

AAPL-QL-TRAIN	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	2147	137	102	91.00%	0.658
pred. Up	743	52	50	6.15%	0.143
pred. Down	768	53	32	3.75%	0.1
class recall	61.53%	21.49%	17.39%		

AAPL-QL-TEST	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	1284	38	27	95.18%	0.768
pred. Up	250	17	11	6.11%	0.159
pred. Down	218	9	10	4.22%	0.116
class recall	73.29%	26.56%	20.83%		

Coca Cola – KO

KO-ANN-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	8254	30	48	99.06%
pred. Down	1	0	0	0%
pred. Up	2	0	1	33.33%
class recall	99.96%	0%	2.04%	

KO-ANN-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	3523	21	28	98.63%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

KO-SVM-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	8257	30	49	99.05%
pred. Down	0	0	0	0.00%
pred. Up	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

KO-SVM-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	3524	21	28	98.63%
pred. Down	0	0	0	0.00%
pred. Up	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

KO-DTREE-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	8176	30	48	99.06%
pred. Down	17	0	0	0.00%
pred. Up	64	0	1	1.54%
class recall	99.02%	0.00%	2.04%	

KO-DTREE-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	3433	19	23	98.79%
pred. Down	46	1	4	1.96%
pred. Up	45	1	1	2.13%
class recall	97.42%	4.76%	3.57%	

KO-ANN-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3678	848	695	70.45%
pred. Up	168	1235	86	82.94%
pred. Down	323	1	1303	80.09%
class recall	88.22%	59.26%	62.52%	

KO-ANN-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	2566	22	10	98.77%
pred. Up	936	6	11	0.63%
pred. Down	23	0	0	0.00%
class recall	72.79%	21.43%	0.00%	

KO-SVM-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	4169	1	1	99.95%
pred. Up	0	2083	0	100.00%
pred. Down	0	0	2083	100.00%
class recall	100.00%	99.95%	99.95%	

KO-SVM-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	3525	28	21	98.63%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

KO-DTREE-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	4062	1	1	99.95%
pred. Up	68	2083	0	96.84%
pred. Down	39	0	2083	98.16%
class recall	97.43%	99.95%	99.95%	

KO-DTREE-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	3524	28	21	98.63%
pred. Up	0	0	0	0.00%
pred. Down	1	0	0	0.00%
class recall	99.97%	0.00%	0.00%	

CocaCola Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	74.56% +/- 16.76%	25.44% +/- 16.76%	70.00%	77.82%	0.426 +/- 0.188
ANN Test	71.96%	28.04%	31.41%	33.13%	0.514
Dtree Train	98.69% +/- 1.39%	1.31% +/- 1.39%	99.11%	98.32%	0.084 +/- 0.077
Dtree Test	98.6%	1.4%	33.32%	32.88%	0.118
SVM Train	99.98% +/- 0.07%	0.02% +/- 0.07%	99.97%	99.98%	0.005 +/- 0.015
SVM Test	98.63%	1.37%	33.33%	32.88%	0.117

CocaCola Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	99.03% +/- 0.84%	0.97% +/- 0.84%	34%	44.13%	0.093 +/- 0.038
ANN Test	98.63%	1.37%	33.33%	32.88%	0.122
Dtree Train	98.09% +/- 1.39%	1.91% +/- 1.39%	33.69%	33.53%	0.124 +/- 0.040
Dtree Test	96.14%	3.86%	35.25%	34.29%	195
SVM Train	99.05% +/- 0.85%	0.95% +/- 0.85%	33.33%	33.02%	0.090 +/- 0.037
SVM Test	98.63%	1.37%	33.33%	32.88%	0.177

KO-QL-TRAIN	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	5644	32	24	99.02%	0.728
pred. Up	1323	10	1	0.75%	0.032
pred. Down	1296	7	5	0.38%	0.017
class recall	68.30%	20.41%	16.67%		

