WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF MATHEMATICS
AND INFORMATION SCIENCE

MASTER THESIS
COMPUTER SCIENCE

Towards Semantic Web

Author: Maciej Obojski

Supervisor: Ph. D Marcin Paprzycki

WARSAW APRIL 2008

…………………………   ………………………

podpis promotora   podpis autora

Table of Contents

Abstract

This work presents approach that is to become a step towards realization of the vision of the Semantic Web. The main goal was to create mechanism which would allow finding specific information on the Internet and utilizing it to complete missing information in instances of ontologically demarcated objects. Application developed for the purpose of this thesis is a part of the Travel Support System (TSS).

The TSS is an approach to fully use benefits of the Semantic Web – a Web where machines are capable of analyzing and semantically processing data. Application and the TSS take advantage of the JADE platform, which is used to develop software agents. The TSS delivers user-specific data, based on personal preferences. Information about restaurants and hotels are stored in the RDF format, which contains data and metadata as well.

For the needs of the task, several approaches were taken into consideration (neural networks, regular expressions, logic wrappers and machine learning). In the thesis, the DATAPROG algorithm, created by Craig Knoblock, was adapted and modified. The algorithm is a machine learning approach. Purpose of DATAPROG is to find statistically significant tokens in a set of examples, and to identify pattern, by which those examples were created. After the DATAPROG finds patterns, then those are stored in a file.

TSS is an agent-based system which consists of several subsystems. This thesis is focused on the Content Collection Subsystem, where main role is played by Coordinator Agent (CA). This agent has been originally equipped in *receiver* and *scheduler* behaviours. In order to integrate the DATAPROG algorithm with existing functionality, CA's *receiver* behaviour has been added new action: *FindMissingTokensAction.* This action starts running whenever there is need to find missing data. Algorithm queries Google for top ten results. For each page in results, the code logic removes HTML tags. Such obtained document is traversed and patterns identified by DATAPROG algorithm are stored in *HashTable* object. Algorithm returns the most often found answer to the query.

Algorithm was tested in practice, and results are presented and discussed.

Streszczenie

Niniejsza praca prezentuje ideę oraz oprogramowanie, które przybliży wizję Sieci Semantycznej. Głównym celem było stworzenie mechanizmu pozwalającego na znajdowanie specyficznej informacji w Internecie, oraz wykorzystanie jej do uzupełnienia brakujących danych w instancjach ontologicznie opisanych obiektów. Aplikacja stworzona na potrzeby tej pracy jest częścią Systemu Wspomagania Podróży (TSS od angielskiej nazwy – Travel Support System).

System TSS jest próbą wykorzystania wszystkich korzyści płynących z zastosowania Sieci Semantycznej – Sieci w której komputery są w stanie zrozumieć, zanalizować i semantycznie przetworzyć dane. Aplikacja oraz TSS używają platformy JADE, skonstruowanej do tworzenia oraz zarządzania agentami. TSS dostarcza spersonalizowanych

danych, bazując na indywidualnych preferencjach użytkowników. Informację o restauracjach oraz hotelach są trzymane w formacie RDF, który zawiera dane oraz metadane (dane o danych).

Na potrzeby zadania zostało rozpatrzonych kilka rozwiązań: sieci neuronowe, wyrażenia regularne, wrappery logiczne oraz uczenie maszynowe. W pracy użyty został algorytm DATAPROG, opracowany przez Craiga Knoblocka. Algorytm został zmodyfikowany na potrzeby zadania, jest on przykładem rozwiązania nazywanego uczeniem maszynowym. Celem algorytm jest znalezienie statystycznie znaczących ciągów liter w zbiorze przykładów, oraz zidentyfikowanie wzorca za pomocą którego dane przykłady zostały stworzone. Po pomyślnym znalezieniu wzorca (lub wzorców), algorytm DATAPROG zapisuje wyniki w pliku.

TSS jest systemem wielo agentowym, który składa się z kilku podsystemów. Niniejsza praca skupia się na Podsystemie Wyszukiwania Danych (Content Collection Subsystem), gdzie główne zadania są obsługiwanie przez agenta nazwanego: Coordinator Agent (CA). Ten agent został domyślnie wyposażony w dwa zachowania: *receiver* (odbiorca) oraz *scheduler* (planowanie). W celu zintegrowania algorytmu DATAPROG z istniejącą funkcjonalnością, CA, zachowanie *receiver* zostało wzbogacone o nową akcję: *FindMissingTokensAction.* Ta akcja rozpoczyna działania gdy tylko zajdzie potrzeba odnalezienia brakujących danych. Algorytm odpytuje Google i przetwarza 10 pierwszych wyników zapytania. Dla każdej zwróconej strony internetowej, usuwane są znaczniki HTML. Tak spreparowany dokument jest przeszukiwany celem znalezienia wzorców zidentyfikowanych przez DATAPROG. Algorytm zwraca najczęściej powtarzający się wynik.

Algorytm został przetestowany w praktyce. Wyniki są opisane i skomentowane.

1. Travel Support System

The Semantic Web

"I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize."

Tim Berners-Lee

Tim Berners-Lee is the inventor of the World Wide Web. During early stages of development of the Hyper Text Markup Language (HTML) he put forward a proposal that the evolution of the network (Internet) should be devoted to understanding information by machines. This vision of machines being able to read and understand data throughout the Internet is called the *Semantic Web.*

Unfortunately the Web evolved not as Tim Berners-Lee planned it. Nowadays documents located in the Internet are mainly focused for eye-catching effects, and the really important data is mixed with constructs responsible for layout and graphic aspects of the web page. HTML interpreters built-in into popular browsers can properly render graphical information described by various tags. But none of those browsers is capable of telling what does the document concern.

