StockWatcher
A Semantic Web Application for Custom Selection of Financial News

Laurens Mast, Alex Micu, Flavius Frasincar, Viorel Milea* and Uzay Kaymak

Econometric Institute, Erasmus University Rotterdam,
3062 PA Rotterdam, Netherlands
**E-mail: milea@few.eur.nl*
www.few.eur.nl

In this paper we present StockWatcher, an OWL-based application that enables the extraction of relevant news items from RSS feeds concerning financial markets. The application's goal is to present a customized, aggregated view of the news categorized by different topics and at the same time rate these news items based on their relevance. The selection of the relevant news items is based on a customizable user portfolio.

Keywords: Semantic Web, financial news, RSS, OWL, stocks

1. Introduction

Unlike printed media or television programs, on the Web news can be made public as soon as it emerges. Simultaneously, Web coverage is continually increasing. News websites provide RSS-feeds facilitating the public to remain up-to-date on nearly any topic of interest.

One of the domains where access to information, and implicitly news, plays a crucial role is represented by financial markets. With the introduction of new products such as click funds,[1] the level of involvement of the general public in investment activities is on the rise. This increased involvement comes to underline the need for access to mediums which can provide relevant and reliable economical news within short time intervals. The Web comes to meet this need, while at the same time confronting users with an overwhelming amount of information. Questions such as *'Where does news appear faster?'* or *'Which news websites are trustworthy?'* have already risen.

With the emergence of the Semantic Web, languages such as OWL[2] and RDF(S)[3,4] help provide the basis towards speeding up this process. The goal pursued in this paper is related to this, and consists of creating an application that helps casual internet users with a relevant involvement in financial markets to find relevant news regarding their portfolio. This effort has resulted in StockWatcher, an application that enables the presentation of a customized, aggregated view of news items categorized by different topics, and at the same time rates these news items based on their relevance.

After this introductory section, we focus on related projects in Section 2. Next, we present our application in Section 3, where we focus on the architecture and user interaction with the website. A more visual presentation of the application and the generated results is given in Section 4. Finally, we conclude in Section 5 where we also provide some ideas for further research.

2. Background

In this section we provide a brief overview of projects related to our current goal. Due to the practical nature of the research, the focus is on related applications and technologies.

2.1. *Related Projects*

The Artequakt[5] project is probably one of the most well-known Semantic Web project currently in development. One of the main factors that contributed to this popularity relates to the symbiosis between the application and the Semantic Web technologies. Artequakts goal is to find information on the internet about artists and paintings, from different sources, bring that information together, and present it to different users. The main point of interest is the information extraction phase. At this stage, Artequakt actively searches for data on the web to fill up the knowledge base. For this purpose GATE[6] is employed, a natural language engineering framework.[7] However the data needed by StockWatcher is available on websites that already provide extensive metadata about the provided information, thus deeming

the use of HTML wrappers sufficient in the case of StockWatcher.

Another application designed for similar purposes as StockWatcher is Market News Analyzer (MNA).[8] Despite its name, the application does not analyze the news, but only extracts the information from various RSS-feeds, such as Yahoo Finance, and presents specific news segments to the user. It features a large range of companies, from which one can select to receive news, and also plenty of statistical data about the amount of news found, sorted by different indicators.

However, not all systems employ a Semantic Web approach. Examples of such systems are: NewsPiper[9] and SpeedResearch Stock Market Browser.[10] Not making use of the benefits provided by the Semantic Web technology could become a downside for these applications in the near future. For example, in contrast to StockWatchers OWL data representation, these systems offer no possibility to share the contained data with other applications.

3. StockWatcher

StockWatcher is a web based application that allows users compose retrieve news from (custom) RSS feeds that are relevant to their own portfolio. Currently, the focus is on companies active on Nasdaq, allowing the user to compose his/her portfolio from the Nasdaq-100 index,[11] where heavy hitters such as Microsoft, Dell or Google can be found. The choice to restrict the application to these companies is motivated by the requirement of remaining synoptic, a benefit for the layout of the application. However, extensions based on the current system are relatively easy to design, e.g. an extension to include more companies. A customizable HTML-wrapper for Hoovers.com is used for the extraction of all the information on the Nasdaq-100 companies. This website is specialized in giving information for more or less 40.000 public and non-public companies.

The ontology used by the system is created in OWL, with the use of Protégé.[12] By using OWL we are able to facilitate greater machine interpretability than by employing other techniques, e.g., XML or RDF. The ontology itself is also unique as currently no ontologies with such a focus are available.