KO-QL-TEST	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	2528	19	14	98.71%	0.728
pred. Up	531	6	4	1.11%	0.033
pred. Down	465	3	3	0.64%	0.027
class recall	71.74%	4.59%	2.70%		

Microsoft – MSFT

MSFT-ANN-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3823	125	88	94.72%
pred. Up	22	3	4	10.34%
pred. Down	13	3	0	0%
class recall	99.09%	2.29%	0%	

MSFT-ANN-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1616	17	19	97.82%
pred. Up	66	10	7	12.05%
pred. Down	13	1	0	0%
class recall	95.34%	35.71%	0%	

MSFT-SVM-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3858	131	92	94.54%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

MSFT-SVM-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1695	28	26	96.91%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

MSFT-DTREE-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	3648	120	84	94.7%
pred. Up	108	7	6	5.79%
pred. Down	102	4	2	1.85%
class recall	94.56%	5.34%	2.17%	

MSFT-DTREE-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1615	25	25	97%
pred. Up	53	2	1	3.57%
pred. Down	27	1	0	0%
class recall	95.28%	7.14%	0%	

MSFT-ANN-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	1881	341	235	76.56%
pred. Up	94	658	18	85.45%
pred. Down	66	21	767	89.81%
class recall	92.16%	64.51%	75.20%	

MSFT-ANN-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	66	0	0	100.00%
pred. Up	1264	21	16	1.61%
pred. Down	366	7	10	2.61%
class recall	3.89%	75.00%	38.46%	

MSFT-SVM-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	2041	44	34	96.32%
pred. Up	0	976	0	100.00%
pred. Down	0	0	986	100.00%
class recall	100.00%	95.69%	96.67%	

MSFT-SVM-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1696	28	26	96.91%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

MSFT-DTREE-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	1812	46	18	96.59%
pred. Up	126	942	22	86.42%
pred. Down	103	32	980	87.89%
class recall	88.78%	92.35%	96.08%	

MSFT-DTREE-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	475	2	2	99.16%
pred. Up	847	5	9	0.58%
pred. Down	374	21	15	3.66%
class recall	28.01%	17.86%	57.69%	

Microsoft Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	81.01% +/- 13.21%	18.99% +/- 13.21%	77.29%	83.94%	0.379 +/- 0.125
ANN Test	5.54%	94.46%	39.12%	34.74%	0.967
Dtree Train	91.50% +/- 6.13%	8.50% +/- 6.13%	92.4%	90.3%	0.259 +/- 0.123
Dtree Test	28.29%	71.71%	34.52%	34.47%	0.845
SVM Train	98.09% +/- 3.26%	1.91% +/- 3.26%	97.45%	98.77%	0.080 +/- 0.112
SVM Test	96.91%	3.09%	33.33%	32.30%	0.176

Microsoft Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	93.75% +/- 5.05%	6.25% +/- 5.05%	33.77% +/- 1.24%	35.02%	0.231 +/- 0.087
ANN Test	92.97%	7.03%	43.68%	36.62%	0.24
Dtree Train	89.61% +/- 5.55%	10.39% +/- 5.55%	34.12% +/- 2.28%	33.76% +/- 3.04%	0.305 +/- 0.089
Dtree Test	92.45%	7.55%	34.14%	33.52%	0.270
SVM Train	94.53% +/- 4.35%	5.47% +/- 4.35%	33.33%	31.51%	0.219 +/- 0.082
SVM Test	96.91%	3.09%	33.33%	32.30%	0.176

MSFT-QL-TRAIN	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	2685	87	52	95.08%	0.735
pred. Up	604	23	15	3.58%	0.097
pred. Down	572	23	26	4.19%	0.131
class recall	69.54%	17.29%	27.96%		

MSFT-QL-TEST	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	1246	19	15	97.34%	0.773
pred. Up	236	4	7	1.62%	0.056
pred. Down	213	5	4	1.81%	0.061
class recall	73.51%	14.29%	15.39%		

