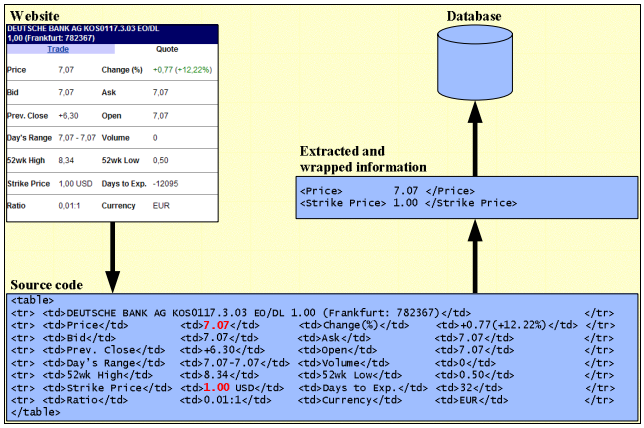
**IWI Discussion Paper Series**

**# 4 (May 20, 2003)**[1]

ISSN 1612-3646

# Automatic Extraction of Derivative Market Prices from Webpages using a Software Agent[2]

**Patrick Bartels**[3] and **Michael H. Breitner**[4]




---

[1] Copies or a PDF-file are available upon request: Institut für Wirtschaftsinformatik, Universität Hannover, Königsworther Platz 1, 30167 Hannover (www.iwi.uni-hannover.de).

[2] This paper summarizes first results of a work in progress. A final version of this paper will be submitted to "The Journal of Finance" in summer 2003.

[3] Research fellow and lecturer (bartels@iwi.uni-hannover.de).

[4] Full Professor for Information Systems Research/Business Administration (breitner@iwi.uni-hannover.de).

## Table of contents

# Automatic Extraction of Derivative Market Prices from Web-pages using a Software Agent

Patrick Bartels[1] and Michael H. Breitner[2]

Universität Hannover, Institut für Wirtschaftsinformatik, Königsworther Platz 1,

D-30167 Hannover, Germany

Some current research of derivative pricing is dedicated to artificial neural networks to generate market prices (see Breitner (2000 and 2001)) instead of analytical prices developed by Black, Scholes and Merton (1973) or Cox, Ross and Rubinstein (1979). Needed data are usually taken from commercial finance databases. This paper presents the software agent PISA[3] extracting quotes from webpages to generate cost free quote databases. Such databases provide time series for the training of neural networks. Extrapolating series with neural networks enables all kinds of forecasting. Interpolating data obtained enables, e. g., a comparison of derivative prices from different issuers and a synthesis of market price functions. This paper presents a comparison of selected programming languages to find the most suitable for the given tasks. The components of PISA are described in detail. The paper closes with examples for the extraction process.

**Keywords:** Software agent, market prices, derivatives, artificial neural networks.

## I. Introduction

Increasing importance of foreign currency transactions requires an appropriate hedging against possible risks. This gets more and more important since the volatility of the five biggest currencies is still increasing. The uncertainty about exchange rates makes handling of future contracts difficult. Using options and other derivatives enables hedging against upcoming risks by redistributing them to other agents. Therefore both sides need a reliable derivative pricing model. Today's pricing models mostly base on the Black/Scholes/Merton-model or the Cox/Ross/Rubinstein-approach. These models depend on some unrealistic assumptions, e. g. regarding true-market-conditions. Alternative current research focuses on using artificial neural networks to estimate market prices instead of analytical prices. This approach needs a sufficient dataset for training the neural networks. The dataset is usually taken from commercial databases which must be paid. Here, an alternative approach to get this input dataset using software agents extracting the information from the Internet for free is presented. Beside for

---

1 Research fellow and lecturer, email: bartels@iwi.uni-hannover.de, phone: ++49 511 762-4979, fax: ++49 511 762-4013.

2 Full professor for Information Systems Research and Business Administration, email: breitner@iwi.uni-hannover.de, phone: ++49 511 762-4901, fax: ++49 511 762-4013.

3 PISA = Partially Intelligent Software Agent

market prices neural networks can extrapolate the extracted time series, i. e. neural networks can forecast future values.
Not every programming language is comfortable for realizing such agents. Several selected languages are compared to find the most capable one. Each one is considered because of its individual attributes.
The presented software agent uses simultaneously executed threads to request and receive specified websites which contain the needed information. This information is provided on several not password protected Internet websites. In order to assure correct data each derivative's quote is extracted redundant from several websites. These websites usually are written in Hypertext Markup Language (HTML). The website's source code is received and processed to extract this information by exploiting the fact that the needed information pieces are each contained in a specific HTML-Tag. As HTML is developed for human interaction and not for machine transaction this requires a new approach to identify the wanted information reliably within the source code. This paper shows how information pieces can be identified reliably. A reliable and powerful tool with which a gratis dataset for training neural networks can be collected is build. The tool has been tested successfully by extracting values from over 180 webpages at the same time.

# II. Derivative market prices

## A. Neural networks in finance research

Neural networks are a very popular technique in financial market research. Many publications deal with possible applications of neural networks in financial markets, see Azoff (1996), Refenes (1996), Vemuri and Rogers (1994) and Zirilli (1997). Artificial neural networks are information-processing systems inspired by the way the densely interconnected, parallel structure of the mammalian brain processes information. Artificial neural networks are mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. That is why neural networks sometimes are called learning networks. Key element of the artificial neural network system is the novel structure of the information processing system. It is composed of an, eventually large, number of highly interconnected processing elements. These elements are analogous to neurons and tied together with weighted connections analogous to synapses. For further information about neural network's functionality, see Anderson (1997), Gallant (1995), Hecht-Nielsen (1994) and Hertz, Krogh and Palmer (1996).
Using neural networks mathematically means optimization. The input data are processed in a special way by the net. The network returns a mathematical function

$$f_{app}(x\,;\,p) \tag{1}$$

as result. This function depends on a vector $x$ that contains the input data and a vector $p$ that contains the arbitrary network parameters. The returned function approximates a correlation of the input variables. With this function either future values can be extrapolated or missing values within a discrete series can be interpolated. To estimate the function's quality approximated and desired output data are compared. As reference either a dataset with validated data or a known reference function

$$f_{ref}(x) \tag{2}$$

is used. The calculated error

$$\varepsilon(f_{app}(x;p);f_{ref}(x)) = f_{ref}(x) - f_{app}(x;p) \tag{3}$$

should be as small as possible after the neural network training. In order to minimize $\varepsilon$, the network tries other input/output-relations and compares the results again. If $\varepsilon$ is less than before the new function is accepted as the currently best one. Otherwise the function is discarded and another one is tried. The described process is a kind of a learning process. For details regarding the training process, see Breitner and Bartelsen (1999), Breitner (2001) and White, Gallant, Stinchcombe and Wooldridge (1992).

The described learning process is utilized for different kinds of applications. Recent financial market research concentrates on two common application ranges. Forecasting time series using neural networks has been discussed for many years now. The usual aim is to generate a mathematical function that predicts future values of the underlying time series. For the necessary training neural networks usually need a large dataset with a dense time series, e. g. derivative prices or interest rates. The network generated mathematical functions usually are validated using their market prices as reference. The needed data usually is taken from commercial databases. Price forecast is an example of extrapolating time series with neural networks. Second common application range is pricing derivatives. Today's analytical methods use several estimated variables. The derivative prices of different issuers usually are not equal. In cases where highly accurate prices are needed, neural networks can be used to generate market prices instead of analytical prices. Informative solutions are made by Hutchinson, Lo and Poggio (1994), Breitner (2001), Breitner (1997) and Breitner and Ambrosius (1999). The outcome of the network training process is a function that interpolates input prices. Changing the input variables market prices are generated for each variable combination without any estimation. As input data time price series of historical prices are needed. The series have to be as dense as possible and must not have large data gaps.

For optimizing the output data both forecasting and market price generating applications need validated data as a reference. For functional correctness the input data have to be correct and the time series must be dense. Those data usually is bought in databases. Financial service providers, e. g. Bloomberg[4] and Reuters[5], buy the data directly from stock exchanges and resell them. Such databases usually are very expensive. An alternative approach is to get that information from the Internet. Many websites publish current or time delayed option and derivative prices cost free. E. g., most German warrant issuers publish their latest prices on their own websites. Further sources are financial services company's websites. Most websites do not offer real-time prices but this is not necessary to get a dense price/time series. Neural networks are trained with historical series. The needed pieces of information can be automatically collected using autonomously working software. This program, called software agent, stores the data in a database. Most gratis accessible websites do not offer all needed data. Using them for information gathering complemental information from other websites can be extracted and merged.

---

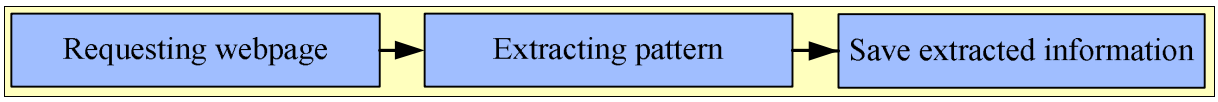
4 Bloomberg web address: http://www.bloomberg.com

5 Reuters web address: http://www.reuters.com

**Figure 1**

**Schematic extraction process**

*The figure shows the schematic extraction process. First the webpage that contains the wanted information is requested at a webserver. Each webpage has a unique web address, called Uniform Resource Locator (URL). The webserver sends the requested file to the agent. The received source code is processed. To identify and extract the wanted information a reliable method has to be realized that can handle all most common sources of errors. The extracted information is saved. Depending on further application it is stored in text files or databases.*




Visitors of financial websites can only see the currently published price. It is not possible to check the prices' up-to-dateness or its correctness. The user usually has no access to the underlying data and he is limited to the given functions. In addition to the reduced functionality the user does not know anything about the webpage's reliability.

Beside neural network training and evaluating website reliability a program that continuously collects data from webpages can be used to compare prices of derivatives from different issuers. Comparing the extracted issuer's prices enables to differ cheap and expensive derivatives.

Primary aim of this paper is to show a possibility to extract data from redundant websites automatically and to build a reliable database. This database will provide the neural network training data. With the presented approach it is possible to create any kind of time series from web published information. Here, data for a neural network input dataset to generate derivative market prices are collected. Basically all derivative kinds can be extracted. To generate a dense time series such an extraction approach is inappropriate for thinly traded derivatives or newly created derivatives. Here, German options are chosen as example. To achieve the mentioned goals, the *partially intelligent software agent* PISA is developed. The agent enables achieving both aims, extracting several stock, option or any other derivative prices efficiently and evaluating the offering websites. The program loads specified webpages, extracts the demanded data and stores them on computers in formats that can be easily further processed, see Figure 1. For example files formatted with the eXtensible Markup Language (XML) can be used which can be easily imported into databases, see Figure 2 for the complete process. The needed pieces of information are derived from the option price influencing variables described in the next subsection.