StockWatcher uses a Microsoft Access database to store the relevant information. The primary factor for choosing Access is its easy to use graphical interface. Although it falls short of being a fully object-oriented development tool, it does provide the required functionality for StockWatcher. The SQL commands are executed from a Java module, connected to the database through a standard ODBC driver.

Having briefly outlined the StockWatcher application in the previous paragraphs, we focus in the following on the details regarding the architecture of the system and the user interface in the following sections. We describe the main components that make up StockWatcher and focus on the particular characteristics that render this application as unique.

3.1. *Architecture*

An overview of the system is depicted in Figure 1. As shown, the conceptual model (CM) has been separated in three parts. Part A of the model, *data extraction*, is responsible for extracting information concerning certain companies, and storing it in the local Access database. Once the database has been populated, users can compose their own portfolio from the companies in the database. By creating the portfolio, an ontology corresponding to this portfolio is automatically generated. All this takes place in part B, *ontology creation*. In part C, *news searching*, the application searches various news feeds for relevant news based on the custom portfolio ontology. Across the next three sections we give a more comprehensive description of each of the three parts outlined in this paragraph.

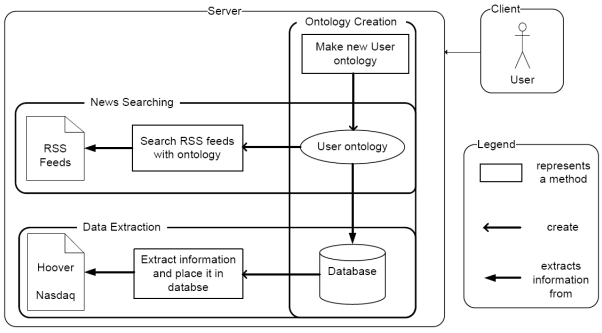

Fig. 1. The StockWatcher Architecture.

3.1.1. *Data Extraction*

The first step of using the StockWatcher system is related to information extraction. Initially, the infor-

mation was extracted realtime from Nasdaq-100 each time a user logged on. Due to performance issues consisting of loading time and uptime for the various websites used to extract information, it seems faster and safer to extract the information once, and place it in the local database. We registered improvements in speed from 2 up to 6 seconds.

The module *Nasdaq2Database* is responsible for the information extraction and data storage. This module provides the functionality to extract the company trade name from Nasdaq-100. This information along with the company full name is stored in the database. Additional functionality relates to extracting the rest of the information from Hoovers.com, by using the ID number corresponding to the trade name on Nasdaq-100. This information contains the key people in the company, the competitors and the industry it is active in.

3.1.2. *Ontology Creation*

In identifying the concepts involved in the StockWatcher application, we concluded that the appropriate starting point is the ontology. There are a lot of tools available for this purpose, such as SemanticWorks,[13] SWOOP[14] or TopBraid Composer.[15] The selection eventually fell on Protégé OWL, for the obvious advantage of its simplicity and widespread popularity.

One of the main classes represented in the financial StockWatcher ontology is the *Company* class. Additionally, a differentiation needs to be made between companies in the user's portfolio and their competitors. This spawns 2 subclasses of the main class *Company*. In addition we identified an *Industry* class, serving as a pool for all companies. This enables the grouping of certain companies within a specific industry, making it easier to recognize competitors when using SPARQL[16] queries. Finally, the class *Person* denotes the important people in a certain company.

In order to enable reasoning on the created ontologies, a Description Logic Implementers Group (DIG) compliant reasoner had to be installed next to Protégé. Our choice fell on RacerPro,[17] for various reasons. The most important one relates to this reasoners' popularity in combination with Protégé, and the easy way of configuring procedures. Protégé offers different tests that can be applied on the ontology. Most important of them was Classify Taxonomy. By running this test, the reasoner checks if the classes and subclasses are built correctly. Matters concerning disjoints and conditions are being tested at this stage.

After the extracted data from Nasdaq-100 and Hoovers.com is available in the database, StockWatcher is ready for live use. As soon as a user logs on to the website and selects his/her portfolio, part B of the application is activated. Making use of different techniques facilitated by the Jena framework,[18] StockWatcher is able to manipulate the financial ontology, which serves as a frame.