Pfizer – PFE

PFE-ANN-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	4712	44	43	98.19%
pred. Up	17	2	0	10.53%
pred. Down	4	1	0	0%
class recall	99.56%	4.26%	0%	

PFE-ANN-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	2007	16	20	98.24%
pred. Up	14	2	0	12.5%
pred. Down	8	0	0	0%
class recall	98.92%	11.11%	0%	

PFE-SVM-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	4733	47	43	98.13%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

PFE-SVM-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	2029	18	20	98.16%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

PFE-DTREE-UNBAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	4667	45	42	98.17%
pred. Up	44	2	1	4.26%
pred. Down	22	0	0	0.00%
class recall	98.61%	4.26%	0.00%	

PFE-DTREE-UNBAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1807	14	11	98.64%
pred. Up	218	3	9	1.30%
pred. Down	4	1	0	0.00%
class recall	89.06%	16.67%	0.00%	

PFE-ANN-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	2350	24	100	94.99%
pred. Up	32	1178	1	97.27%
pred. Down	30	3	1104	97.10%
class recall	97.43%	97.76%	91.62%	

PFE-ANN-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	75	1	3	94.94%
pred. Up	1173	16	10	1.33%
pred. Down	782	1	7	0.89%
class recall	3.69%	88.89%	35.00%	

PFE-SVM-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	2412	16	16	98.69%
pred. Up	0	1189	0	100.00%
pred. Down	0	0	1189	100.00%
class recall	100.00%	98.67%	98.67%	

PFE-SVM-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	2030	18	20	98.16%
pred. Up	0	0	0	0.00%
pred. Down	0	0	0	0.00%
class recall	100.00%	0.00%	0.00%	

PFE-DTREE-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	2257	1	9	99.56%
pred. Up	60	1204	1	95.18%
pred. Down	95	0	1195	92.64%
class recall	93.57%	99.92%	99.17%	

PFE-DTREE-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	1674	15	19	98.01%
pred. Up	285	3	1	1.04%
pred. Down	71	0	0	0.00%
class recall	82.46%	16.67%	0.00%	

Pfizer Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	96.06% +/- 3.36%	3.94% +/- 3.36%	95.6%	96.45%	0.179 +/- 0.073
ANN Test	4.74%	95.26%	42.53%	32.39%	0.969
Dtree Train	96.56% +/- 3.44%	3.44% +/- 3.44%	97.55%	95.79%	0.142 +/- 0.116
Dtree Test	81.09%	18.91%	33.04%	33.02%	0.429
SVM Train	99.34% +/- 1.26%	0.66% +/- 1.26%	99.11%	99.56%	0.042 +/- 0.070
SVM Test	98.16%	1.84%	33.33%	32.72%	0.136

Pfizer Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	97.74% +/- 1.83%	2.26% +/- 1.83%	34.6%	36.24%	0.136 +/- 0.062
ANN Test	97.19%	2.81%	36.68%	36.91%	0.177
Dtree Train	96.81% +/- 2.34%	3.19% +/- 2.34%	34.29%	34.14%	0.167 +/- 0.062
Dtree Test	87.57%	12.43%	35.24%	33.31%	0.311
SVM Train	98.13% +/- 1.24%	1.87% +/- 1.24%	33.33%	32.71%	0.126 +/- 0.052
SVM Test	98.16%	1.84%	33.33%	32.72%	0.136

PFE-QL-TRAIN	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	3036	25	27	98.32%	0.689
pred. Up	859	15	8	1.70%	0.07
pred. Down	844	7	8	0.93%	0.039
class recall	64.06%	31.91%	18.60%		

PFE-QL-TEST	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	1462	12	16	98.12%	0.761
pred. Up	299	3	3	0.98%	0.039
pred. Down	268	3	1	0.37%	0.014
class recall	72.05%	16.67%	5.00%		

Yahoo! – YHOO

YHOO-ANN-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	1677	189	247	79.37%
pred. Down	43	10	10	15.87%
pred. Up	74	18	28	23.33%
class recall	93.48%	4.61%	9.82%	