There exist special <META> tags by which page creator can instruct search engines or web crawlers about the content of the page. Nevertheless it happens very often that creators omit this tag. Even if it is specified – it is only a guide telling what the document contains – if one is interested in more accurate data then those tags are useless. The problem lies in the definition of the HTML – programmer can easily describe layout of the page, but high level relations that describe the data in the document, cannot be defined. Tim Berners-Lee has a vision that tags defining layout and the data itself are two separate things; what is more – data are structured and described by series of relations that allow writing appropriate mechanisms that can automatically join various data placed in Internet.

Semantic Web is a great tool that can help us deal with problems that normal users come across almost every day:

- information overload

  Internet is a mess – finding valuable and reliable information is a task which begins to get harder and harder. By describing the data ontologically people would be able to find exactly what they were looking for

- poor content aggregation

  Most of programming projects focused on gathering data from the web are using technology called *screen scrapping*. It means that a special program called Wrapper has hard-coded structure of HTML document and ‘knows’ exactly where to seek information about specific data. E.g.: Wrappers "remember” that for information about address they should look for fifth <TR> tag and third <TD> tag. This type of content aggregation is volatile and is based upon the structure of document – every change of layout forces programmer to rewrite the wrapper.

Below there is presented diagram created by Tim Berners-Lee showing vision of Semantic Web. From the diagram one can deduce that basically it is a structure of a standard HTML document. But the main difference is that relations were added. Those relations describe how content of the document interacts with other data located on the Web. Since there exist only several types of those relations (often called *Metadata*), then there is no difficulty in implementing a given model. Thus it is possible that machines will be able to “understand” what content is being presented to the user (e.g. machines are capable of telling whether document describing Java refers to island or rather programming language; then they could present to user pages with travel agencies, or with books about programming).

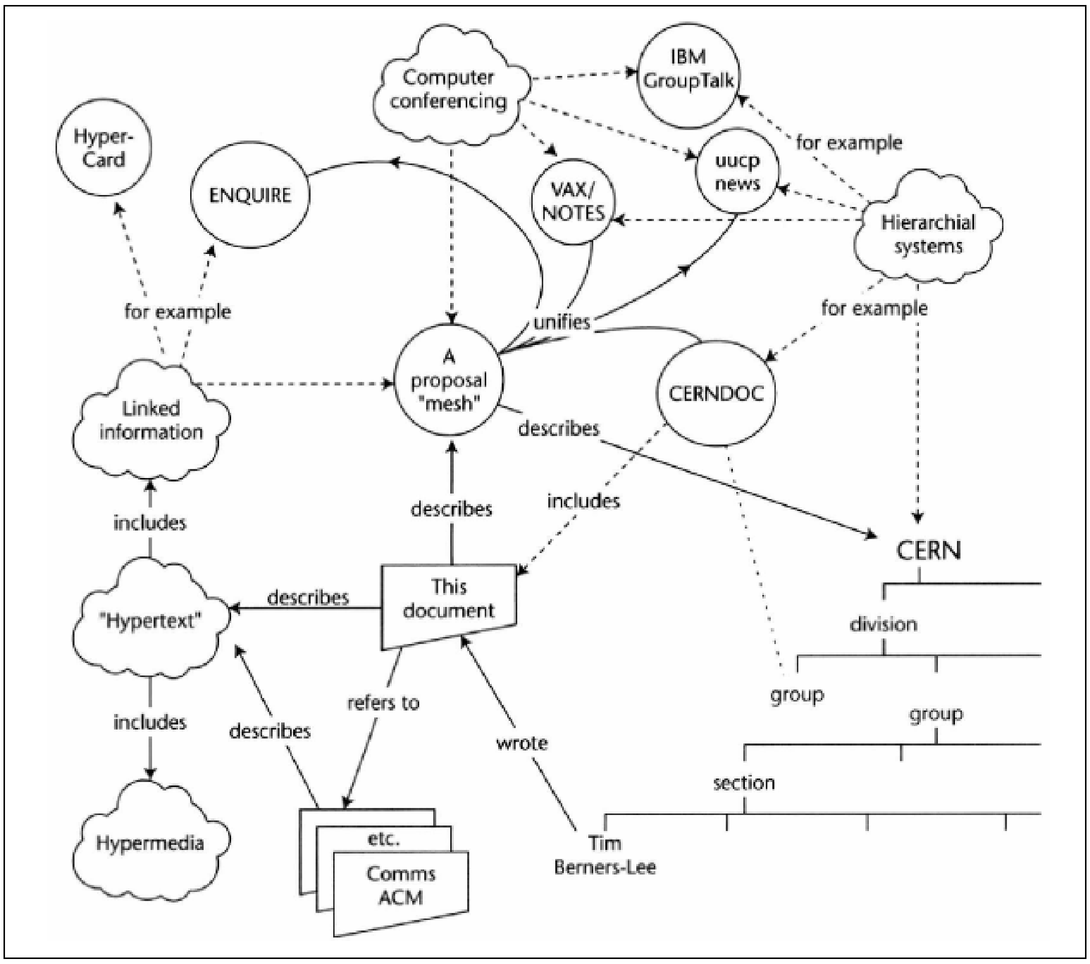


Figure 1: Vision of Semantic Web by Tim Berners-Lee

The Semantic Web is sometimes referred to as Web 3.0. Currently there is a huge amount of web sites that enable user to add own content and share it worldwide – this phenomena is called Web 2.0. Examples of those user-content based portals are: Flickr, MySpace, or in Poland Grono or Nasza-Klasa. Evolution of those portals was possible thanks to development of appropriate tools. Asynchronous JavaScript and XML (AJAX) is one among multiple technologies that allow addition of text, images, videos and documents real-time by concurrent users.