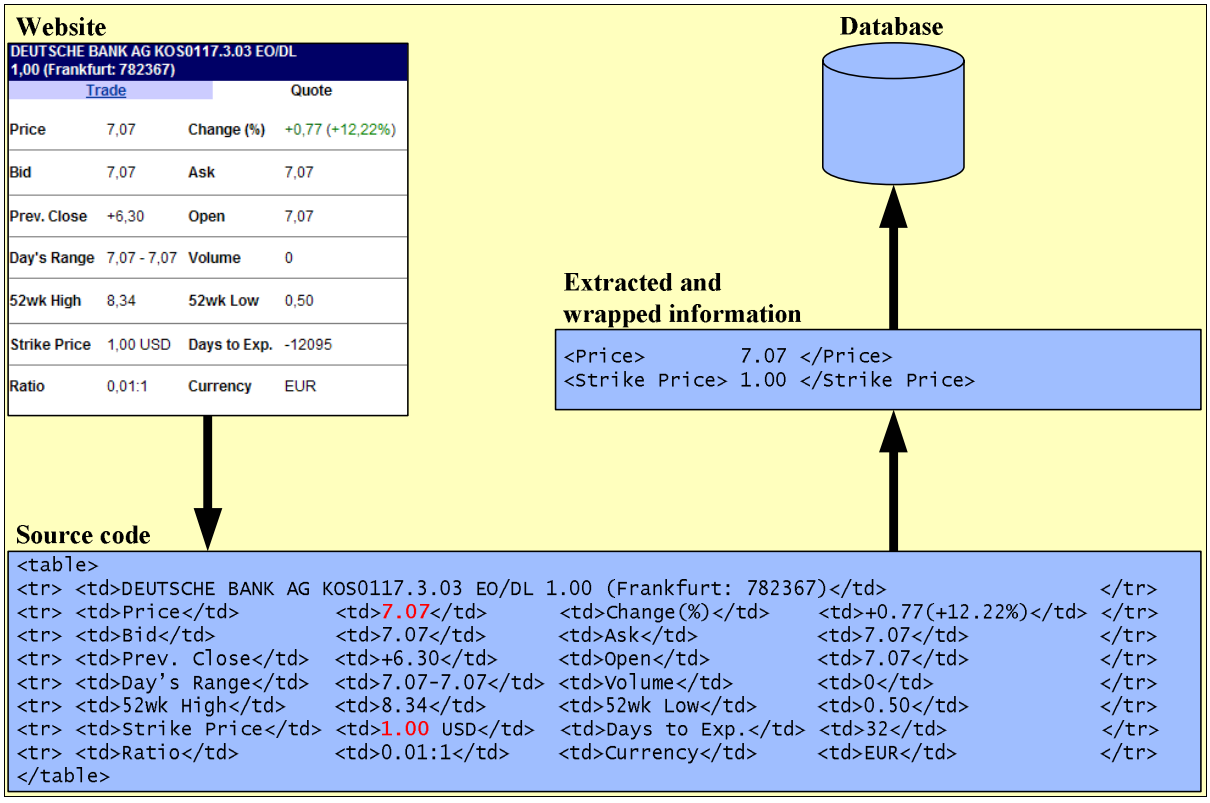
## B. Derivative market prices

Options are traded at an organized exchange since 1973. Since then there has been a growth in option markets and today they are traded all over the world and around the clock by banks and other financial institutions. For details regarding derivatives, especially options, see Hull (1999). Since the beginning Fischer Black, Myron Scholes and Robert Merton were looking for methods to price stock options. They developed a model that is still influencing traders who both price and hedge options. The importance of the so called Black/Scholes-model (Black and Scholes (1973) and Merton (1996)) was recognized when Black and Scholes were awarded the Nobel Prize in 1997. Since the late seventies three other scientists, John Ross,

**Figure 2**

**Information extraction process**

*The needed information usually is published in websites with a table structure like shown below. The way the internet browser displays information is usually described with the Hypertext Markup Language (HTML). HTML is a plain text language that uses so called tags for identifying an element. Most tags consist of a beginning tag and an ending tag. These two tags bracket the displayed information. The name of the used HTML tag just describes the way the content is displayed but not its context. The HTML source code can be processed with special methods to extract the wanted information. The extracted information can be wrapped in special tags that describe their content's context. Therefore usually the eXtensible Markup Language is used. Such XML code comfortably can be stored in a database and is highly platform independent.*




Stephen Ross and Mark Rubinstein, started research on option pricing (Cox and Ross (1976)). In contradiction to Black, Scholes and Merton they used a numerical method for pricing more general derivatives. The so called Cox/Rubinstein-model is also widely spread.

Common to both models is that the correct option price is investigated by an option-underlying portfolio with risk-free profit. The theoretically fair option price $p_{BS}$ (Black/Scholes) depends on

- underlying price $s$,
- strike price $b$,
- time to expiration $r$,
- risk-free interest rate $ir$ to expiration and
- future volatility $\sigma_S$ of the underlying price measured by its annualized standard deviation of percentage change in daily price.

The model of Cox and Rubinstein depends on *s*, *b*, *r* and *ir*, too. In "contradiction" to the Black/Scholes-model the future volatility is represented by the likelihood of rising and falling of the underlying price *s* in each time step.

The two mentioned analytic pricing models both base on two problematic assumptions:

- First the markets are assumed to be efficient so that a prediction of the direction of the market or an individual underlying is not possible.
- Second the future volatility $\sigma_s$ of the underlying price is assumed to be accurately estimated and is a priori known to seller and buyer of an option.

As a result of estimating the volatility in both models, $\sigma_{s,\,i}$ oscillates erratically and neither the Black/Scholes model nor the Cox/Ross/Rubinstein model captures option market conditions. In particular the every important option price sensitivities (option Greeks)

$$\Delta_* := \frac{\partial}{\partial s} p_*,\ \Gamma_* := \frac{\partial}{\partial s^2} p_*,\ \Theta_* := \frac{\partial}{\partial r} p_* \text{ and } \Omega_* := \frac{s}{p_*}\Delta_* \tag{4}$$

usually are inaccurate. These problems do not appear when instead of an analytical model market prices are used. The generation of market prices depends on historical and actual prices of independent options. For details of the option pricing process with neural networks, see Breitner (1999) and Hutchinson (1994). In contrast to the theoretical pricing models high accurate neural networks can learn true market pricing of options and warrants. Like the theoretical option price

$$p_{BS}(s, b, r, ir, \sigma_s) \tag{5}$$

or

$$p_{CRR}(s, b, r, ir, \sigma_s) \tag{6}$$

the market price

$$p_R(s, b, r, t) \tag{7}$$

depends on the permanently available underlying price *s*, strike price *b* and time to expiration *r*. But instead of *ir* and the artificially estimated $\sigma_s$ the permanently available trading day *t* is used as direct input for the pricing model is used (Breitner (1999)). From these variables the pieces of information to extract are derived.

## C. Underlying data

To generate market derivative prices it is necessary to extract specified information from the websites. The influencing variables for each price/time combination are mainly:

- Derivative price that is published on the webpage including bid- and ask-price if available, and
- related date and time of the price.

The price related date and time are extracted because in most cases the published prices are not real time prices. The extraction's point of time can not be used instead. The time to expiration is calculated from the extracted time posterior and can be expressed in minutes to expiration.

All other derivative attributes like expiration date or strike price are known and are not time-dependent. They can be a priori or a posteriori given by the user. Extracting this static information makes it easier to handle the extraction process. Eventually existing errors in linked webpages are noticed and identified. A common error is a wrong or incorrectly allocated option's security identification number. The agent automatically extracts static information as well as dynamic information. The neural network does not need this information for generating market prices but to identify a certain price time combination it is reasonable to extract them as well. Since the webpages of a website normally are identically formatted the extraction task configuration is equal as well. User's handling of the agent, especially the configuration, is much easier if only the web address (URL = uniform resource locator) has to be changed for each derivative without specifying any static information. An URL is an explicit and unique address of a webpage.
The following static information pieces are extracted:

- Security identification number,
- expiration date and
- strike price.

Here, German options are extracted. The in Germany used German security identification number (Wertpapierkennnummer = WKN) has 6-digits. Instead other international standards can be used as well. The related pattern just has to be modified.
Usually training data has to be available in a plain text file to be usable for the neural networks. The agent has to support data storage in a text file directly or the ability to store them in a database which can export plain text files.
Important is that as training dataset not only frequent prices of one single option are needed. Many different but comparable option prices from several issuers must be collected. The underlying asset has to be equal. Attributes like strike price and expiration date can differ.

# III. Agent PISA

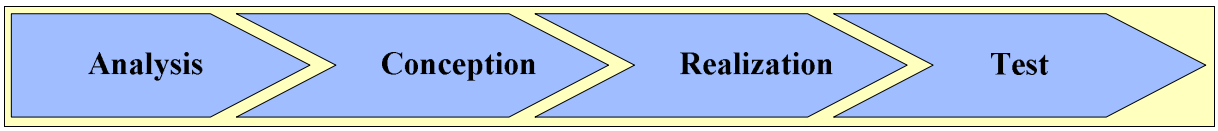
## A. Agent paradigm

The mentioned aim is achieved by using software agent technology. Pattie Maes (1994) developed one of the first software agents with the aim to reduce work and information overload through agent technology. Primary advantage of using an agent instead of a person is that agents are much more effective than employees and that they can work 24 hours a day and 7 days a week (except breakdowns and maintenance work). In addition to effectiveness artificial agents can work more efficiently. Where a human employee requires regular payment for daytime and extended benefits off the regular hours of work an agent contents itself with electricity. Agents can handle dozens of extraction operations per minute automatically. Being able to work a longer period of time with lower costs and their higher efficiency make software agents qualified for webmining tasks.
There is no general accepted definition of the term *software agent* although the difference between normal programs and software agents has been discussed intensely for many years now (Franklin and Graesser (1996), Bradshaw (1997)). A common characterization was disposed by Jennings and Woodridge (1995). Their definition is the origin for almost all current research on agent technology. Accordantly an agent is a represen-

**Figure 3**

**Standard software development process**

*The figure schematically shows the standard software development process. The process contains four separate phases. Within the analysis phase the future system environment is analyzed for software requirements. All necessary functions and abilities have to be identified and summarized. From these identified requirements within the conception phase a theoretical software concept is developed. This concept contains components of the software and their functionality. The conception phase contains a consideration which programming language to use. The complete concept has to be carefully checked if it can accomplish all identified requirements. Within the realization phase the approved concept is programmed. Afterwards the program has to be tested under real life conditions.*




tative which works on behalf of a person. Thereby it has the following attributes: *Autonomy* – The agent should be able to execute the assigned tasks on its own without any callback. *Social behavior* – Agents interact with other agents and at least with the user. *Ability to react* – Perception of the system environment the agent is "living" in and the ability to react on basis of more or less precisely defined decision patterns. *Consciousness* – An agent does not only react on events but is proactive. The hardest to realize attribute certainly is the last one. The first three attributes are much easier to achieve. Consciousness is reserved for so-called high-level agents. The proficiency of the four characteristics depends on the agent's aims. Here, the agent has to receive and process webpages automatically. Caglayan and Harrison (1997) and Murch and Johnson (1998) present an overview of existing agent applications. Autonomy is required inasmuch as the program has to work over a long period of time without necessary user interaction. User interaction is however necessary for configuration. This requires only little social behavior. The agent has to react on the perceived situation. Consciousness is not necessary since all necessary decisions can be made using hard coded rules.

There are miscellaneous levels of agent intelligence. The user predefined information extraction from a website requires only minor agent intelligence. A case differentiation within the program code is sufficient because all possible cases can be planned. The decisions the agent has to make are which text to extract and what to do if the pattern was not found. As the structure of the processed webpages is known in advance the agent just has to filter the user specified pattern without "thinking". All situations appearing unknown can be handled by standard decision rules. This does not suffice to call the agent intelligent. The presented agent is called a partial intelligent agent and its intelligence will be further developed.

The development process equals the standard software development process. The four main process phases are illustrated in Figure 3. The analysis phase accords with the next subsection where the requirements are analyzed. The conception phase consists of two subsections where a programming language and agent's architecture conception are chosen. The agent's testing is described in an own section.

## B. Requirements

This subsection describes the analysis phases of the agent development process. Like every program the agent has to fulfill general and special requirements. The general ones are typical for any software development project (Gomer and Budimir (2000)). The agent's *performance* is determined as the needed time from website loading to saving the data. This period has to be short enough to process an adequate number of extractions within an adequate small time interval. Here, the interval length is determined by the interval period in which the paged are requested. A standard interval is one minute for many applications. Usually at an office workplace standard personal computer are used. The agent's performance shell be adequate for such computers. An inefficient use of the executing system's resources results in decreasing performance. The programming language must provide efficient methods to set unused memory free automatically. These methods are called "Garbage Collection". At least these features have to be implemented manually.

In order to assure that the agent also accomplishes future performance needs the agent has to be *scalable*. The number of extracted patterns should be limited only the system's power. The agent has to be flexible enough to be executed on an increasing number of computers simultaneously, if necessary. The resulting data could be merged manually. It follows that the agent must be applicable on as many operating systems as possible. The language the agent is programmed in has to be available for as much environments as possible as well.

Here, for the neural network market prices synthesis a continuous data flow is mandatory. The agent has to be permanently available during trading hours. Furthermore, the hazard of breakdowns has to be minimized. In case an extraction task can not be processed the agent has to notice this and terminate the task manually to set the system's resources free. To assure this the programming language has to be as robust as possible.

The input data the neural network processes have to be correct. Only a few outliers are tolerable. Therefore the agent's work *accurateness* is very important. One way to approve this is to extract each price from several websites redundantly. A side effect is that in case of wrong provided prices the agent can recognize and fix them.