YHOO-ANN-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	863	32	32	93.1%
pred. Down	22	2	1	8%
pred. Up	25	0	7	21.88%
class recall	94.84%	5.88%	17.5%	

YHOO-SVM-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	1794	217	285	78.14%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

YHOO-SVM-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	910	34	40	92.48%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

YHOO-DTREE-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	1395	140	207	80.08%
pred. Down	175	33	41	13.25%
pred. Up	224	44	37	12.13%
class recall	77.76%	15.21%	12.98%	

YHOO-DTREE-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	890	33	38	92.61%
pred. Down	4	0	0	0%
pred. Up	16	1	2	10.53%
class recall	97.8%	0%	5%	

YHOO-ANN-BAL-TRAIN	true Stay	true Down	true Up	class precisior
pred. Stay	942	194	213	69.83%
pred. Down	100	251	72	59.34%
pred. Up	107	129	289	55.05%
class recall	81.98%	43.73%	50.35%	

YHOO-ANN-BAL-TEST	true Stay	true Down	true Up	class precisior
pred. Stay	16	0	2	88.89%
pred. Down	530	16	13	2.86%
pred. Up	366	18	25	6.11%
class recall	1.75%	47.06%	62.50%	

YHOO-SVM-BAL-TRAIN	true Stay	true Down	true Up	class precisior
pred. Stay	1149	188	240	72.86%
pred. Down	0	386	0	100%
pred. Up	0	0	334	100%
class recall	100%	67.25%	58.19%	

YHOO-SVM-BAL-TEST	true Stay	true Down	true Up	class precisior
pred. Stay	912	34	40	92.49%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

YHOO-DTREE-BAL-TRAIN	true Stay	true Down	true Up	class precisior
pred. Stay	863	151	150	74.14%
pred. Down	141	268	131	49.63%
pred. Up	145	155	293	49.41%
class recall	75.11%	46.69%	51.05%	

YHOO-DTREE-BAL-TEST	true Stay	true Down	true Up	class precisior
pred. Stay	88	3	7	89.80%
pred. Down	192	7	5	3.43%
pred. Up	632	24	28	4.09%
class recall	9.65%	20.59%	70.00%	

Yahoo Balanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	64.52% +/- 17.12%	35.48% +/- 17.12%	58.69%	50.76% +/- 11.66%	0.539 +/- 0.142
ANN Test	5.78%	94.22%	37.10%	32.62%	0.965
Dtree Train	62.00% +/- 10.41%	38.00% +/- 10.41%	57.61%	45.56% +/- 9.81%	0.593 +/- 0.077
Dtree Test	12.47%	87.53%	33.41%	32.44%	0.935
SVM Train	81.37% +/- 14.09%	18.63% +/- 14.09%	75.15%	90.95%	0.397 +/- 0.169
SVM Test	92.49%	7.51%	33.33%	30.83%	0.274

Yahoo Unbalanced	Accuracy	Class. Error	Mean Recall	Mean Prec.	RMSE
ANN Train	74.69% +/- 13.83%	25.31% +/- 13.83%	35.97%	39.52%	0.463 +/- 0.150
ANN Test	88.62%	11.38%	39.41%	40.99%	0.324
Dtree Train	63.81% +/- 15.26%	36.19% +/- 15.26%	35.32%	32.85% +/- 2.92%	0.559 +/- 0.157
Dtree Test	90.65%	9.35%	34.27%	34.38%	0.301
SVM Train	78.14% +/- 12.13%	21.86% +/- 12.13%	33.33%	26.05%	0.430 +/- 0.185
SVM Test	92.48%	7.52%	33.33%	30.83%	0.274

YHOO-QL-TRAIN	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	1014	142	116	79.72%	0.599
pred. Up	437	73	61	12.78%	0.213
pred. Down	348	71	40	8.71%	0.151
class recall	56.36%	25.52%	18.43%		