Web 3.0 means new human-computer relations, meaning that omnipresent machines, connected to the Internet, are capable of sensing human intentions. Thus computers will offer us products, entertainment or any other type of data that one really intends to use. One example of this can be refrigerator of the future – having ability to analyse our diet, knowing which products are preferred by us, capable of reading nourishment's expiry date. Such machine, when connected to the Internet will be able to make shopping instead of us, and controlling our food quality. It is hard to underestimate positive aspects of such solution.

Next diagram presents different stages of growth of Web. Beginning from simple USENET groups and FTP transmission through SQL databases placed on the Web until now – dynamically generated content using AJAX, instant news retrieval using RSS and ATOM. Current stage of evolution is provided for years 2000-2010. This document and many others in the Internet can be serious advantage in development of Semantic Web, and in future – Semantic Search.

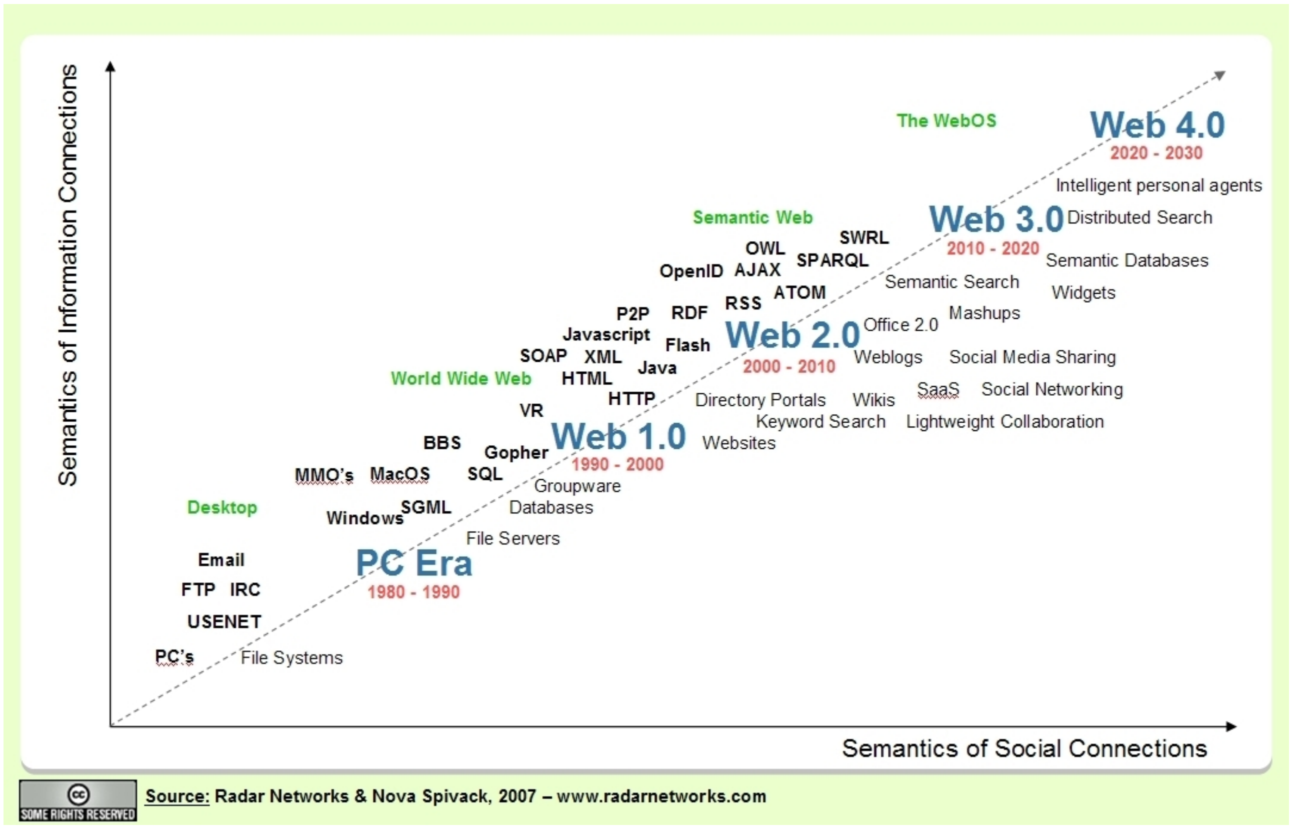


Figure 2: Diagram showing evolution of the Web and technologies used

When talking about Semantic Web, technologies used in it should be described. Two main technologies that make dream about Semantic Web come true are presented below:

- Software agents

Software agent is an autonomic programming unit, which is capable of interacting and communicating with other agents. Agent is simply natural step of evolution in Computer Engineering. When a programmer starts to design software, it is good to have tools that model our reality as closely as possible. Few years ago programmers had only low-level programming languages (like Assembler) then Bjarne Stroustrup invented classes, which could have properties, interfaces; they could inherit one each other – just like real life objects. Currently Software Agent is the highest possible level of abstraction which is capable of modelling reality. Agents can communicate, interact, and use each others' predispositions – just like people in real world.

In order to develop Agents one can use JADE framework written in JAVA programming language. There exist several different platforms, but this thesis is restricted only to JADE framework. Crucial part of agent-based system is communication – it makes agent learning possible. Messages are sent between agents using the Agent Communication Language (ACL).

[10][20][21]

- RDF
  Resource Description Framework is an XML-based model used to represent data and metadata. RDF metadata are statements about resources organized in triples in form: subject-predicate-object. Elementary example showing great advantage of data written in RDF format is ("Maciek", "is_author", "this_document"). Having data in this format human can easily deduce that: "Maciek is author of this document". The same logic can be applied when writing program gathering information from the Internet. Currently programs collecting information from web pages operate on strings, and are not capable of deciding whether: *"Maciek is author of this document"* and *"Maciek has written this document"* are statements meaning exactly the same or do they have nothing in common.