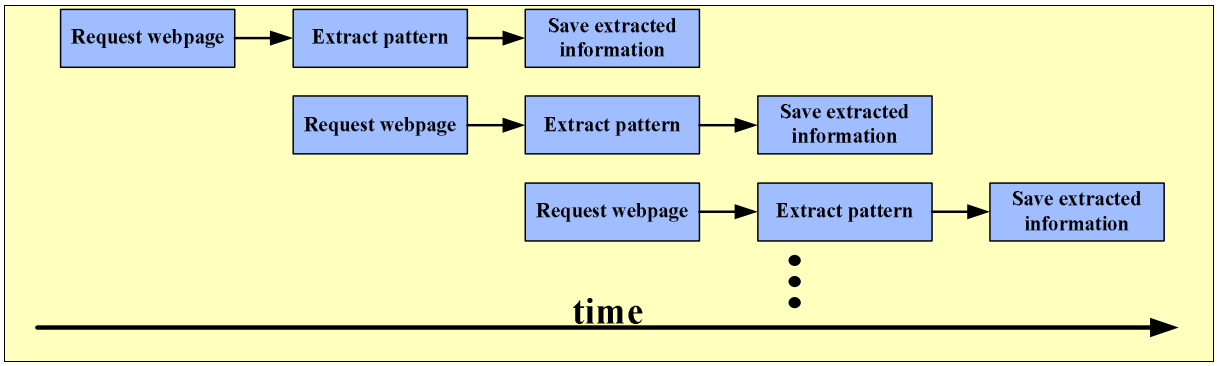
Beside these general requirements there are the following additional ones, derived from the required functionality. First the agent must have the ability to download webpages from the Internet. This requires the support of the accordant Internet protocols by the programming language. The most common protocols with which client computer communicate with web-servers are TCP (Transmission Control Protocol), IP (Internet Protocol) and HTTP (Hypertext Transfer Protocol). The support of these protocols is further mentioned as *Internet functions*.

The addresses of the needed webpages mostly are not a priori known to the user. Investigating them manually is uncomfortable. Beside requesting and loading HTML-files, the agent has to be able to follow hyperlinks included in the source code of a given website. This behavior is called crawling. Hyperlinks usually address external webpages directly by a complete URL. Usually a relative hyperlink is used if the related webpage is located on the same webserver as the referring page. Relative hyperlinks describe the position of the addressed page within the web-server's file system. The server's Internet address is not contained in relative hyperlinks. Webpages are only addressable with a complete URL. The complete web address is automati-

**Figure 4**

**Increasing performance by using simultaneous threads**

*Usually programs start a new task when the previous task is finished. Programs performance can be measurably increased when simultaneous threads are used. Each thread is executed beside others. Each thread consists of several tasks. Each task usually requires different parts of a computer. To assure efficient use and avoid idle time these components can execute tasks of other threads. Here, efficient usage of a computer is a very important requirement for processing several hundred webpages within short period. An insufficient number of webpages can result in a gap in the time series. Such a gap must not appear if the data are used for training neural networks.*




cally generated by the requesting program. The agent has to be able to handle both absolute and relative hyperlinks. The downloaded file has to be analyzed and the user defined patterns have to be extracted. Extracting hyperlinks and regular information patterns works equal. A common way to implement such functions to use so called *regular patterns*. These are special characters representing a characters pattern. Using special methods a text can be checked on appearance of this pattern and the congruent parts can be extracted. Both Jeffrey Friedl (2002) and B. Chng (2002) give a detailed introduction to functionality and usage of regular expressions. Such expressions and methods should be supported by the programming language as well. Alternatively simple *string tokenizer* functions can be used. These functions represent the minimum ability of text manipulation the language has to support. Nevertheless these string tokenizer functions would be neither comfortable nor powerful compared with regular expressions.

Executing the user's task the results have to be saved, either in a text file or a database. The simplest way to save the results is to put the data into a plain text file. This requires the ability of *file handling* operations. Large amounts of data can be better managed and analyzed when they are available in a database. It is reasonable to choose a language that supports database standards like the ODBC-protocol (Open Database Connectivity) to assure compatibility. ODBC is a widely spread standard protocol for external database access that is supported by almost every current database. Using a database the necessary text file for the neural network patterns comfortably can be created as a selection output from the database. This facilitates a further preparation of the extracted data, e. g. invalid datasets can be filtered and outliers can be determined.

Concerning the necessary performance the agent has to be able to execute several tasks simultaneously. This feature is called *multithreading* and is one of the most important requirements the programming language has to comply. Without multithreading abilities the agent would spend plenty of time with waiting. This decreases the necessary performance, see Figure 4.

The agent has to be able to start these threads at user specified times, i. e. *timer functions* are needed.
An independent software-agent must not only run on one special platform to increase scalability. Especially multiple operating systems have to be supported, at least both the UNIX and Microsoft Windows families. *Platform independency* is a very important criterion, here.

## C. Choice of a programming language

Choosing the best programming language is a task where an all-embracing comparison is impossible. Only the most popular and most powerful languages are considered, here. There are others that might be capable as well. We restrain the view to the following languages which will be considered because of their individual properties. A more general analysis of agent technologies is given by Nwana and Woodridge (1997).

**PHP**

PHP (PHP: Hypertext Preprocessor) is an HTML-embedded scripting language originally invented for realizing dynamic webpages. An informative introduction to PHP and its applications is given by Welling and Thomson (2003). The close relationship to the Internet and the fact that PHP is provided for all current operating systems make the language interesting for the web mining agent. The syntax is very similar to C, Java or Perl and makes it easy to learn. A PHP program's source code is completely embedded in an HTML file. When a client computer requests such a page, the web-server, web-server in terms of software, sends the parts that are marked as PHP code to an interpreter-software called PHP-Parser. This interpreter-software executes the script and sends the result back to the web-server that puts these parts into the original website. This website will be send as response to the requesting client in pure HTML. In case the PHP script is correct the client's browser receives a normal website.
PHP functionality, especially the possibility to generate graphical components on-the-fly, makes it comfortable to create a graphical user interface embedded in a conventional HTML file. The program can be executed from all over the world by an Internet browser and runs on the webserver. This is an advantage and also a disadvantage. The necessity of a running webserver restrains the user in running the agent wherever he wants unless there is a webserver installed.
Although the syntax for objects and classes has been improved over the years, there are important functions missing, e. g. the principle of overloading that increases the possibility of reusing program code is still missing. PHP does not support any timing function. The only way to start and end an agent's actions is to use external programs like the Windows Task Planner or the Linux Crontab. A while/until-loop can be used to check the current time and to decide whether to start an action or not. This uses plenty of a system’s resources just to check the time every few milliseconds. This would affect the agent's performance negatively. Another disadvantage regarding the performance is the missing multithreading support. Only one task can be executed at a time and makes the system not capable to fulfill the given task of extracting about 100 warrant prices every minute. All other requirements like object based orientation and support of the necessary Internet protocols are fulfilled. In summary, here, PHP is not a capable language for a partly intelligent agent with the given task because of the missing timer and thread functions.

**JavaScript**

The fact that PHP uses webserver capacity the agent is running on is disadvantageous. JavaScript is a scripting language that uses the capacity of the client computer instead (Flanagan (2001)). As well as PHP JavaScript is a popular scripting language that allows creating a graphical and ubiquitous user interface. A webserver is not necessary. A JavaScript program's source code is similar to PHP embedded in the HTML source code but not interpreted on the server's side. Java-Script is interpreted by the requesting browser. The fact that the necessary interpreter software is implemented in nearly 100% of the current browsers and their gratis availability for current operating systems provides the possibility to use the browser as a graphical user interface without using a locally installed programming language. In addition the necessary performance for multiple simultaneously agent executions can be spread over several computers. This makes the agent independent from a single server.

JavaScript is not an object oriented but object based programming language and supports some important attributes which allow a high programming code reuse. The most important supported principles are encapsulation and inheritance. Overloading is not possible what constraints reusability.

Necessary protocols for requesting and receiving Internet webpages are supported as well as methods to handle regular expressions. So far an agent for extracting information from the Internet can be completely realized in JavaScript. But the language does not support any file handling or database access method. This makes it unusable for a mining agent because a very important part of the mining process is to save the extracted information for further handling. Here, JavaScript is not a capable solution for programming a software agent.

**Perl**

The language Perl was developed in 1987 and is sufficient object oriented since version 5, released last year. A compact introduction to Perl is published by Randy Reames (2002). For more detailed information look at Schwartz and Christiansen (1997). Perl is a scripting language whose interpreter software is running on the program executing system. It is optionally possible to parse requested webpages by an interpreter program similar to PHP. This provides the possibility to create a graphical interface using HTML files loaded over the Internet. In difference to JavaScript and PHP this is an option and not a necessity. Since a script has to be compiled on each executing system the solution of implementing it in a website is an advantage. Otherwise the Perl interpreter software, which is available for all current platforms, has to be installed on each system the agent is used on. This complicates usage.

Perl supports all necessary Internet protocols and regarded regular pattern methods. Extracted data can be saved in text files as well as in databases since the ODBC standard is supported. The most important difference to the languages mentioned above is multithreading support including synchronization of the running processes. So, several tasks can be executed simultaneously without interfering each other.

The easiest way to realize a timer is using implemented methods of a language. Perl does not provide such functions. Because of the popularity and the evolutional development for over 15 years many freeware scripts are available in the Internet. Using them makes it possible to implement most of the missing functions. In case there are no Perl scripts for a special problem, C, C++ or Java code can be embedded in the Perl source code as well. The necessary

scripts are available for free. External scripts are usually not well documented and correct functioning is usually not approved. As far as possible no external programs should be used.
Using threads it is necessary to be able to lock special objects for certified use, e. g. a writing operation into a text file where the extraction results are saved. Only one access to a file is possible at the same time. Only one thread may get access to write into a file at the same time. Otherwise information could be lost or errors could occur. Such locking features are supported by Perl.
Altogether Perl can be considered as a capable language for realizing agents, here. The only aspects confining this consideration is that timer functions have to be implemented by using external programs and that Perl does not support an internal error handling. This would facilitate programming an independently working agent. Perl actually is a capable solution proved through existing agents completely programmed in Perl. Tommie Jones (2002) documented an example for a webmining agent on his Website.
Because of the mentioned constraints further high-level languages are considered. These have, compared with scripting languages, more needed functions implemented inherently and are more efficient. Regarded are C++, C# and Java. Visual Basic also is a very popular language but because of the platform dependency, it is not considered. All considered languages facilitate loading webpages using the TCP/IP-standard and saving data in text files as well as in databases using ODBC.

**C++**

The language C++ is a successor of the language C. Primary improvement is the complete object orientation in C++. This includes inheritance, polymorphism and encapsulation. These characteristics make the language very powerful and enable high code reusability. For more detailed information, see Stroustrup (2000). Further improvements are advanced error handling and possibility of method overloading. The language does not contain any method for parsing source code. This has to be programmed manually. Although a lot of such modules are available cost free in the Internet, this is not a target-oriented approach as their functionality is not necessarily given. Another disadvantage is that external programs usually are not well documented. For realizing such a feature manually C++ supports regular expressions as well as less advanced string manipulation methods. So far the language seams to be capable for the current aim and the affinity to C facilitates existing knowledge and program code reuse written in C.
Simultaneously executed threads can only be realized in C++ by using external software packages. These are usually not platform independent. This makes the agent platform dependent. Maintaining independency means using different packages for each kind of environment. In addition these packages usually do not provide timer methods. These can be implemented by using externals packages as well. Most likely the cooperation of the timer and the thread module would be difficult.
Summarized C++ is a capable language for realizing the agent in general. Platform independency is constricted if external software packages are used, since needed capabilities are not included in standard C++. This is why this language is considered as improper, here.

**Java**

Java is a new language grown up with the Internet and is taken into consideration because of two reasons. First Java was developed with a network oriented approach. Normally programs

only run on the computer they are installed on. Java programs can be executed on remote computers without the necessity of a local installation. A graphical user interface is as easy to realize as in a scripting language. This is the primary reason for considering Java. Secondly remote execution requires a platform independent language design because in a network usually many operating systems are used. Since the extraction agent is intended to run on most current operating systems this is an important feature. A practical introduction to Java and its architecture is given by Bloch (2001).

Java works differently compared to the other languages mentioned above. Programs realized in a conventional language are compiled directly into an executable file. Java splits this process into two parts. First the compiler creates so called *Java Byte Code* of a program's source code. This Java Byte Code can be transferred to any other computer where it is executed by the *Java Virtual Machine*, a platform dependent interpreter software. The Java Virtual Machine is available for most current platforms. Platform independency is assured.

The network oriented approach provides the possibility to realize a graphical user interface that can be remotely accessed. A graphical user interface is important for a comfortable handling. Java program can be executed remotely in two ways. First it can be started by special tools. These tools access the remote computer and pass the user interface to the evoking computer. This technique withdraws the necessity of a running webserver for remote execution. Second, in contradiction to C# or C++, Java programs can be integrated into a website. The agent can be executed on a central server by starting it from a distant computer by opening the website. No webserver is needed, since the program is executed by the *Java Virtual Machine* included in most common browsers. The website envelops the program. Popular application ranges are security sensitive applications, e. g. online banking solutions. Such webpages contain so called Java Applets with which a user can handle his banking transactions securely. Security has been a very important aim developing Java to avoid misusage. The language's design enables the user to start the agent either remotely, with or without a webserver, or locally. Altogether Java is a very flexible language what assures adequate scalability, too.