YHOO-QL-TEST	true Stay	true Up	true Down	class precision	f2 score
pred. Stay	589	23	23	92.76%	0.689
pred. Up	150	11	10	6.43%	0.166
pred. Down	171	6	1	0.56%	0.016
class recall	64.72%	27.50%	2.94%		

3% Configuration

YHOO-ANN-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	1032	311	326	61.83%
pred. Down	153	60	88	19.93%
pred. Up	162	67	98	29.97%
class recall	76.61%	13.7%	19.14%	

YHOO-ANN-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	662	66	71	82.85%
pred. Down	84	11	8	10.68%
pred. Up	51	15	17	20.48%
class recall	83.06%	11.96%	17.71%	

YHOO-SVM-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	1347	438	512	58.64%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

YHOO-SVM-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	797	92	96	80.91%
pred. Down	0	0	0	0%
pred. Up	0	0	0	0%
class recall	100%	0%	0%	

YHOO-DTREE-UNBAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	880	227	300	62.54%
pred. Down	204	109	114	25.53%
pred. Up	263	102	98	21.17%
class recall	65.33%	24.89%	19.14%	

YHOO-DTREE-UNBAL-TEST	true Stay	true Down	true Up	class precision
pred. Stay	604	67	70	81.51%
pred. Down	69	12	14	12.63%
pred. Up	124	13	12	8.05%
class recall	75.78%	13.04%	12.5%	

YHOO-ANN-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	1072	162	196	74.97%
pred. Up	46	263	184	53.35%
pred. Down	35	150	195	51.32%
class recall	92.97%	45.74%	33.91%	

YHOO-ANN-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	11	3	2	68.75%
pred. Up	526	58	56	9.06%
pred. Down	263	35	34	10.24%
class recall	1.38%	60.42%	36.96%	

YHOO-SVM-BAL-TRAIN	true Stay	true Up	true Down	class precision
pred. Stay	1153	486	434	55.62%
pred. Up	0	89	0	100%
pred. Down	0	0	141	100%
class recall	100%	15.48%	24.52%	

YHOO-SVM-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	800	96	92	80.97%
pred. Up	0	0	0	0%
pred. Down	0	0	0	0%
class recall	100%	0%	0%	

YHOO-DTREE-BAL-TRAIN	true Stay	true Down	true Up	class precision
pred. Stay	863	151	150	74.14%
pred. Down	141	268	131	49.63%
pred. Up	145	155	293	49.41%
class recall	75.11%	46.69%	51.05%	

YHOO-DTREE-BAL-TEST	true Stay	true Up	true Down	class precision
pred. Stay	76	14	8	77.55%
pred. Up	557	58	71	8.45%
pred. Down	167	20	17	8.33%
class recall	9.5%	63.04%	17.71%	

YHOO	Accuracy	Class.Error	Mean Recall	Mean Precision	RMSE
ANN-BAL-TRAIN	66.42% +/- 22.02%	33.58% +/- 22.02%	60.87% +/- 11.63%	62.52% +/- 8.76%	0.511 +/- 0.179
ANN-BAL-TEST	10.43%	89.57%	32.92%	29.35%	0.93
SVM-BAL-TRAIN	60.05% +/- 16.86%	39.95% +/- 16.86%	44.04% +/- 11.92%	85.21%	0.616 +/- 0.141
SVM-BAL-TRAIN	80.97%	19.03%	33.33%	29.99%	0.436
DTREE-BAL-TRAIN	62.00% +/- 10.41%	38.00% +/- 10.41%	57.61%	45.56% +/- 9.81%	0.593 +/- 0.077
DTREE-BAL-TEST	15.28%	84.72%	30.08%	31.45%	0.918 +/- 0.000
ANN-UNBAL-TRAIN	51.81% +/- 15.71%	48.19% +/- 15.71%	34.25% +/- 3.27%	35.81% +/- 5.46%	0.640 +/- 0.114
ANN-UNBAL-TEST	70.05%	29.95%	37.58%	38.01%	0.514
SVM-UNBAL-TRAIN	58.64% +/- 16.33%	41.36% +/- 16.33%	33.33%	19.55%	0.627 +/- 0.145
SVM-UNBAL-TEST	80.91%	19.09%	33.33%	26.97%	0.437
DTREE-UNBAL-TRAIN	47.32% +/- 10.48%	52.68% +/- 10.48%	33.86% +/- 3.05%	34.36% +/- 2.85%	0.701 +/- 0.075
DTREE-UNBAL-TEST	63.76%	36.24%	33.78%	34.07%	0.588