Travel Support System

The Travel Support System (TSS) is an approach to fully use benefits of the Semantic Web. It is a multi-agent system, where agents are responsible for collecting, managing and presenting data to the user. Data are stored in the RDF format. Main part of the system consists of restaurants – locations and specifications of those are taken from the ChefMoz database. It is the largest restaurant database which has its contents stored in RDF format. The TSS is developed by people World-wide. Top-level overview of the current architecture can be seen on the following picture:

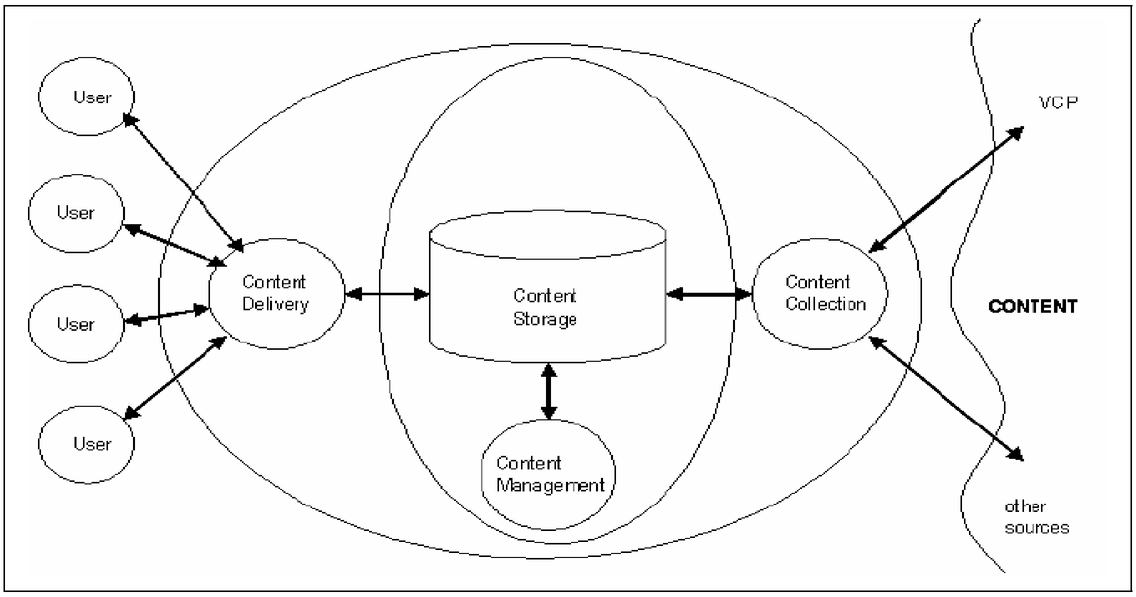


Figure 3: Overview of the Travel Support System Architecture

From this diagram one can see how the system works: external users connect to the Content Delivery Subsystem, which is responsible for presenting data retrieved from the Content Management Subsystem. Content Collection Subsystem collects the data and generates content either from Verified Content Providers (VCP) or other sources.

For storing data in the system the creators of TSS have decided to use Hewlett-Packard's JENA ontological database.

Slightly clearer is the Use-case diagram of the system, from one can easily deduce how the system resolves users' needs, and which agents perform which tasks:

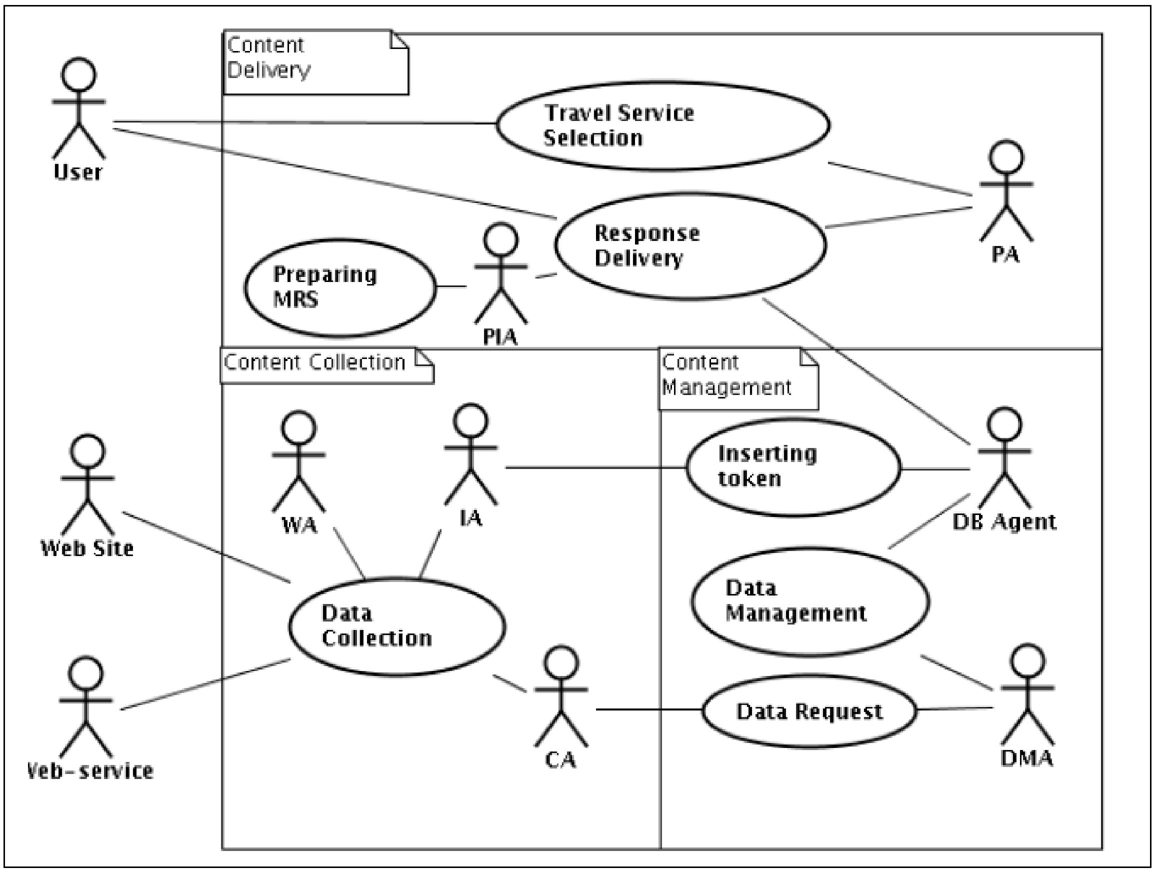


Figure 4: Use-case diagram of the system

The Travel Support System consists of three subsystems:

- **Content Delivery Subsystem**

  This subsystem is responsible for format and the semantics of the user-system communication. In this case when user sends a query, then the Personal Agent (PA) initiates user profile in the system and requests data from the Personalization Infrastructure Agent (PIA). PIA consists of a number of extremely simple rule-based “RDF sub-agents” – those subagents extend original query to obtain Maximum Result Set, which is delivered to the PA. Personal Agent filters, organizes and presents results to the user. The PA is also used in process of updating user’s profile.

- **Content Management Subsystem**

This subsystem is responsible for managing and assuring that data kept in database are up-to-date. It also is responsible for correctness of data. There are two agents present in this subsystem – Database Agent (DB Agent) – responsible for inserting data into Database – and Data Management Agent – which monitors system seeking for obsolete or incomplete data.

- **Content Collection Subsystem**

  This subsystem is responsible for collecting data from the Internet. Multiple Wrapper Agents (WA) travel through Internet searching and converting found data into RDF format. After conversion data are sent to Indexing Agent, which in cooperation with Database Agent inserts data into database. At the same time the Coordinator Agent (CA) waits for requests from DMA which informs Content Collection Subsystem about errors in data. CA instructs appropriate WA to gather required information. Content Collection Subsystem is the most important part from the point of view of this thesis. Structure and processes which take place in this subsystem will be presented in detail.

As for now, the Content Collection Subsystems gathers data only from the Verified Content Providers. Hopefully this thesis and software developed for the needs of it will have an impact so that the data for the database will be collected not only from VCPs but also from any existing source on the Internet.

Content Collection Subsystem Described

As mentioned before – this is the most important system from our point of view. Let us take a look at the schema of the subsystem:

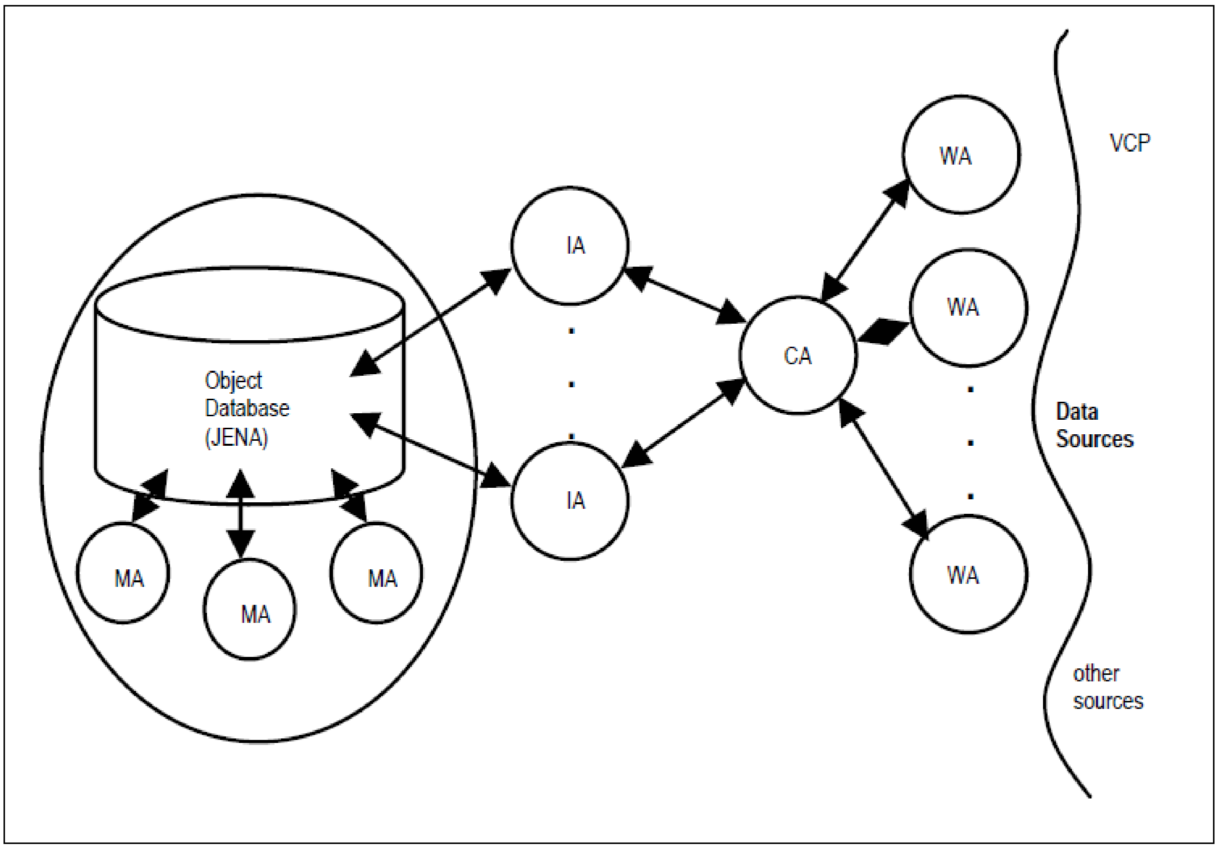


Figure 5: Content Collection Subsystem overview

Content Collection Subsystem consist of series of *Wrapper Agents* – since there are very few RDF-demarcated databases on the Internet, wrappers have to parse HTML pages one by one and extract data from it by using screen scrapping. WA generates RDF triples and sends those to the *Coordinator Agent* which schedules WAs and passes received triples to the JENA database. Currently there is one Wrapper Agent assigned to gathering data from one source (e.g.: Marriott hotel home page or Hilton hotel home page). Having one agent per VCP is very flexible idea – in case when one of those content providers decides to publish data in RDF format or changes page layout, then a programmer simply adjusts this WA and the system works properly. Whole CCS was implemented by Szymon Pisarek (subject to his master thesis). As in the model – there exists exactly one Coordinator Agent (which probably will be a performance bottleneck when the system grows) and there are several Wrapper Agents. Szymon Pisarek developed CCS as a standalone JAVA application [25]. One of the tasks was to integrate his code into the existing system. In order to do that it was necessary to add required libraries and make additional changes to the source code.

In the following image, the Content Collection Subsystem use-case diagram is presented:, where there are presented possible actions, as well as most important parts of the subsystem.

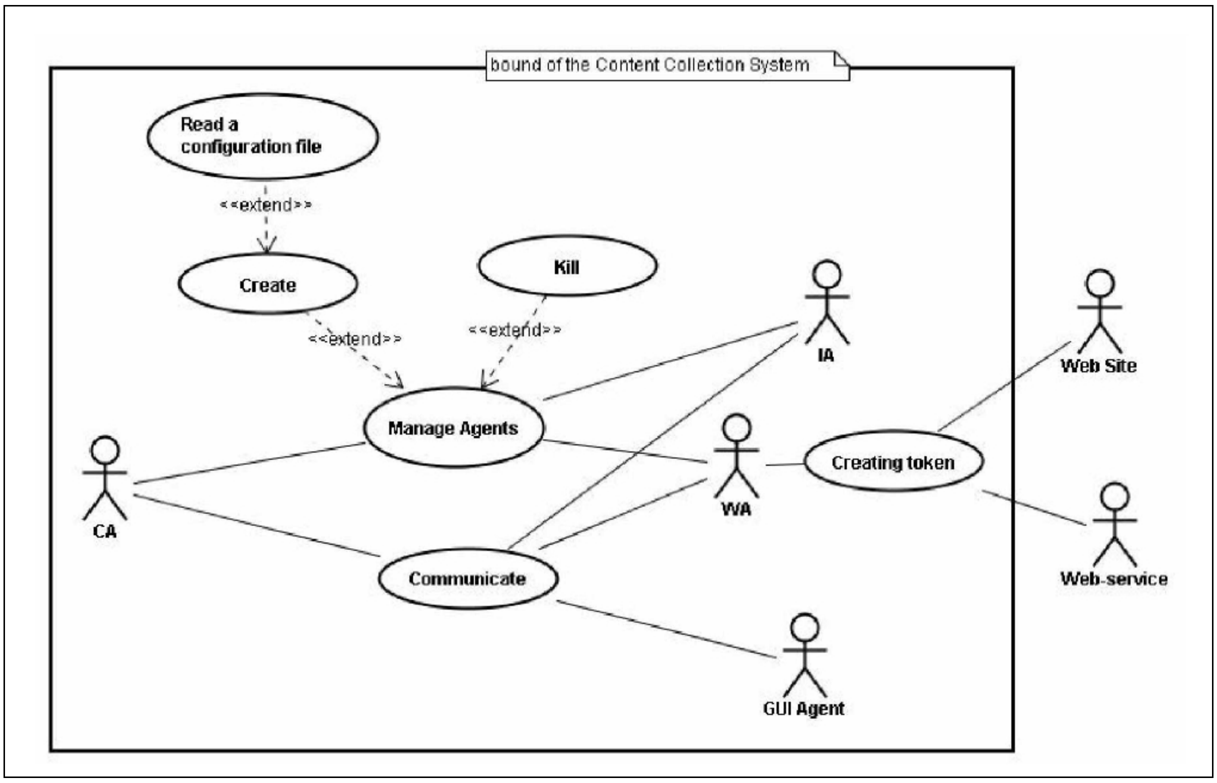


Figure 5: Content Collection Subsystem use-case diagram

As it was mentioned – central part of the system is Coordinator Agent which processes responses from Wrapper Agents, and supports requests from the Indexing Agent.

Missing Data Problem

Let us now consider possible problems which are caused by using multi-Wrapper technology. Having one Wrapper Agent per Verified Content Provider can lead to extreme situation, when programmers are not improving the system but are only maintaining it. For example let us assume that the system (in near future) consists of 50 WAs for every bigger hotel and 300 WAs for restaurants. Then let us make further assumption, that every hotel or restaurant changes its page once a year (e.g.: new menu, new holiday offer, new techniques for creating web pages etc.). This means that 350 pages change every year – this can lead to situation, when programmer has to write new wrapper agent almost every day. The situation presented is made with assumption that programmer is aware of fact that wrappers extract wrong data – thus programmer knows which part of the system has to be rewritten. If there is no such control, then the system can contain thousands of improper records and only way of finding that out is by chance (or when system users complain). This is one of the reasons that illustrate how big is the need for some mechanisms, which can gather data from almost every source and which are independent from the page layout.