Java provides all necessary functions. All current protocols for Internet use are supported. The wanted patterns can be extracted by using regular patterns. For further information regarding regular expressions in Java, see the website of Dana Nourie and Mike McCloskey (2002). Results can be stored in files and databases. Java does not directly support ODBC but uses an own approach called JDBC (Java Database Connectivity). It is possible to access databases via ODBC because corresponding methods are included in the standard Java development kit.

In Java threads can be created by succeeding the Thread-object. The resulting objects can be started and ended with special timer functions. Synchronizing running threads is possible as well as locking specified objects in order to protect them from competing access. Java has, in contradiction to C++, an internal error handling and a powerful garbage collection that automatically sets unused memory free.

Altogether Java is a capable language for realizing the agent without any compromises. No external programs have to be used. The language's network oriented origin enables an adequate grade of scalability. If the agent needs more resources the tasks can be split and executed on different computers. The capability is emphasized by a noticeable trend: Many agent development environments are realized in Java. This shows that Java's qualification is

**Table I**

**Comparison of the discussed programming languages (March 2003)**

*The agent's programming language must accomplish some necessary requirements, shown in this table. The six considered languages are each checked for nine functions. The sign ✘ means that the function is not supported, the token ✓ stands for support. PHP and Java-Script do not support multithreading and error handling functions. This constrains performance and accurateness. Both PHP and Java-Script are not capable solutions for programming an agent. The language Perl does support multithreading. Error handling methods can be programmed manually. For a scalable agent remote method invocation has to be supported which is missing at Perl. C++ supports all necessary functions except timer functions. These can be realized by using free external program packages available in the internet (missing does not really matter and the symbol is set in brace). Both C# and Java support all necessary functions as well. C# only works with Microsoft Windows. Java's platform independency is an important advantage. Here, Java is chosen for programming the agent.*

| ⇩ Functions \| Languages ⇨ | PHP | Java-Script | Perl | C++ | C# | Java |
|---|---|---|---|---|---|---|
| Internet functions | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Regular expressions | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| String tokenizer functions | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Multithreading | ✘ | ✘ | ✓ | ✓ | ✓ | ✓ |
| Error handling | ✘ | ✘ | ✘ | ✓ | ✓ | ✓ |
| Timer functions | ✘ | ✘ | ✘ | (✘) | ✓ | ✓ |
| File handling | ✓ | ✘ | ✓ | ✓ | ✓ | ✓ |
| ODBC database support | ✓ | ✘ | ✓ | ✘ | ✓ | ✓ |
| Remote method invocation | ✘ | ✘ | ✘ | ✓ | ✓ | ✓ |
| Platform independency | ✓ | ✓ | ✓ | ✓ | ✘ | ✓ |

widely approved. Michael Petsch (2001) summarizes the most common development environments for mobile agents.

**C#**

C# (pronounced "see sharp") is a high-level language developed by Microsoft to combine the advantages of C++ and Java and to avoid their disadvantages. This makes C# interesting for the given task. If Java is capable, C# might be even better. The language C# descends from a project called J++, an effort of Microsoft to develop an own Java version until this was forbidden juridically. The closeness to J++ is the reason why the syntax of C# is very similar to Java's. Java and C# also have almost identical functionality.

The most important advantage of C# is the relatedness to the Microsoft .NET-Framework. In this Framework the source code is, similar to Java, not directly compiled into the executable file but in a so called *Intermediate Language*. This is a processor and system independent program code that is executed by a runtime environment. .NET is intended to enable compiling not only C#-code but also any other language into intermediate code. Therefore programs consist of components coded in different languages. This would increase the agent's extensibility. In March 2003 the Framework does not support many languages and it is only available for Microsoft Windows.

The mentioned advantages approve usage of C# for realizing the agent. Although C# is as capable as Java, it is not considered any further. Platform independency is not adequate and the costs for Microsoft licenses are significant.

**Choosing the best**

Table I summarizes the most important properties of the discussed programming languages. Which language the agent should be programmed in can not been answered objectively. Apparently important requirements are not fulfilled by PHP and JavaScript. For the third scripting language Perl the missing garbage collection, timer classes and uncomfortable error handling constrict its applicability. Nevertheless Perl is a capable language for realizing agents in general. Although Perl is, like mentioned above, a capable possibility it is not considered any further because C++, C# and Java are superior.

The most important argument in favor of C++ is productivity. If there are existing programs written in C++ they can be reused with minor modification. The syntax of C++ bases on C so that all existing knowledge, in case there are C-programmers, can be used. Java is a completely new language with new syntax which makes it harder to reuse old programs or knowledge. Since the agent is a completely new development this advantages do not single out Java and C# because no compatibility to existing systems is necessary. Another counter argument towards Java is that in the upcoming future agents should be as mobile as possible. This makes an interpreter language like Java more capable than C or C++. This is the most important reason why most existing development environments for mobile agent are realized in Java.

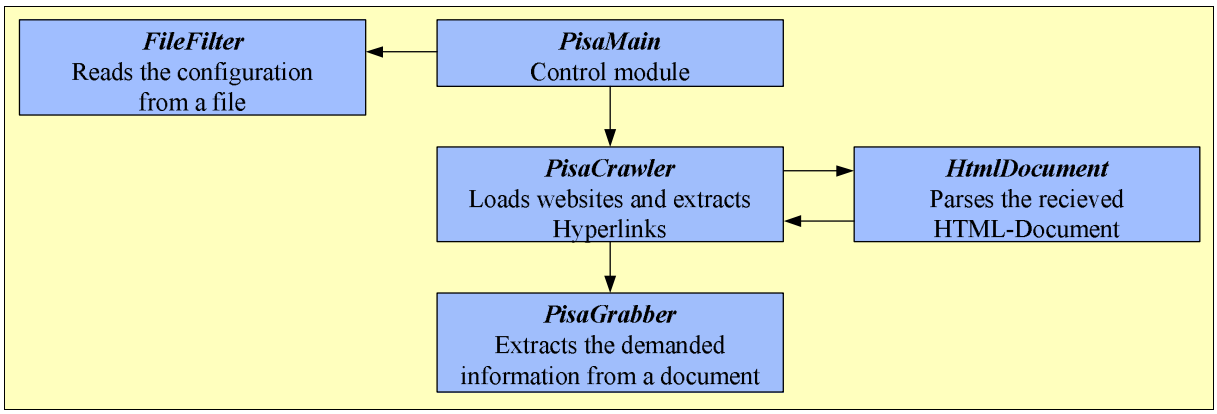
Derogatory to Java is that its architecture slows down execution performance. The performance of C++ is more efficient. Java splits into compiler and interpreter parts. This results in an overhead. The garbage collection mechanism in Java slows down the language as well. A third reason for the overhead of Java programs are the used secure thread-libraries. These libraries stabilize multithreading programs and protect them from affecting each others negatively. A secure runtime environment that can deal with multiple and simultaneously running threads is strongly necessary for the agent as well as an efficient use of the available system resources. Because of this little less performance is not as disadvantageous as an insecure and inefficient agent. Missing security would decrease agent's autonomy because a user has to check correct functioning frequently.

A current tendency in agent development is to enrich existing programming languages with agent oriented libraries. A trend towards Java as the basic language is noticeable. Considering the mentioned advantages and disadvantages of C++ and Java the latter is the better one, here. Finally the choice has to be made between Java and C#. The idea behind C#, mostly the .NET relatedness, usually makes it the better choice. Here, the agent has to run on Windows- and Unix-Systems. Although Microsoft works on a runtime environment that is supposed to run on every operating system the current version exists only for Windows. Java as programming language for the agent is the best choice, here.

**Figure 5**

**Components of the agent PISA**

*PISA consists of five components. The component PisaMain initiates and starts all user defined extraction tasks. The user made definitions are taken from the configuration file. The configuration file is processed by the component FileFilter. The initiated tasks run as background threads. That means that all tasks can be executed simultaneously. Each task is represented by a single PisaCrawler object. These objects request the defined webpages and approve that during the crawling process no webpage is requested multiple times. Each received webpage is process by the HtmlDocument component. This component parses the passed website's source code. For each HTML tag an array is returned. These arrays are further processed by the PisaGrabber component which identifies and returns the user wanted patterns. After the extraction the PisaCrawler object saves the data either in a plain text file, XML-file or a database.*




## D. Software-architecture

The agent PISA consists of five main components, see Figure 5. PISA starts with executing the main module called *PisaMain* that initiates the extraction tasks defined by the user. The configuration of these tasks is saved in a file with a specific syntax. Every object is collated to an extraction object. In order to build precise objects out of the information in the configuration file the component *FileFilter* reads one or more configuration files and returns their content as objects to the evoking PisaMain module. The PisaMain-object starts exactly one *PisaCrawler* object for each URL. Per URL many information pieces can be extracted, e. g. the current option price and it's update time. Each piece of information within a PisaCrawler element is represented by a *PisaGrabber*-object. A PisaGrabber-object "grabs" the wanted information out of the webpages. To enable the PisaGrabber-object to extract data the webpage's source code is processed by an *HtmlDocument*-object.

The PisaMain-object starts the PisaCrawler-objects with a one second time delay, each. Otherwise it can happen, depending on the configuration, that a webserver receives dozens of requests within less than one second if they are send out one after another without time delay. This can result in a server crash or at least in not answering some of the requests. The latter results from a standard mechanism that avoids webserver overload by ignoring requests when a specified number of requests in a period is exceeded. If page requests are not answered this results in information gaps in the generated time series. The delay time is adjustable in PISA's configuration file.

**Figure 6**

**Derivative quotes marked in HTML and XML**

*This figure shows an option's quotes marked in HTML and XML. The HTML source code is structured as a table. The quotes are all marked with the same kind of tag that describes the content of a table cell. The given parameters for the background color are equal. The extraction program gets no information about the meaning of a cell's content. Extraction of the wanted pattern by its surrounding tag is not possible. This complicates extraction. Identifying the wanted piece of information using a regular pattern is not target-oriented as well. E. g. the pattern for the current price also matches the change rate. In the XML code each piece of information is marked with an explicit XML-tag which gives information about the regarding context. Even if several options are published within the same webpage each one can be clearly identified by the given parameters. Here, parameters for the German security identification number (Wertpapierkennnummer = WKN) and the option type are given.*

```
01  <table>
02        <tr>
03           <td bgcolor="95b4ff" >Euro/Dollar Call option</td>
04           <td bgcolor="95b4ff" >782367</td>
05           <td bgcolor="95b4ff" >6,18 €</td>
06           <td bgcolor="95b4ff" >5,11 %</td>
07        </tr>
08  </table>
```

Example of a standard HTML-Table containing a derivative quotes.

```
01  <option wkn="782367" type="Euro/Dollar Call option">
02    <price>6,18 €</price>
03    <change>7.8.2002</change>
04  </option>
```

Example of a common XML-Tag containing the same derivative quotes.

## PisaCrawler

A *PisaCrawler*-object accomplishes the given tasks in a specified interval. It can be configured to start and end at a specified time in the future. Within this period it is executed in user defined intervals. Without given interval it terminates itself after the first evaluation.

Each object and each pass have the same operating cycle. First the object creates an *HtmlDocument*-object that contains the passed URL's source code. In case the user wants to extract information not only from the page identified by the URL but also from webpages the basic site is linking to, the *PisaGrabber*-component calls a special routine of the *HtmlDocument*-object. This method extracts all hyperlinks from the site and returns them in an array. This array is successively processed in the same way like the first URL. In order to avoid loading a page twice each requested URL is checked if it has been already requested before. The ability to crawl through a website gives this component its name. PISA works until the depth of webpages is reached.

Important is that each PisaGrabber-object is a single and independent thread. All started PisaCrawler-objects can work simultaneously and independently. Each object can have different parameters and properties. Here, this construction is very important because otherwise PISA would waste a lot of time by waiting for requested webpages to be received. PISA is requesting new webpages while waiting. This increases the performance significant, see Figure 4.

**HtmlDocument – parsing the source code of loaded webpages**

The *HtmlDocument* component's major task is to request and analyze webpages. The evoking class passes an URL. The related webpage is requested and, if the address is valid, the source code is received. If several addresses are requested simultaneously the crawling process performance can be increased. Therefore, the HtmlDocument-object is realized as an independent thread.