3.1.3. *News Searching*

As soon as the ontology is complete and the RSS feeds are retrieved, the search for relevant news is initialized. In the early stages of the development of the application, the search engine only considered whether the title or description of the news items on the RSS feeds enclosed any words which appeared in the ontology, such as Google. Eventually this did not prove to be a good method. Partial matches or very common words caused many irrelevant news items to be selected. The search algorithm required refinement. Common words such as 'systems' and 'incorporated' were filtered so they would not cause any additional mismatches. Additionally, the minimal length for words has been adjusted to three characters for the same reason. All words have been split up and only counted as a match when they were exactly the same. Partial matches such as 'Dell' in 'modelling' didn't count as a match.

This add-on improved the search results, but only marginally. Even if almost every word searched for was relevant, irrelevant matches still appeared in the results. Imagine the following scenario: searching for news about 'Adobe' the application comes across a news item called *'Similar to companies such as Corel, Microsoft and Adobe. But we are talking about...'*. To fix this problem a score system has been implemented. A match in the title scores 2 points, matches in the description only 1. A news item has to have at least 2 points to appear in the results. Now the application is able to rank the news items, by counting the scored points. By ignoring the rest of the news, i.e., the ones that scored lower than 2 points, the relevance of the results improved sig-

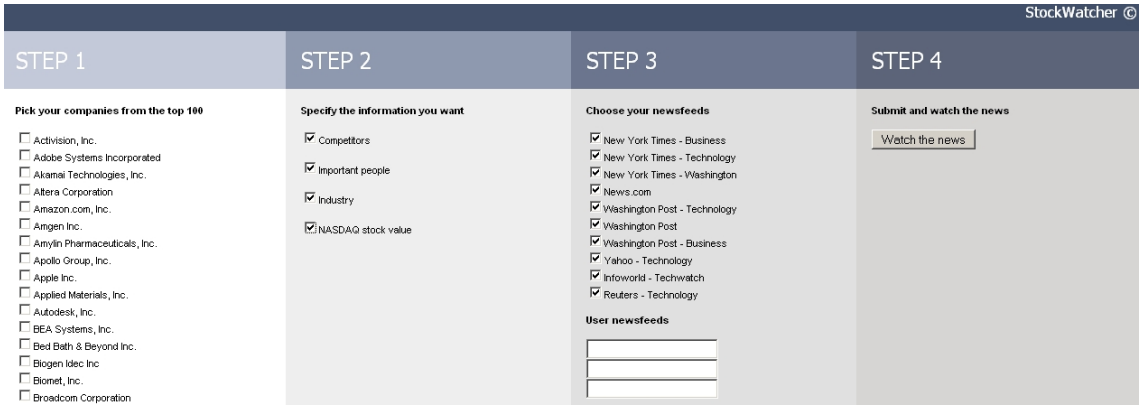

Fig. 2. StockWatcher: User Input.

nificantly. The major improvement is especially noticeable with well-known companies such Dell and Google. These brand names are often used as reference or example in articles with little or no relevance to the company itself.

Furthermore, the score system can be used as framework for further improvements. It is possible to include more sophisticated search algorithms which can further improve the relevance of the returned news items. Examples of such approaches include calculating the distance between certain words,[19] or trying to find the meaning of the words in the text.[20]

To search the RSS feeds we use Informa[21] and SPARQL.[16] Informa is a Java framework that brings together the best of two other Java news reader applications, namely HotSheet[22] and Risotto.[23] The main feature of Informa consists of being able to retrieve almost any news feed available on the Internet. Above that, it offers good documentation making it easy to use in applications. SPARQL is a query language and data access protocol for the Semantic Web. Its main use is to extract information from (RDF(S) or OWL) ontologies.

3.2. *User interface*

One of the most important aspects in creating the interface for a website is user friendliness. Factors like usability, design, consistency, navigation and simplicity play a great role in deciding how user-friendly and effective a website is.? With this in mind we designed a user interface focused on simplicity and efficiency.

StockWatcher gives the opportunity to be used with or without making an account: site visitors can choose between logging in or skip this step and directly set-up a portfolio. The disadvantage of the second choice relates to the fact that the system will not remember the user's preferences concerning the companies they were interested in. If an account is created, the portfolio is stored in the database, and can be reloaded each time the system is accessed through the login interface. Once the user has entered the website, either way, a menu consisting of four steps will appear. This is reproduced in Figure 2.