Current solution (one Wrapper Agent per one Verified Content Provider) has another drawback – in the Content Management Subsystem there exist a group of agents called Data Management Agents. It is their role to find incomplete or erroneous data in the repository. Currently – when the DMA finds a record for which there is need to gather data once more from the Internet, this request is sent to the Coordinator Agent in the Content Collection Subsystem. Now the Coordinator Agent has no other way of acting but to call an appropriate Wrapper Agent and try to retrieve data once more. Since data in database were generated by Wrapper Agent, then (if nothing changed in the Web Site) the same data will be generated once more. Of course this is not an acceptable situation.

From previous paragraphs comes one conclusion – there is need for mechanism which is not dependant to changes in page layout. This mechanism should also be capable of extracting data from any web page or other data source. Since whole system is agent-based, then it would be good that such solution is also implemented as an agent. The best place for new agent behaviour is shown in the next diagram:

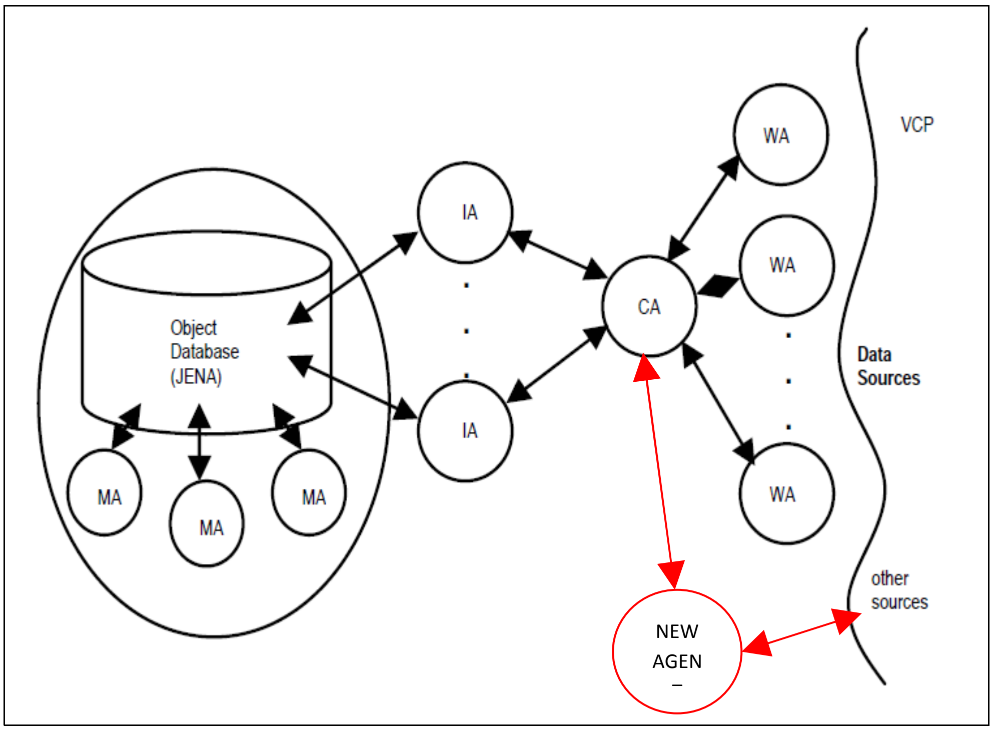


Figure 6: Content Collection Subsystem extended by the new wrapper

New wrapper should work in strict cooperation with the Coordinator Agent. It should be started whenever the CA receives message with an order to retrieve some missing data from the Internet. Newly created wrapper will be responsible for gathering data which could not be extracted by normal use of the appropriate wrapper. WAs use Verified Content Providers as their primary (and in most cases only) data source. Thus preferred place to start looking for missing data are "other sources". Of course new wrapper will be capable of extracting data from every source.

In order to achieve mentioned functionality it was needed to make changes to the Coordinator Agent originally developed by Szymon Pisarek. Let us now take a deeper look at the implementation of the CA. The complete logic responsible for coordinating the CCC is split into the following classes:

- *CoordinatorAgent*

  Description taken from the source code:

```
CoordinatorAgent
   Description taken from source code:

"This class represents Coordinator Agent who plays vital role in
   Content Collection Subsystem. He is responsible for:

1.       starting Indexing Agents at CCS start-up

2. starting Wrapper Agents at CCS start-up and starting them in
   appropriate time (in appropriate intervals)
```

Listing 1: Description of the CoordinatorAgent

Coordinator Agent can have three behaviours:

  - receiver – responsible for receiving the ACL messages
  - scheduler – responsible for starting Indexing Agents and Wrapper Agents.
  - GUIreceiver – responsible for receiving messages from the GUI agent.

Those behaviours are implemented into following three classes:

- *CoordinatorReceiver*

  Description from the source code:

```
This behaviour belongs to Coordinator, is responsible for
receiving

1.    an IndexToken [action] from a Wrapper Agent -
Coordinator saves the received token in its Tokens Priority
Queue

2.    an RequestToken action from an Indexing Agent -
Coordinator gets a token with the highest priority form its
Tokens Priority Queue and sends it to the Indexing Agent
```

Listing 2: Description of the CoordinatorReceiver behaviour

- *CoordinatorScheduler*

  Description from the source code:

```
“This class represents a scheduler of the Coordinator Agent.

 This class is responsible for:

 1.   starting Indexing Agents at CCS start-up

 2.   starting Wrapper Agents at CCS start-up and starting
them in appropriate time (after appropriate intervals)”
```

Listing 3: Description of the CoordinatorScheduler behaviour

- *CoordinatorGUIReceiver*

  This behaviour handles the GUI messages (e. g.: shutdown of the application should cause that all agents are stopped before termination of the system).