An information extraction program is much easier to create, if all webpages were written in XML. XML-tags usually describe which kind of value they contain. One single pattern can be used to identify a wanted piece of information, see Figure 6. The current standard is HTML. The markup language HTML is intended for human consumption and not for machine interpretation. For HTML a two-stage process is necessary. First the submitted source code has to be parsed. It is analyzed for each needed HTML-tag. For each kind of tag one array is created. Every array-field contains the content of exactly one tag of a kind. The tags are stored in order of appearance. This enables a successive comparison of the array fields and the search for a special pattern.

In the second step, the needed pieces of information are identified. This is made by the PisaGrabber-object.

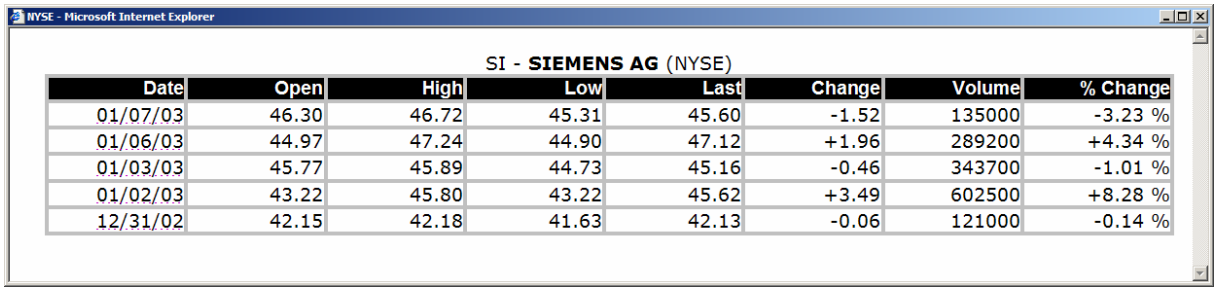
**PisaGrabber – Extracting patterns**

The PisaGrabber component exploits the fact that many webpages are generated automatically from an underlying database. The information from the database is usually inserted into HTML templates. The structure of the webpages within one website is equal or at least very alike and the HTML structure is fairly specific and regular. Such texts are called semi-structured. A standard approach for extracting information from semi-structured text is using delimiters for identifying needed pieces of information. Software which accomplishes the tasks of extraction is often called wrapper because it wraps the extracted information into delimiters. This enables comfortable further handling. For more information about information extraction from the World Wide Web using pattern, see Eikvil (1999).

The component PisaGrabber extracts the demanded patterns from a given HTML-Document which is passed as an *HtmlDocument*-object from the PisaCrawler module. The easiest way to extract a pattern is successive comparison of each array field with the wanted pattern. The user could define the number of the searched pattern, e. g. the second appearance of a 2-digit decimal number. This approach is neither comfortable nor capable because the number of appearance might change. If an option's bid-price is appearance number two of a 2-digit number today, it can be the third one tomorrow. This makes it hard to keep PISA working and finding the right pattern. Instead, a different approach is chosen. Here, the number of a requested pattern is defined relative to an anchor. This anchor is also defined by a pattern. An example clarifies this procedure: A typical table with stock prices is shown in Figure 7. The last given price can be found by using a pattern for a 2-digit decimal number. The current date can be used as the anchor which can be identified by using a suitable pattern. The current price position is the forth table cell after the current date. To specify the wanted information and its anchor pattern regular expressions are used, e. g. the pattern for a 2-digit decimal number with exactly two decimal places is "**[d]{2}[.][d]{2}**".

**Figure 7**
**An example of a webpage presenting the quotes of the Siemens AG at the NYSE**

*Usually stock, derivative and warrant quotes are published in tables like pictured below. This highly structured layout can be exploited to extract certain pieces of information. The simplest way of identifying a piece of information is to compare the webpage's source code with a specified pattern. In case of multiple appearances the number of the wanted information can be specified. E. g. the last published price for the Siemens stock is the fourth appearance of a two digit decimal number with two decimal places. This solution is comfortable but not reliable and in some cases not even effective. The number of a pattern's appearances changes, e. g. when the lowest price becomes a one digit number the current price becomes the third appearance. An alternative approach is to define a pattern by specifying its relative position to a defined anchor. E. g. this anchor can be the current date 01/07/03 and the wanted pattern is contained the fourth table cell right of this anchor. Only this cell is analyzed for appearance of a two digit decimal number. In case the webpage's structure before or behind the quote table changes this does not affect the extraction process. Only if the table's structure changes the pattern has to be modified.*


NYSE - Microsoft Internet Explorer

SI - **SIEMENS AG** (NYSE)

| Date | Open | High | Low | Last | Change | Volume | % Change |
|---|---|---|---|---|---|---|---|
| 01/07/03 | 46.30 | 46.72 | 45.31 | 45.60 | -1.52 | 135000 | -3.23 % |
| 01/06/03 | 44.97 | 47.24 | 44.90 | 47.12 | +1.96 | 289200 | +4.34 % |
| 01/03/03 | 45.77 | 45.89 | 44.73 | 45.16 | -0.46 | 343700 | -1.01 % |
| 01/02/03 | 43.22 | 45.80 | 43.22 | 45.62 | +3.49 | 602500 | +8.28 % |
| 12/31/02 | 42.15 | 42.18 | 41.63 | 42.13 | -0.06 | 121000 | -0.14 % |

Using this approach the user has to know four important things from the requested pattern:

1. The pattern of the requested information (In the example the price is a 2-digit decimal number);
2. The pattern of the anchor (The current date is used in the mentioned example);
3. The number of tags between the anchor-pattern and the wanted information;
4. The kind of tag the patterns are formatted with, i. e. in most cases this is the standard tag for table-cells "<TD>".

Optional the user can define names for the each extracted piece of information. This enables storage in platform independent XML-files.

Beside the extracted information the loading time and the time needed for the parsing process are stored in a special file by the PisaGrabber-object. This enables further analyzes regarding the capability of websites for extraction jobs.

**Resulting File Formats**

Extracted data are internally stored in an array which makes it easy to save the results in plain text files. The array is an associative array. It is possible to save the extracted data in XML-files. The array-key is used as the XML-tag's name. The resulting XML-file only contains the data and no information about the layout.

If the results are stored in a plain text file they are not separated by any characters. The columns have equal widths. This makes it possible to read this file with every program without care about eventually not supported delimiting characters.

Extracting data from several webpages leads to the problem that the display format of numbers and dates do most likely differ from each other; e. g. a date can be formatted like `02/22/03` or `02/22/2003` or `02/22` or in the European notation `22.1.2003`. If for further handling a database or spreadsheet program is used, these programs usually need a special format to recognize the kind of information. Otherwise the data would be handled as ordinary text. In order to assure this in the configuration file a special format for the extracted patterns can be defined. PISA formats text, numbers and dates to facilitate result import in retailing programs. Correct formatting is initiated by the PisaGrabber-object before the results are saved.

# IV. Agent testing

## A. Test specifications

With the test correct functioning of PISA is proved. PISA has to be available during trading hours, it has to work automatically and it has to be able to handle occurring errors. Secondly the long-term test shows whether the developed extraction approach is reliable for extracting the wanted information.

Analyzing the extracted data mainly the following questions are considered:

- Are the time series dense enough for neural network training?
- Are the prices published cost free in the Internet up to date (Only reliable and up to date information is useful!)?
- Is every website equally capable to extract the wanted information? Are there differences in information quality and regarding to the loading time?

In addition the following questions are considered:

- Are there any differences in the prices regarding the issuers of options and warrants?
- If there are differences in the prices regarding the issuers, are they constant and do they occur for all options of an issuer?

Neural networks need dense time series which can be most suitable extracted from liquid derivatives. Here, German options are chosen for the test. Some exchange rate options have a comparatively high trading volume. Experiences of the authors shows showed that €/US-$ call warrants are suitable. Therefore they are chosen for the test, here. Options of three issuers are considered. In the best case the issuer publishes his prices on his website. The price extracted from other webpages then can be verified easily then. The issuers "Citibank Group",[6] "UBS Warburg"[7] and "Deutsche Bank"[8] publish their warrant prices on their websites. Therefore and because they are the most important issuers in Germany their options are chosen for the test.

If product prices are going to be compared, the products themselves have to be comparable. Here, the products are German warrants, i. e. certified options, and their most important price-defining properties are the expiration date and the strike price. Regarding to the expiration date the 4 most important ones are chosen, each in the middle of the months March, June, September and December. The exact date is not the same in all cases. The "Citibank Group"

---

6 Citibank web address: http://www.citibank.de

7 UBS Warburg web address: http://www.ubswarburg.de/

8 Deutsche Bank web address: http://www.deutschebank.de

**Table II**

**Analyzed options with issuer, strike price and expiration date**

*The table shows all considered options, their German security identification number and the related issuer sorted by expiration date and strike price. Four different strike prices are chose. For each strike price four different expiration dates are considered. For each expiration date and strike price combination three options are extracted. The option's issuers are UBS Warburg, Deutsche Bank and Citibank. Deutsche Bank does not offer an option with an expiration date in June 2003 and a strike price of 1.05 $. The option's quotes for the overall 59 options are extracted from the issuer's pages and additionally from the webpages of Citibank and "Onvista – The eFinance Company".*

| ⇩ Exp. date | ⇩ Issuer \| | Token ⇩ \| | Strike ⇨ | 0.90 $ | 0.95 $ | 1.00 $ | 1.05 $ | 1.10 $ |
|---|---|---|---|---|---|---|---|---|
| Mar 2003 | UBS Warburg | UBS | | 574297 | 574299 | 574301 | 574303 | 574305 |
| | Deutsche Bank | MAX | | 782363 | 782365 | 782367 | 782368 | 782369 |
| | Citibank | CIT | | 582577 | 582579 | 582581 | 582583 | 582585 |
| Jun 2003 | UBS Warburg | UBS | | 574311 | 574313 | 574315 | 574317 | 574319 |
| | Deutsche Bank | MAX | | 782388 | 782390 | 782392 | ----- | 689469 |
| | Citibank | CIT | | 640962 | 640964 | 640966 | 640968 | 640970 |
| Sep 2003 | UBS Warburg | UBS | | 638663 | 638665 | 638667 | 638669 | 638671 |
| | Deutsche Bank | MAX | | 782465 | 782467 | 782469 | 782470 | 782471 |
| | Citibank | CIT | | 640976 | 640978 | 640980 | 640982 | 640984 |
| Dec 2003 | UBS Warburg | UBS | | 638677 | 638679 | 638681 | 638683 | 638685 |
| | Deutsche Bank | MAX | | 782510 | 782512 | 782513 | 782515 | 689470 |
| | Citibank | CIT | | 667962 | 667964 | 667966 | 667968 | 667970 |

dates their options seven days later than "UBS Warburg" and "Deutsche Bank". Relative difference is negligible small, here. Regarding to the strike price five prices between 0.90 and 1.10 US-$ are picked, see Table II.

Additionally to the issuer's webpages each option's price is extracted from two other financial service providers who publish option prices on their website. "Comdirect Bank"[9] and "Onvista – The eFinance Company"[10] are the biggest financial information websites in Germany.

## B. Test procedure

The first test platform is a Microsoft Windows PC with an 800 MHz Pentium III and 256 MB RAM. The second run is accomplished on two Linux PCs with two 2.4 GHz XEON-Processors and 512 MB RAM. The reason for using two completely different environments is to analyze PISA's limits. For the test specifications defined above minimum interval and maximum possible number of tasks per minute are important.

In both cases the test job is started Monday morning at 6 a. m. and terminated at Friday 10 p. m. The extraction job can be paused during nighttime when no German options are traded. Up-to-dateness is evaluated as well as differences in the webserver response time depending on the time of day. To find out when the prices start to change in the morning and stop to change in the late evening PISA runs 24 hours.

In order to evaluate loading time beside the wanted information patterns the time for requesting a website and for parsing are saved. It can be measured if there is any kind of connection between these parameters. The needed time for loading and parsing a website should be short. If the individual predefined intervals are too long the extraction is useless. Neural networks need a dense output of warrant prices, here.