BROMAS Executions

	fscore_stay	fscore_up	fscore_dow	Accuracy
YHOO-RL	0.664	0.134	0.075	0.588
YHOO-SVM	0.984	0	0	0.925
YHOO-DTREE	0.984	0	0	0.925
YHOO-ANN	0.946	0.096	0	0.883
YHOO-TRADE	0.982	0	0	0.923
MSFT-RL	0.71	0.027	0.023	0.647
MSFT-SVM	0.994	0	0	0.969
MSFT-DTREE	0.841	0	0.012	0.79
MSFT-ANN	0.992	0	0	0.967
MSFT-TRADE	0.969	0	0	0.939
KO-RL	0.639	0.169	0.063	0.565
KO-SVM	0.984	0	0	0.925
KO-DTREE	0.984	0	0	0.925
KO-ANN	0.946	0.096	0	0.883
KO-TRADER	0.981	0	0	0.922
PFE-RL	0.702	0.115	0.14	0.628
PFE-SVM	0.984	0	0	0.925
PFE-DTREE	0.984	0	0	0.925
PFE-ANN	0.946	0.096	0	0.883
PFE-TRADER	0.984	0	0	0.925
AAPL-RL	0.489	0.113	0.087	0.427
AAPL-DTREE	0.986	0	0	0.939
AAPL-SVM	0.987	0	0	0.94
AAPL-ANN	0.972	0	0	0.922
AAPL-TRADE	0.983	0	0	0.935

Apple – AAPL

AAPL-TRADER	true Stay	true Down	true Up	class Precision
pred. Stay	1743	48	63	0.94
pred. Down	10	0	0	0
pred. Up	0	0	0	0
class Recall	0.994	0	0	

AAPL-ANN	true Stay	true Down	true Up	class Precision
pred. Stay	1718	48	63	0.939
pred. Down	26	0	1	0
pred. Up	8	0	0	0
class Recall	0.981	0	0	

AAPL-SVM	true Stay	true Down	true Up	class Precision
pred. Stay	1752	48	64	0.94
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

AAPL-DTREE	true Stay	true Down	true Up	class Precision
pred. Stay	1750	48	64	0.94
pred. Down	2	0	0	0
pred. Up	0	0	0	0
class Recall	0.999	0	0	

AAPL-RL	true Stay	true Down	true Up	class Precision
pred. Stay	766	21	28	0.94
pred. Down	690	16	22	0.022
pred. Up	297	11	13	0.04
class Recall	0.437	0.333	0.206	

Coca Cola – KO

KO-TRADER	true Stay	true Down	true Up	class Precision
pred. Stay	908	34	40	0.925
pred. Down	3	0	0	0
pred. Up	0	0	0	0
class Recall	0.997	0	0	

KO-ANN	true Stay	true Down	true Up	class Precision
pred. Stay	865	31	36	0.928
pred. Down	3	0	0	0
pred. Up	42	3	4	0.082
class Recall	0.951	0	0.1	

KO-SVM	true Stay	true Down	true Up	class Precision
pred. Stay	910	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

KO-DTREE	true Stay	true Down	true Up	class Precision
pred. Stay	910	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

KO-RL	true Stay	true Down	true Up	class Precision
pred. Stay	540	22	17	0.933
pred. Down	168	4	10	0.022
pred. Up	203	8	13	0.058
class Recall	0.593	0.118	0.325	