Step 1 shows the companies from Nasdaq-100. Here the user can set-up his/her portfolio. Existing users can add new companies to their portfolio, or remove existing ones, an action that will also change the user's profile in the database. In Step 2 the user can select which information they are interested in: *competitors* of the selected companies, *important people* engaged to a relevant degree in the activities of the companies in the user's portfolio, news regarding the *industry* the companies are active in, and the *NASDAQ stock value*, providing numerical data on the current performance of the selected companies. The first three options have direct impact on the ontology, while the fourth only relates to a real-time quantitative measure of how the companies are currently performing. Finally, in Step 3, the users are given the possibility to choose the RSS feeds from which data should be retrieved. Also, users are allowed to customize the standard list of feeds by adding additional ones for the retrieval of news. StockWatcher supports most of the economical related news feeds, although a scenario where users make use of smaller, less popular RSS feeds is possible.

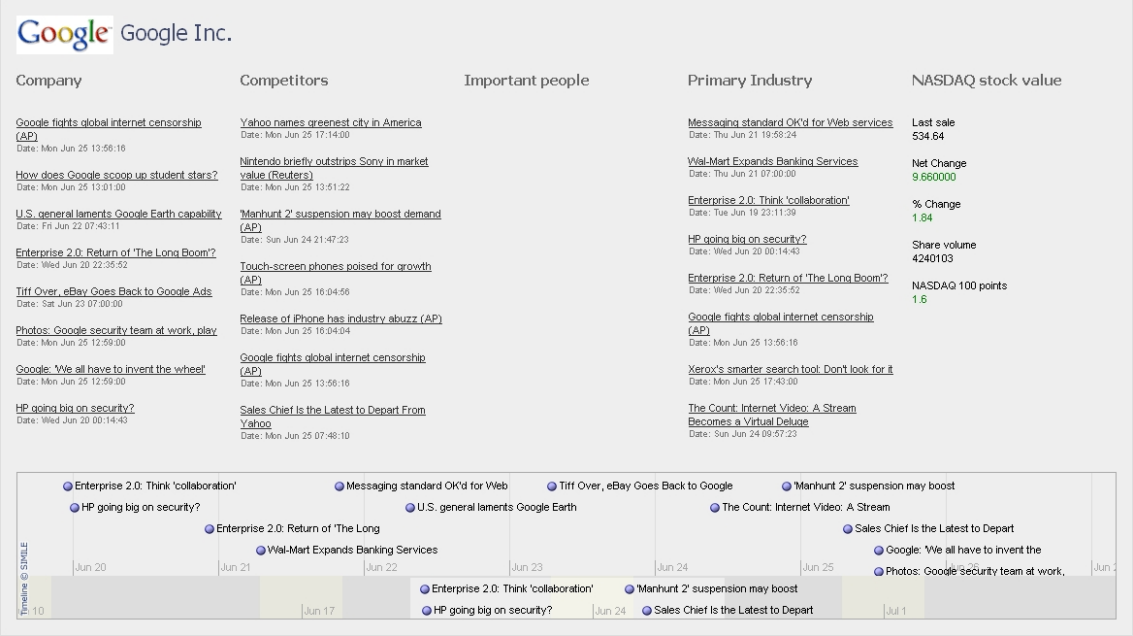

Fig. 3. StockWatcher: Output.

4. Results preview

For the presentation of the results, StockWatcher makes use of an application called Timeline,[24] developed within the SIMILE project.[25] This visualization tool has been created to properly illustrate time-based events. Timeline does not require to be installed, neither server-side nor client-side, and the events used to fill it can be easily stored in an XML file. Figure 3 gives an overview of the system's output when the portfolio consists of only one company, namely Google. The first column represents news items generated by searching the feeds for the company name. Next to this column news concerning competitors of the company are presented, followed by news regarding important people. The last two columns give an overview of the news messages regarding the industry in which the company is active and the latest stock value of that particular company, respectively. The news items are sorted according to the received rating.

The additional Timeline bar adds a temporal dimension to the selected news items by presenting them in the order in which they emerged. This bar gives users the ability to easily spot the latest news they are concerned with and possible time-determined interactions between these news items. However, the Timeline could be the building stone for additional functionality, such as combining it with share prices in order to track the correlation between the emerging news and their corresponding impact on the value of stocks.

5. Conclusions and Further Research

The main focus of this paper is on presenting the StockWatcher application. The previous chapters provide a description of its architecture, and offer details about the programming tools that were used, such as the Jena framework and Informa. By attaching a rating system to the proposed algorithms we were able to rate the news on their reliability and relevance. This resulted in a web application that can display trustworthy news items regarding the user's portfolio, on subjects considered relevant by the user.