---

9 Comdirect Bank web address: http://www.comdirect.de

10 Onvista web address: http://www.onvista.de

## Table III

## Example of a PISA output file

*The table shows a cantle of a PISA output file generated during the test. All values are identified and saved correctly. In some cases the ask price is not published on the webpage. It is not stored in the output file then. Empty values are not filled with replacement character because this complicates further handling. Files like this output file can be directly transformed into an input data file for neural networks.*

| Option identification token | Extraction time | Ask price | Expiration date | Bid price | Update time | | Strike price | German security identification number |
|---|---|---|---|---|---|---|---|---|
| "574297ONV" | 20.01.03 06:00:00 | 0.000 | 17.03.2003 | 0.000 | 01.01.2003 | 01:00:00 | 0.90 | 574297 |
| "574299ONV" | 20.01.03 06:00:02 | 1.000 | 17.03.2003 | 8.300 | 17.01.2003 | 21:58:00 | 0.95 | 574299 |
| "574301ONV" | 20.01.03 06:00:05 | 6.300 | 17.03.2003 | 3.840 | 17.01.2003 | 21:58:00 | 1.00 | 574301 |
| "574303ONV" | 20.01.03 06:00:07 | 2.330 | 17.03.2003 | 2.300 | 17.01.2003 | 21:58:00 | 1.05 | 574303 |
| "574305ONV" | 20.01.03 06:00:10 | 0.580 | 17.03.2003 | 0.550 | 17.01.2003 | 21:58:00 | 1.10 | 574305 |
| "582579ONV" | 20.01.03 06:00:12 | 0.770 | 17.03.2003 | 0.740 | 20.01.2003 | 05:42:00 | 0.95 | 582579 |
| "582581ONV" | 20.01.03 06:00:15 | 0.000 | 17.03.2003 | 0.000 | 01.01.2003 | 01:00:00 | 1.00 | 582581 |
| "582577ONV" | 20.01.03 06:00:17 | 5.460 | 17.03.2003 | 5.430 | 20.01.2003 | 05:44:00 | 0.90 | 582577 |
| "582583ONV" | 20.01.03 06:00:20 | 2.170 | 17.03.2003 | 2.140 | 20.01.2003 | 05:44:00 | 1.05 | 582583 |
| "582585ONV" | 20.01.03 06:00:22 | 0.530 | 17.03.2003 | 0.500 | 20.01.2003 | 05:35:00 | 1.10 | 582585 |
| "782369ONV" | 20.01.03 06:00:25 | 0.000 | 17.03.2003 | 0.000 | 01.01.2003 | 01:00:00 | 1.10 | 782369 |
| "782368ONV" | 20.01.03 06:00:27 | 2.360 | 17.03.2003 | 2.330 | 17.01.2003 | 00:08:00 | 1.05 | 782368 |
| "782367ONV" | 20.01.03 06:00:30 | 6.340 | 17.03.2003 | 6.310 | 17.01.2003 | 00:08:00 | 1.00 | 782367 |
| "782365ONV" | 20.01.03 06:00:32 | 1.020 | 17.03.2003 | 0.990 | 17.01.2003 | 00:08:00 | 0.95 | 782365 |
| "782363ONV" | 20.01.03 06:00:35 | 5.700 | 17.03.2003 | 5.670 | 17.01.2003 | 00:08:00 | 0.90 | 782363 |
| "574297COM" | 20.01.03 06:00:37 | | 17.03.2003 | 5.46 | 17.01.2003 | 12:56:00 | 0.90 | 574297 |
| "574299COM" | 20.01.03 06:00:40 | | 17.03.2003 | 0.93 | 17.01.2003 | 10:33:00 | 0.95 | 574299 |
| "574301COM" | 20.01.03 06:00:42 | | 17.03.2003 | 6.08 | 17.01.2003 | 12:56:00 | 1.00 | 574301 |
| "574303COM" | 20.01.03 06:00:45 | | 17.03.2003 | 2.20 | 17.01.2003 | 16:19:00 | 1.05 | 574303 |
| "574305COM" | 20.01.03 06:00:47 | | 17.03.2003 | 0.52 | 17.01.2003 | 13:13:00 | 1.10 | 574305 |
| "782363COM" | 20.01.03 06:00:50 | | 17.03.2003 | 5.47 | 17.01.2003 | 11:57:00 | 0.90 | 782363 |
| "782365COM" | 20.01.03 06:00:52 | | 17.03.2003 | 0.80 | 17.01.2003 | 11:26:00 | 0.95 | 782365 |
| "782367COM" | 20.01.03 06:00:55 | | 17.03.2003 | 6.08 | 17.01.2003 | 11:57:00 | 1.00 | 782367 |
| "782368COM" | 20.01.03 06:00:57 | | 17.03.2003 | 2.21 | 17.01.2003 | 19:33:00 | 1.05 | 782368 |
| "782369COM" | 20.01.03 06:01:00 | | 17.03.2003 | 0.53 | 17.01.2003 | 12:32:00 | 1.10 | 782369 |
| "582577COM" | 20.01.03 06:01:02 | | 17.03.2003 | 5.54 | 17.01.2003 | 11:42:00 | 0.90 | 582577 |
| "582579COM" | 20.01.03 06:01:05 | | 17.03.2003 | 0.83 | 17.01.2003 | 12:49:00 | 0.95 | 582579 |
| "582581COM" | 20.01.03 06:01:07 | | 17.03.2003 | 6.18 | 17.01.2003 | 16:50:00 | 1.00 | 582581 |
| "582583COM" | 20.01.03 06:01:10 | | 17.03.2003 | 2.23 | 17.01.2003 | 19:47:00 | 1.05 | 582583 |
| "582585COM" | 20.01.03 06:01:12 | | 17.03.2003 | 0.56 | 17.01.2003 | 19:49:00 | 1.10 | 582585 |
| "582585CIT" | 20.01.03 06:01:32 | 0.58 | 17.03.2003 | 0.55 | 17.01.2003 | 21:55:00 | 1.10 | 582585 |
| "582581CIT" | 20.01.03 06:01:30 | 6.28 | 17.03.2003 | 6.25 | 17.01.2003 | 21:58:00 | 1.00 | 582581 |
| "574297ONV" | 20.01.03 06:02:00 | 5.680 | 17.03.2003 | 3.140 | 17.01.2003 | 21:58:00 | 0.90 | 574297 |
| "574303ONV" | 20.01.03 06:02:07 | 2.330 | 17.03.2003 | 2.300 | 17.01.2003 | 21:58:00 | 1.05 | 574303 |
| "574299ONV" | 20.01.03 06:02:02 | 1.000 | 17.03.2003 | 8.300 | 17.01.2003 | 21:58:00 | 0.95 | 574299 |
| "574305ONV" | 20.01.03 06:02:10 | 0.580 | 17.03.2003 | 0.550 | 17.01.2003 | 21:58:00 | 1.10 | 574305 |
| "574297COM" | 20.01.03 06:02:37 | | 17.03.2003 | 5.46 | 17.01.2003 | 12:56:00 | 0.90 | 574297 |
| "574301ONV" | 20.01.03 06:02:05 | 6.300 | 17.03.2003 | 3.840 | 17.01.2003 | 21:58:00 | 1.00 | 574301 |
| "582579CIT" | 20.01.03 06:01:35 | 10.98 | 17.03.2003 | 10.95 | 17.01.2003 | 21:58:00 | 0.95 | 582579 |
| "574299COM" | 20.01.03 06:02:40 | | 17.03.2003 | 0.93 | 17.01.2003 | 10:33:00 | 0.95 | 574299 |
| "582583ONV" | 20.01.03 06:02:20 | 2.190 | 17.03.2003 | 2.160 | 20.01.2003 | 05:46:00 | 1.05 | 582583 |
| "574301COM" | 20.01.03 06:02:42 | | 17.03.2003 | 6.08 | 17.01.2003 | 12:56:00 | 1.00 | 574301 |
| "582581ONV" | 20.01.03 06:02:15 | 6.080 | 17.03.2003 | 6.050 | 20.01.2003 | 05:46:00 | 1.00 | 582581 |
| "582577ONV" | 20.01.03 06:02:17 | 5.480 | 17.03.2003 | 5.450 | 20.01.2003 | 05:46:00 | 0.90 | 582577 |
| "782365MAX" | 20.01.03 06:01:42 | 11.02 | 17.03.2003 | 10.99 | 17.01.2003 | 00:08:00 | 0.95 | 782365 |
| "582579ONV" | 20.01.03 06:02:12 | 0.780 | 17.03.2003 | 0.750 | 20.01.2003 | 05:46:00 | 0.95 | 582579 |
| "782367MAX" | 20.01.03 06:01:45 | 6.34 | 17.03.2003 | 6.31 | 17.01.2003 | 00:08:00 | 1.00 | 782367 |
| "574303COM" | 20.01.03 06:02:45 | | 17.03.2003 | 2.20 | 17.01.2003 | 16:19:00 | 1.05 | 574303 |
| "574305COM" | 20.01.03 06:02:47 | | 17.03.2003 | 0.52 | 17.01.2003 | 13:13:00 | 1.10 | 574305 |
| "782363COM" | 20.01.03 06:02:50 | | 17.03.2003 | 5.47 | 17.01.2003 | 11:57:00 | 0.90 | 782363 |
| "782363ONV" | 20.01.03 06:02:35 | 5.700 | 17.03.2003 | 5.670 | 17.01.2003 | 00:08:00 | 0.90 | 782363 |
| "782365COM" | 20.01.03 06:02:52 | | 17.03.2003 | 0.80 | 17.01.2003 | 11:26:00 | 0.95 | 782365 |
| "582585ONV" | 20.01.03 06:02:22 | 0.530 | 17.03.2003 | 0.500 | 20.01.2003 | 05:35:00 | 1.10 | 582585 |
| "782369ONV" | 20.01.03 06:02:25 | 0.580 | 17.03.2003 | 0.550 | 17.01.2003 | 00:08:00 | 1.10 | 782369 |
| "782367ONV" | 20.01.03 06:02:30 | 6.340 | 17.03.2003 | 6.310 | 17.01.2003 | 00:08:00 | 1.00 | 782367 |
| "782368ONV" | 20.01.03 06:02:27 | 2.360 | 17.03.2003 | 2.330 | 17.01.2003 | 00:08:00 | 1.05 | 782368 |
| "782367COM" | 20.01.03 06:02:55 | | 17.03.2003 | 6.08 | 17.01.2003 | 11:57:00 | 1.00 | 782367 |
| "782368MAX" | 20.01.03 06:01:47 | 2.36 | 17.03.2003 | 2.33 | 17.01.2003 | 00:08:00 | 1.05 | 782368 |
| "782369MAX" | 20.01.03 06:01:50 | 0.58 | 17.03.2003 | 0.55 | 17.01.2003 | 00:08:00 | 1.10 | 782369 |
| "782365ONV" | 20.01.03 06:02:32 | 1.020 | 17.03.2003 | 0.990 | 17.01.2003 | 00:08:00 | 0.95 | 782365 |
| "782363MAX" | 20.01.03 06:01:40 | 15.70 | 17.03.2003 | 15.67 | 17.01.2003 | 00:08:00 | 0.90 | 782363 |
| "582577CIT" | 20.01.03 06:01:37 | 15.67 | 17.03.2003 | 15.64 | 17.01.2003 | 21:58:00 | 0.90 | 582577 |
| "782368COM" | 20.01.03 06:02:57 | | 17.03.2003 | 2.21 | 17.01.2003 | 19:33:00 | 1.05 | 782368 |
| "782369COM" | 20.01.03 06:03:00 | | 17.03.2003 | 0.53 | 17.01.2003 | 12:32:00 | 1.10 | 782369 |

All together almost 180 webpages are processed (5 strikes * 4 expiration dates * 3 issuers * 3 webpages = 180 extraction jobs). Because Deutsche Bank does not offer an option with the strike of 1.05 US-$ and expiration date June 2003, there are exactly 177 webpages to evaluate.

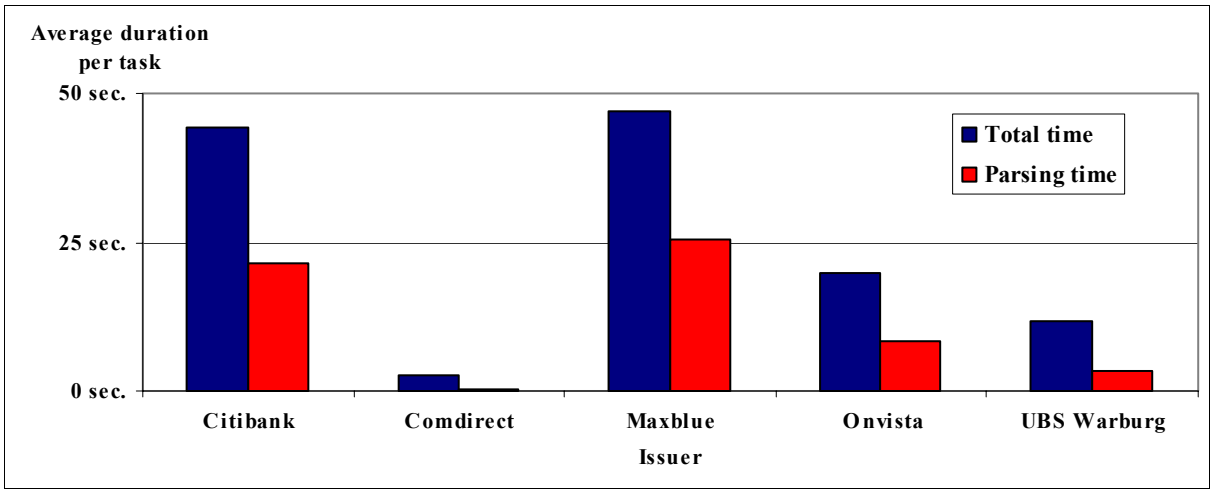
In both testing environments the jobs are initialized with a 1 second time delay. Shifting the jobs increases the likelihood for all requests to be processed correctly. The actual interval is changeable in the agent's configuration file.