Microsoft – MSFT

MSFT-TRADER	true Stay	true Down	true Up	class Precision
pred. Stay	1643	26	27	0.969
pred. Down	53	0	1	0
pred. Up	0	0	0	0
class Recall	0.969	0	0	

MSFT-ANN	true Stay	true Down	true Up	class Precision
pred. Stay	1691	26	28	0.969
pred. Down	4	0	0	0
pred. Up	0	0	0	0
class Recall	0.998	0	0	

MSFT-SVM	true Stay	true Down	true Up	class Precision
pred. Stay	1695	26	28	0.969
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

MSFT-DTREE	true Stay	true Down	true Up	class Precision
pred. Stay	1381	25	27	0.964
pred. Down	314	1	1	0.003
pred. Up	0	0	0	0
class Recall	0.815	0.038	0	

MSFT-RL	true Stay	true Down	true Up	class Precision
pred. Stay	1128	18	16	0.971
pred. Down	321	2	10	0.006
pred. Up	247	6	2	0.008
class Recall	0.665	0.077	0.071	

Pfizer – PFE

PFE-TRADER	true Stay	true Down	true Up	class Precision
pred. Stay	911	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

PFE-ANN	true Stay	true Down	true Up	class Precision
pred. Stay	865	31	36	0.928
pred. Down	3	0	0	0
pred. Up	42	3	4	0.082
class Recall	0.951	0	0.1	

PFE-SVM	true Stay	true Down	true Up	class Precision
pred. Stay	910	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

PFE-DTREE	true Stay	true Down	true Up	class Precision
pred. Stay	910	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

PFE-RL	true Stay	true Down	true Up	class Precision
pred. Stay	603	20	26	0.929
pred. Down	135	8	6	0.054
pred. Up	173	6	8	0.043
class Recall	0.662	0.235	0.2	

Yahoo! – YHOO

YHOO-TRADER	true Stay	true Down	true Up	class Precision
pred. Stay	909	34	40	0.925
pred. Down	1	0	0	0
pred. Up	1	0	0	0
class Recall	0.998	0	0	

YHOO-ANN	true Stay	true Down	true Up	class Precision
pred. Stay	865	31	36	0.928
pred. Down	3	0	0	0
pred. Up	42	3	4	0.082
class Recall	0.951	0	0.1	

YHOO-SVM	true Stay	true Down	true Up	class Precision
pred. Stay	910	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

YHOO-DTREE	true Stay	true Down	true Up	class Precision
pred. Stay	910	34	40	0.925
pred. Down	0	0	0	0
pred. Up	0	0	0	0
class Recall	1	0	0	

YHOO-RL	true Stay	true Down	true Up	class Precision
pred. Stay	565	21	26	0.923
pred. Down	186	5	5	0.051
pred. Up	160	8	9	0.026
class Recall	0.62	0.147	0.225	

Bibliography

Chapter 2

[Heredia] Entorn de simulació de borsa electrònica. PFC, FIB 2005

[BMadrid] Bolsa de Madrid http://www.bolsamadrid.es

[BBcn] Borsa de Barcelona http://www.borsabcn.es

[BFrank] Börse Frankfurt http://www.boerse-frankfurt.de

[Investopedia] Investopedia http://www.investopedia.com

[Investorguide] Investorguide http://www.investorguide.com

[Wiki] Stock Market at Wikipedia http://en.wikipedia.org/wiki/Stock_market

Chapter 4

[Vellido99] A Vellido, PJG Lisboa, J Vaughan. Neural networks in business: a survey of applications (1992-1998). Expert Systems with Applications, 1999

[Trippi92] RR Trippi, E Turban. Neural networks in finance and investing.McGraw-Hill, Inc, 1992

[Skabar02] A Skabar, I Cloete. Neural networks, financial trading and the efficient markets hypothesis. Proceedings of the twenty-fifth Australasian conference on Computer science - Volume 4, 2002