The main goal of the OWL Web Ontology Language relates to making web data machine-understandable. With StockWatcher we have laid a strong basis for future development in this direction. Thanks to the ontology-driven character of the application and the employed techniques (OWL, Jena, SPARQL), we have created a foundation for the analysis of news items in a larger context. With this analysis it will become possible to provide automated advices on the impact of certain news items on share prices. Other tools, such as Timeline, add to the

power of the application by placing predictions in a temporal context, thus enabling the tracking of the exact impact of news items on stock values in an intuitive, visual environment.

One point of improvement relates to the data extraction regarding the relevant companies. Currently this is done by retrieving all the data from on-line Hoovers profiles. As extracting data from an HTML page takes time this is not done at the moment the user searches. Instead, the entire database is periodically updated with new Hoovers data. It is possible to further enhance the application by using an up-to-date database instead of Hoovers to retrieve company-related information. This database should contain all the companies in the top 100 and offer information about their competitors, important people and market. By using a database like this, the application will always use the most up-to-date information and would thus no longer need a regular update.

A final point is related to extending the current application to include more companies than just the NASDAQ-100 index currently supported. Further customization could include other indexes, such as DAX30, CAC40, AEX, etc., and allow the user to insert additional financial entities that are of interest.

Acknowledgement

The authors are partially supported by the EU funded IST STReP Project FP6 - 26896: Time-determined ontology-based information system for realtime stock market analysis. More information is available on the official website[a] of the project.

References

1. F. van Mulligen, Zo werkt een klikfonds (2002), `http://www.morningstar.nl/`.
2. M. Smith, C. Welty and D. McGuinness, OWL Web Ontology Language Guide, W3C Recommendation (2004).
3. G. Klyne and J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation (2004).
4. D. Brickley and R. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation (2004), `http://www.w3.org/TR/rdf-schema/`.
5. S. Kim, H. Alani, W. Hall, P. Lewis, D. Millard, N. Shadbolt and M. Weal, *Semantic Authoring, Annotation and Knowledge Markup (SAAKM) 2002 Workshop at the 15th European Conference on Artificial Intelligence (ECAI 2002), Lyon, France* (2002).
6. H. Cunningham, D. Maynard, K. Bontcheva and V. Tablan, GATE: A framework and graphical development environment for robust NLP tools and applications, in *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.
7. G. G. Chowdhury, *Annual Review of Information Science and Technology* **37**, 51 (2003).
8. Franz AG company, Market news analyzer (2007), `http://www.marketnewsanalyzer.com/`.
9. S. Korzh, Newspiper (2007), `http://www.korzh.com/`.
10. R. M. Kucynski, Speed research stock market browser (2007), `http://www.speedresearch.com/stockwatch/`.
11. NASDAQ-100 (2007), `http://quotes.nasdaq.com/quote.dll?page=nasdaq100/`.
12. Stanford Medical Informatics, Protégé-owl (2007), `http://protege.stanford.edu/`.
13. Altova, Semanticworks (2007), `http://www.marketnewsanalyzer.com/`.
14. University of Maryland College Park, Swoop (2007), `http://code.google.com/p/swoop/`.
15. TopQuadrant, Topbraid composer (2007).
16. E. Prudhommeaux and A. Seaborne, *World Wide Web Consortium* (2004).
17. Racer Systems GmbH & Co. KG, Racerpro (2007), `http://www.racer-systems.com/index.phtml/`.
18. Hewlett-Packard Development Company, LP, Jena (2007), http://jena.sourceforge.net/.
19. S. Kruk, S. Decker and L. Zieborak, *DEXA Conference* (2005).
20. H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis and N. R. Shadbolt, *IEEE Intelligent Systems* **18**, 14 (2003).
21. Informa Project, Informa (2007), `http://informa.sourceforge.net/`.
22. D. Thorp and J. Munsch, Hotsheet (2007), `http://sourceforge.net/projects/hotsheet/`.
23. A. C. Kramer and N. Schmuck, Risotto (2007), `http://sourceforge.net/projects/jsurfer/`.
24. SIMILE project, Timeline (2007), `http://simile.mit.edu/timeline/`.
25. Massachusetts Institute of Technology, SIMILE Project (2007), `http://simile.mit.edu/`.

[a] `http://www.towl.org`