On the 800 MHz machine all 177 tasks are executed on the same computer and the minimum job interval is set to 210 seconds. Otherwise the jobs affect each other because of the limited power. Using the two 2.4 GHz computers the jobs are split into 4 configuration files with

**Figure 8**

**Average task duration**

*The figure shows the average task duration for each processed website. The websites of Citibank and Maxblue are the slowest in both aspects, receiving and parsing webpages. These pages are the largest and have the longest response time. A webpage's size directly affects the needed parsing time. Both webserver response time and parsing time are very small at Comdirect. This page still can not be used for generating a dense time series because it's inadequate up-to-dateness. The other four pages are a capable source for extraction.*




about 45 jobs. Each file is executed by one of the overall four processors. Accordingly the execution interval is set 60 seconds first.

## C. Results

The extraction process works reliably. The wanted data are identified and stored correctly in the specified output files. The layout of the output files is shown in Table III.

**Performance**

Test analysis shows that the 800 MHz computer is not powerful enough to execute all 177 tasks within the specified interval with the given parameters. The duration time of one cycle in which all tasks are executed one time is much longer than the specified interval of 210 seconds. The interval has to be set to over 5 minutes to keep PISA working correctly or the number of extraction tasks must be reduced. In contradiction the Linux computers with each two 2.4 MHz processors work pretty well. All prices are extracted correctly and the surveillance files do not show any critical indication. For the given task PISA's performance is adequate. The execution interval can be even reduced.

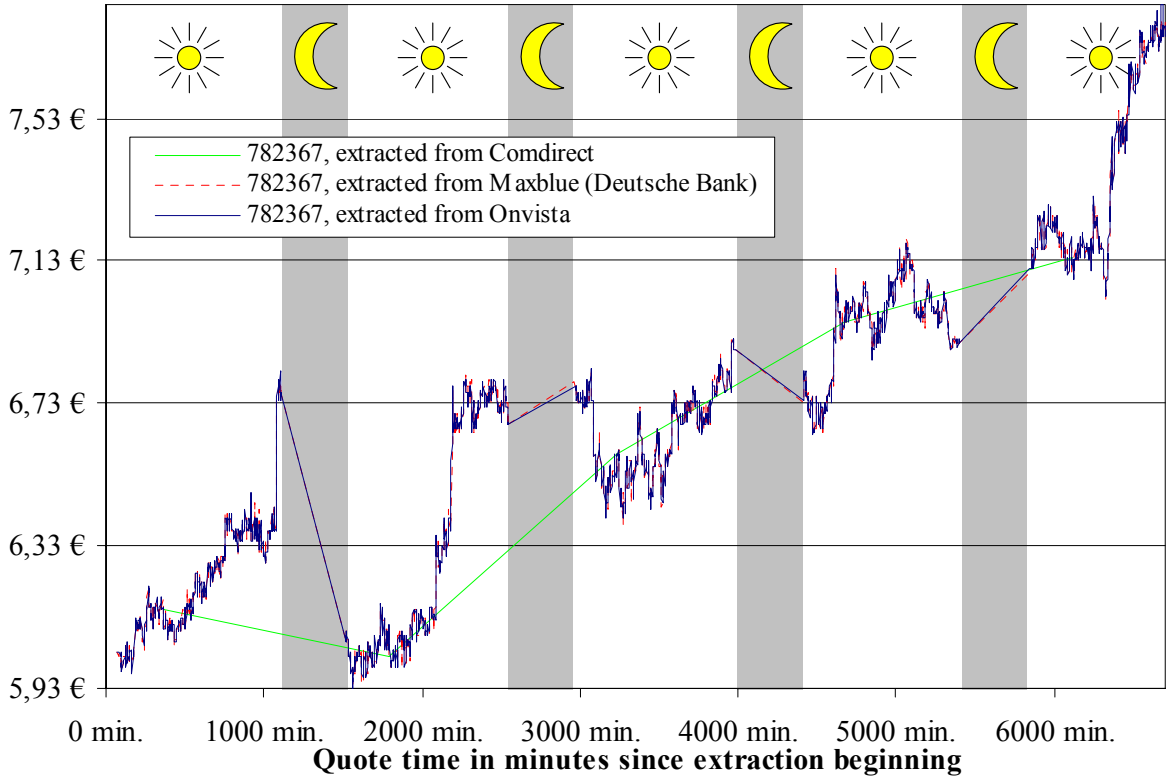
**Considering the extraction duration**

Considering the average extraction task duration a difference regarding the websites is noticeable. Figure 8 shows the total time and time needed for parsing a website. The picture only shows the average duration of the 2.4 GHz computers. The needed duration on the 800 MHz computer is hardly significant. Most of the duration is needed for waiting because the computer was running at its limits most of the time. The total time of Onvista and UBS Warburg is less the half of the time need for the webpages of Citibank or Maxblue. One reason is that the Citibank and Maxblue websites are larger than the others. This explains the higher parsing time. Second reason is that the webserver is operating at full capacity. Then the response time

**Figure 9**

**Extracted option price from three different webpages**

*The figure shows an option's time/price combination over the whole test period. The prices extracted from Onvista and the issuer's page, Deutsche Bank, are equal as expected. The price extracted from Comdirect is not. The price published by Comdirect is extracted correctly by PISA but is only updated seldom, here. This fact appears for all considered option prices extracted from Comdirect. The website's reliability and up-to-dateness are not adequate, here.*



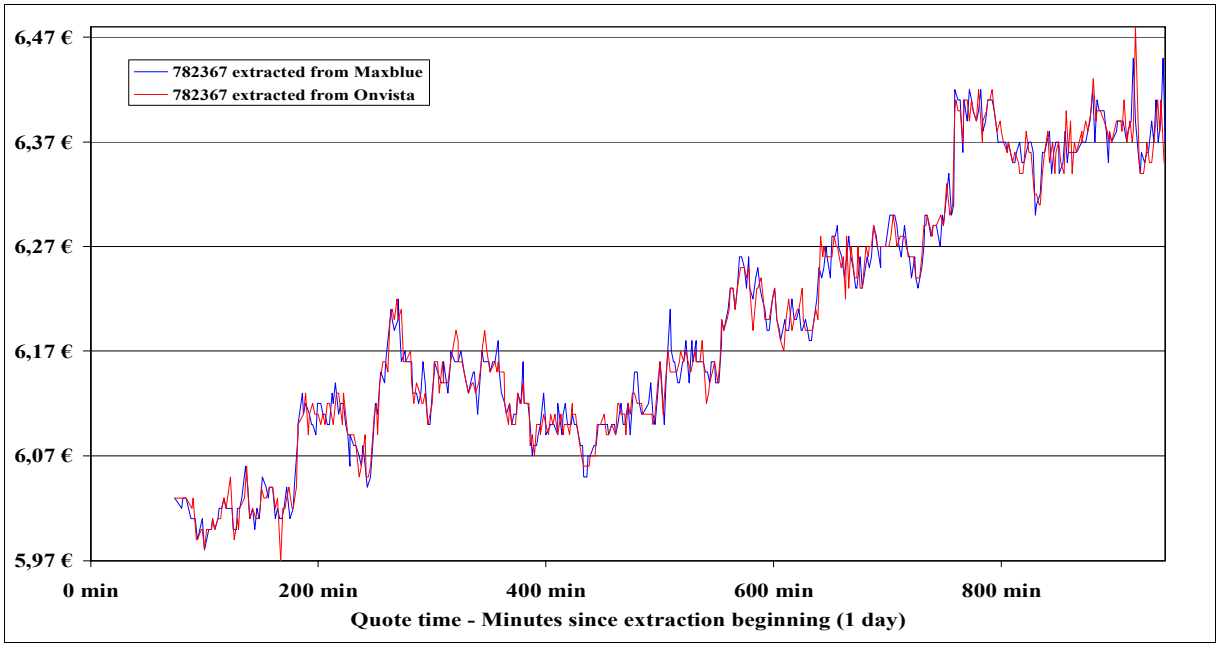

is higher. This thesis is confirmed by the fact that the needed time does not depend on the option but on the time of day. During European nighttime less time is needed for response. A correlation to the used processors is not possible as their speed is equal with 2.4 GHz.

**Reliable functioning**

Concerning reliability it is ascertained that PISA works well. All received webpages are processed correctly in both system environments. In some cases, all about less than 0.5 % of all tasks, PISA requested the website but did not get any response from the server, even if the page is requested multiple times. This does not cause a breakdown.

To validate correctness of the extracted data the saved source code can be examined. An indication for incorrect data is if an option's price charts from different webpages differ. If they are equal it can be assumed that the information is correct. Figure 9 shows the chart of the option with the German security identification number 782367 for the whole week. Comdirect updates the prices only once a day. The reason is that they do not publish prices given by the issuer but prices from the exchange. The update interval can not be predicted. An agent dysfunction can be excepted because PISA has successfully and correctly analyzed the website within the whole week. The price is extracted correctly and stored with the current extraction time and the time of the price which is published on the website. Nevertheless the static information is correct and their webserver is fast. Increasing PISA's efficiency this page can be

**Figure 10**
**Option's intraday price chart**
*The figure shows an intraday price time series of the option with the German security identification number 782367. The strike price 1.00 US-$ and expiration date March 2003. The values are extracted from Onvista and the issuer's page. The issuer named the website Maxblue instead of using their real name Deutsche Bank. The third source Comdirect is not pictured because it's inadequate up-to-dateness. Prices of Onvista usually equal the prices published by the issuer. Little differences are caused by marginal different extraction times.*




used for extracting the static information whereas other webpages are used as source of dynamic information like the price. Combining theses websites decreases processing time.
As expected, option prices are not updated at night. Referring to Figure 9 this is noticeable for the Maxblue and Onvista curves by the straight lines during nighttimes. This applies to all options and websites. The only difference is the exact trading period.
In case the data is correctly published and incorrect or no information is extracted, the only source of error left is the user specifying incorrect patterns. To check this PISA saves the source code of every received webpage. The user can look at the webpage's source code to approve this. Here, all patterns are correctly matched. Altogether, up-to-dateness of the tested websites, except Comdirect, is sufficient, here.
Figure 10 shows an example for a dense time/price combination generated by PISA. The warrant has the German security identification number 782367 and its price has been extracted from the issuer's page and Onvista. Analogous charts can be created for all options. The prices are equal at Onvista's and the related issuer's pages. Little differences base on the fact that the extraction has not been made exactly simultaneously but with marginal time shift. This results in different price/time-combinations. A second reason for not exactly identical charts is the different website updating interval. The smaller the interval is the denser are the resulting charts and the more they equal each other. Considering this analysis the Onvista website is a reliable source for extracting dense time series, here. The data extracted from issuer's page and redundantly from Onvista result in a time series with maximal 60 seconds between the price time combinations. The series are dense enough to be used for neural network training.

**Figure 11**
**Three option's prices from their issuer's page**

The figure shows an intraday chart of three options. The three options all have the same strike price of $ 1.00 and the same expiration date March 2003. The prices are extracted from their respective issuer's page. As expected the prices differ depending on the issuer. The difference is not constant but changes within the day.