[Kim03] K Kim. Financial time series forecasting using Support Vector Machines. Neurocomputing, 2003

[Tay01] FEH Tay, L Cao. Application of support vector machines in financial time series forecasting. Omega, 2001

[Lee02] JW Lee, O Jangmin. A multi-agent Q-learning framework for optimizing stock trading systems. Lecture notes in computer science, Springer 2002

[Nevmyvaka] Y Nevmyvaka, Y Feng, M Kearns. Reinforcement learning for optimized trade execution. Proceedings of the 23rd international conference on Machine learning, 2006

[Duerson] S Duerson, F Saleem, V Kovalev, AH Malik. Reinforcement Learning in Online Stock Trading Systems

[Moody] J Moody, L Wu, Y Liao, M Saffell. Performance Functions and Reinforcement Learning for Trading Systems and Portfolios. Journal of forecasting, 1998.

Chapter 5

[Green97] Green et al. Software Agents: a review. 1997.

[RussellNorvig] SJ Russell, P Norvig. Artificial intelligence: A modern approach. Prentice Hall, 2003

[Pujol06] J Pujol. Structure in Artificial Societies. Ph.D Thesis, LSI 2006

[Busetta99] P Busetta, R Ronnquist, A Hodgson, A Lucas. Jack Intelligent agents – components for intelligent agents in Java. AgentLink News Letter, 1999

[Gambetta90] D Gambetta. Can we trust trust? Oxford, 1990

[Vazquez] J Vazquez. Slides from Multi Agent Systems course. FIB, 2006

[Bishop] CM Bishop. Pattern Recognition and Machine Learning. Springer, 2006

[Mitchell] TM Mitchell. Machine Learning. McGraw-Hill 1997

[Sutton] RS Sutton, AG Barto. Reinforcement Learning. Journal of Cognitive Neuroscience, MIT Press, 1999

[Weiss] G Weiss. Multi Agent Systems:A Modern Approach to Distributed Modern Approach to Artificial Intelligence. MIT Press, 1999

[Wooldridge] M Wooldridge. An Introduction to Multi Agent Systems. Wiley, 2002.

Chapter 6

[Bellifemine] FL Bellifemine. G Caire, D Greenwood. Developing Multi Agent Systems with JADE. Wiley, 2006

[Jadex] http://jadex.informatik.uni-hamburg.de/

[RapidMiner] http://rapid-i.com

[Processing] http://www.processing.org

[Fry] B Fry. Visualizing Data. O'Reilly, 2008

[Flanagan] D Flanagan, Y Matsumoto. The Ruby Programming Language. O'Reilly, 2008

[Vazquez] J Vazquez. Slides from Multi Agent Systems course. FIB, 2006

[Padgham02] L Padgham, M Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, 2002.

[Padgham03] L Padgham, M Winikoff. Prometheus: A methodology for developing intelligent agents. Lecture Notes in Computer Science, 2003

Chapter 7

[Gruber93] TR Gruber. A Translation Approach to Portable Ontology Specification. Knowledge Acquisition 5, 1993

[Gruber95] TR Gruber. Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human Computer Studies, 1995

[Gruber08] TR Gruber. Ontology, to appear in the Encyclopedia of Database Systems. Springer-Verlag, 2008

[Uschold] M Uschold, M Gruninger. Ontologies: Principles, methods and applications. Knowledge engineering review, 1996

[Noy] NF Noy, DL McGuinness. Ontology Development 101: A guide for creating your first ontology. Stanford, 2001

[protégé] http://protege.stanford.edu/

Machine Learning from imbalanced data sets 101

The effect of class distribution on classifier learning

[StockCharts] http://www.stockcharts.com

[Thomas] D Thomas, DH Hansson. Agile Web Development with Rails. Pragmatic Programmers, 2006

[GAPI] Google Visualization API http://code.google.com/apis/visualization/

Chapter 8

[Hsu] CW Hsu, CC Chang, CJ Lin. Libsvm: A practical guide to support vector classification. 2009