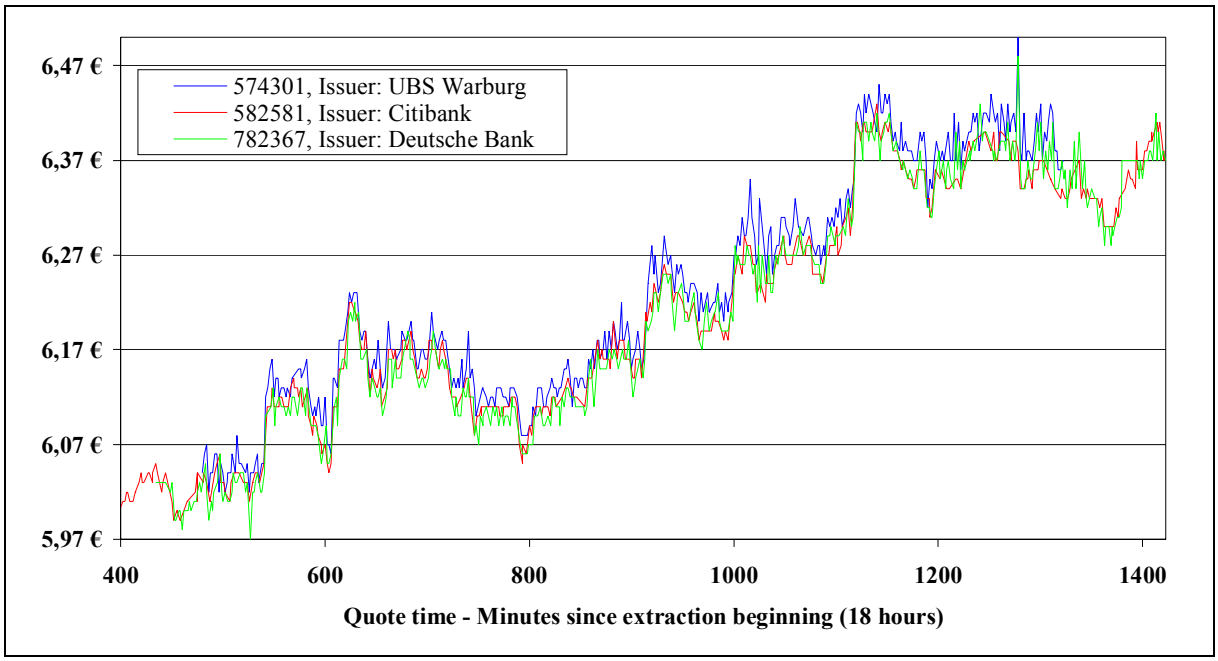



Figure 11 shows the intraday price of the Deutsche Bank option 782367 compared with the accordant options of Citibank and UBS Warburg. Each price has been taken from issuer's page to except wrong information of a third party provider. The chart pattern shows that PISA has extracted the data correctly. Apparently, as expected, prices for comparable options are not equal depending to the issuer. The reasons are differing assumptions in the pricing process. Neural networks interpolate the underlying time series and predict market prices on basis of given data. Considering the results the price difference depending on the issuer is not constant. It changes irregularly during the test period.

Important information is missing on some webpages. On the Citibank website only the current price with its according time is reliably provided. Static information like the expiration date and/or the strike price can not be extracted accurately. Analyzing the stored source code shows that these facts are not provided on the website. Nevertheless the prices are reliable and the site is usable for this task, except that some information has to be extracted elsewhere. Since these facts do not vary time-dependent this is not a problem.

All things considered not all websites are equally capable for the extraction job. There are differences in up-to-dateness and regarding contained information. The internet provides a lot of websites which offer free option prices. Using them to extract the needed information redundantly enables the agent to build a reliable time series for neural network training.

## V. Conclusions

Time series are used for many tasks. One important application area is neural networks. Neural Networks can generate mathematical functions that emulate relations between the input

variables. This is called supervised learning. Using these functions underlying time series can be either extrapolated or interpolated. Possible applications are forecast of future values and generation of derivative market prices. In both cases the underlying training data have to be dense and reliably correct. Usually expensive commercial databases are used. The Internet offers the needed data for free. Instead of a commercial database the needed data can be extracted from the Internet using the partial intelligent software agent PISA. Software agents are programs that work autonomously in the name of a user and execute specified tasks. Here, PISA extracts time series for neural network training from the internet. Compared to manually extraction autonomous software agents reduce work and costs. Quality and efficiency of the extraction process are increased.

PISA executes given extraction tasks in specified intervals defined in seconds, minutes, hoers, etc. The executing system's power and the number of execution tasks determine the smallest possible interval. To increase efficiency all tasks are executed simultaneously. Both aspects assure efficient usage of the computer's components and network's transfer rate. As a result a common desktop computer is sufficient to execute PISA. The agent can be used in most current offices. Only for large extraction jobs a computer with two or more processors is recommended to keep extraction intervals small. Alternatively several computers can be used simultaneously. The extraction tasks can be partitioned. The tasks are started and terminated timer controlled. In most neural network environments both Windows and UNIX computers are used. To avoid a restriction to only one operating system PISA supports both. To accomplish the mentioned demands PISA is completely realized in Java.

Redundant extraction of information from several webpages prevents data gaps. Extracted data can be stored in text files and databases using the ODBC interface. The text files are well structured and can be easily used by neuro simulators. Databases have the advantage that data can be further calculated, e. g. to filter inaccurate values or to merge different data sources. In addition results can be saved in XML files. XML files are platform independent and can be imported in almost every spread sheet program. Most financial analysis programs support data import for XML files.

PISA works reliably and correctly. The redundant extraction process assures that the resulting database is correct, with few exceptions, dense and has no data gaps usually. This has been proved by several long term tests. PISA can work continuously and autonomously for many weeks without user interference.

Legal restrictions constrain the usage of PISA for commercial aspects. Most website carriers indicate that any commercial use of the published information is forbidden. For non commercial use webmining tools are allowed. It should nevertheless be considered that frequent webpage requests increase webserver utilization. For the webserver carrier this results in increasing hardware costs. Agents like PISA should only be used in a way that no web service provider is disadvantaged.

## References

Anderson, J. A., 1997. *An Introduction to Neural Networks* (MIT Press, Cambridge).

Azoff, E. M., 1996. *Neural network time series forecasting of financial markets* (Wiley, New York).

Bartels, Patrick, 2002, *Konzeption und Entwicklung eines intelligenten Agenten zum Internet Content Mining*, (Diplomarbeit, Universität Hannover).

Black, F. and M. Scholes, 1973, The Pricing of Options and Corporate Liabilities, *Journal of Political Economy* 81, 637 – 659.

Bloch, Joshua, 2001. *Effective Java Programming Language Guide* (Addison-Wesley, Boston).

Bradshaw, Jeffrey M., 1997, An Introduction to Software Agents, in Jeffrey M. Bradshaw, ed.: *Software Agents* (Menlo Park, Cambridge and London).

Breitner, Michael H., 1999, Heuristic Option Pricing with Neural Networks and the Neuro-Computer Synapse 3, *Optimization* 81, 319 – 333.

Breitner, Michael H., 2001. *Nichtlineare, multivariate Approximation mit Prezeptrons und anderen Funktionen auf verschiedenen Hochleistungsrechnern*, (Habilitationsschrift, TU Clausthal, Clausthal-Zellerfeld).

Breitner, Michael H. and M. Ambrosius, 1999, SQP-training of Perceptrons with the Neurocomputer SYNAPSE 3 Applied to Fast Empiric Option Prices, in L. J. Cromme, ed.: *Proceedings of the CoWAN '98*, 65 - 84 (Shaker, Aachen).

Breitner, Michael H. and S. Bartelsen, 1997, Optimal Portfolio Management through Forecasting of Stronger Stock Price Changes with the Neurocomputer SYNAPSE, *TU Clausthal-Mathematik-Bericht* 11, 15 p.

Breitner, Michael H. and S. Bartelsen, 1999, Training of Large Three-layer Perceptrons with the Neurocomputer SYNAPSE, in P. Kall, ed.: *Operations Research '98*, 562 – 570 (Springer, Berlin).

Caglayan, Alper K. and Colin G. Harrison, 1997. *Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents* (John Wiley & Sons, New York).

Chng, B., 2002, *Matchmaking With Regular Expressions*, comp. http://www.javaworld.com/javaworld/jw-07-2001/jw-0713-regex.html, downloaded 05/23/03.

Cox, John C. and Stephen A. Ross, 1976, A Survey of Some New Results in Financial Option Pricing Theory, *Journal of Finance* 31, 2, 383 - 402.

Eikvil, Line, 1999, *Information Extraction from the World Wide Web*, Report No. 495, Norwegian Computing Centre.

Flanagan, David, 2001. *JavaScript: The Definitive Guide* (O'Reilley & Associates, Sebastopol).

Franklin, Stan and Art Graesser, 1996, *Is It An Agent Or Just A Program – A Taxonomy Of Autonomous Agents*, comp. http://www.dbgroup.unimo.it/IIA/references/paper0.html, downloaded 05/23/03.

Friedl, Jeffrey, 2002, *Mastering Regular Expressions* (O'Reilly & Associates, Sebastopol).

Gallant, S. I., 1995. *Neural Network Learning and Expert Systems* (MIT Press, Cambridge).

Gomer, Peter and Miroslav Budimir, 2000, Agentenbasierter Rentenhandel, *Wirtschaftsinformatik* 42, 124 – 131.

Hecht-Nielsen, Robert, 1994. *Neurocomputing* (Addison-Wesley, Reading).

Hertz, J., Anders Krogh and Richard Palmer, 1996. *Introduction to the Theory of Neural Computation* (Addison-Wesley, Reading).

Hull, John C., 1999. *Options, Futures and other Derivatives* (Prentice Hall, Upper Saddle River, NJ.).

Hutchinson, James M., Andrew W. Lo and Tomaso Poggio, 1994, A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks, *Journal of Finance* 49, 3, 851 - 889.

Jones, Tommie, 2002, *Webmining with Perl*, comp. http://www.devshed.com/Server_Side/Perl/DataMining/DataMining.pdf, downloaded 05/23/03.

Meas, Pattie, 1994, Agents that reduce work and information overload, *Communications of the ACM* 37, 30 – 40.

Merton, Robert C., 1976, The Impact on Option Pricing of Specification Error in the Underlying Stock Price Return, *Journal of Finance* 31, 2, 333 - 350.

Murch, R. and T. Johnson, 1998. *Intelligent Software Agents* (Prentice Hall PTR, Upper Saddle River).

Nourie, Dana and Mike McCloskey, 2002, *Regular Expressions and the Java Programming Language*. Comp. http://developer.java.sun.com/developer/technicalArticles/releases/1.4regex/, downloaded 05/23/03.

Nwana, Hyacinth S. and Michael Woodridge, 1997, Software Agent Technology, in Nwana, Hyacinth S. and Nader Azarmi, eds.: *Software Agents And Soft Computing* (Springer, Berlin and Heidelberg).

Petsch, M., 2001, Aktuelle Entwicklungsumgebungen für mobile Agenten und Multiagentensysteme, *Wirtschaftsinformatik* 2, 175 – 182.

Reames, Randy, 2002, *What is Perl?*, comp. http://www.sysadminmag.com/tpj/whatisperl.html, downloaded 05/23/03.

Refenes, A.-P., 1996. *Neural Networks in Capital Markets* (Wiley, New York).

Schwartz, Randal and Tom Christiansen, 1997. *Learning Perl* (O'Reilly & Associates, Sebastopol).

Stroustrup, Bjarne, 2000, *The C++ Programming Language* (Addison-Wesley, Boston).

Vemuri, V. R. and R. D. Rogers, 1994. *Artificial Neural Networks: Forecasting Time Series* (Institut of Electrical and Electronic Engineers (IEEE) Computer Society Press, Los Alamitos).

Welling, Luke and Laura Thomson, 2003. *PHP and MySQL Web Development* (Sams publishing, Indianapolis).

White, H., A. R. Gallant, K. Hornik, M. Stinchcombe and J. Wooldrige, 1992. *Artificial neural networks: approximation and learning theory* (Blackwell, Oxford).

Woodridge, Michael and Jennings, Nicholas, 1995, Intelligent Agents: Theory and Practice, *Knowledge Engineering Review* 10, 115 – 152.

Zirilli, J. S., 1997. *Financial Prediction Using Neural Networks* (International Thomson Computer Press, London).

# IWI Discussion Paper Series

ISSN 1612-3646

Michael H. Breitner, *Rufus Isaacs and the Early Years of Differential Games*, 36 p., # 1, January 22, 2003.

Gabriela Hoppe, Michael H. Breitner, *Classification and Sustainability Analysis of e-Learning Applications*, 26 p., # 2, February 13, 2003.

Tobias Brüggemann, Michael H. Breitner, *Preisvergleichsdienste: Alternative Konzepte und Geschäftsmodelle*, 22 p., # 3, February 14, 2003.

Patrick Bartels, Michael H. Breitner, *Automatic Extraction of Derivative Prices from Webpages using a Software Agent*, 32 p., #4, May 20, 2003.