  Receiver behaviour and Scheduler behaviour are more complicated than GUIreceiver, so the creator has decided to move code not connected with agent programming into separate classes (this functionality is called Business Logic). Thus there exist two more classes:

- *CoordinatorReceiverBL, CoordinatorSchedulerBL*

CoordinatorReceiver class currently handles two types of requests. One request is from the Wrapper Agent when it is supplying new token to store in DB. Second request may come from the Indexing Agent which asks the Coordinator Agent for tokens created by Wrapper Agents. Here is the right place to add new functionality – *CoordinatorReceiver* should be able to receive message from the Data Management Agent which will be asking for missing data.

Let us inspect *"performAction"* method in *CoordinatorReceiverBL* class:

```
/**
 * Performs requested action.
 */
public void performAction(AgentAction action, ACLMessage response)
    throws AgentException, OntologyException, Codec.CodecException
{
    if (action == null)
        throw new IllegalArgumentException("AgentAction can't be null");
    if (response == null) throw new
        IllegalArgumentException("Response (ACLMessage) can't be null");

    if (action instanceof SaveAction)
        performSaveAction((SaveAction)action);
    else if (action instanceof RequestAction)
        performRequestAction((RequestAction)action, response);
    else
        throw new AgentException(myAgent.getAID(), "Unknown Action");

    response.setPerformative(ACLMessage.INFORM);
}
```

Listing 4: performAction method, from the Coordinator Agent

This method shows that this Agent's behaviour can only handle 'Save' action and 'Request' action. Any new behaviour that this agent should support must be added here.

.

Research

In the previous chapter the place in the system where the new functionality will be located was proposed. Also the functionality itself was discussed. Empty tokens that can be found are defined by the ontology. Ontologies tell what kind of data can be searched. In this master thesis there is a restriction that only address-based data will be taken into account. Hopefully the implementation can be easily extended to any ontology ever defined in the system.

Discovering way to find specific data on the web

The task is to parse hundreds of web pages for some specific data. This data is semi-structured; e.g.: addresses in Poland have a strict format. Most of them begin with text "ul.", followed by name of the street, then number of building and number of flat. Full address contains also zip-code which in Poland consists of two digits a dash and another three digits. Assumption is that all ontologically described data have some structure, which can be described in some machine-readable way.

Below there is presented list of several techniques that were considered during research phase of the master thesis:

1. Regular Expressions

Regular Expressions (often called regex or regexp) are special strings [27] that describe patterns and structure of tokens. Everyone who is creating documents is familiar with special sign, called *wildcard* – represented by sign *. Such special sign is used for finding all files ending with desired extension, e.g.: *.txt for text documents. Regular expressions language is far more detailed and can be used for describing more sophisticated and (if applicable) recursive patterns. Below there is an example of more complicated regular expression representing a valid e-mail address:

```
\b[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b
```

Regular expressions are widespread technology – each programming language and most of text editors have built-in processor responsible for recognizing and applying regex.

Knowing this technique, first thing that comes across one's mind is to write down regular expressions representing each type of data that can be represented in the system. After that one can simply apply selected expression to set of web pages until finding of proper data.

This approach has drawback – a programmer has got to write down address formats for every possible country in whole world. That is task impossible to complete. What is more – there is assumption that narrows search to address data, and in the system there exist ontologies that tell us about different details of objects; e.g.: whether the restaurant is non-smoking, or are the pets allowed. Some of those ontologies are hard to define, or may lead to over-generalization that multiple different data types are represented by single regular expression

2. Artificial Neural network

Often called simply *Neuron Network* is mathematical model based on biological neural network. It consists of group of neurons, typically distributed into three layers (input layer, hidden layer and output layer). There exist various types of neural networks (including recursive, and having multiple hidden layers). Each neuron in such network can have multiple inputs and multiple outputs. Input signals are weighted and result is passed to the activation function. Overview of such network can be seen here:

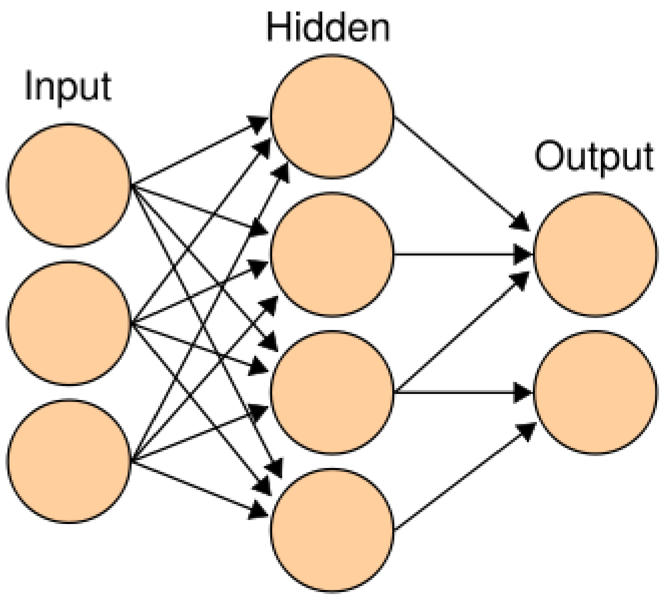


Image 7: Schema of artificial neural network

What is in interest to my master thesis is that neuron networks have the ability to learn – user can train neuron network by supporting set of input data and comparing it with desired output. Those artificial networks are widely used in weather forecasts, analysing stock market and for searching patterns as well.

3. Logic Wrappers and Inductive Logic Programming

"Logic wrappers [...] are a new technology for constructing wrappers for relational data extraction from the Web. This technology borrows ideas from the areas of logic programming and inductive learning" [1]

This technique is based on the fact that vast majority of web pages is written using HTML language, which can be transformed to XHTML (this means web pages which conform the syntax of XML)[3]. Such document can be represented by labelled ordered tree. L-wrapper is a set of patterns that is shared by structure and the data. One can say that this technique is very similar to the regular expressions described earlier, but there is main difference – L-wrappers based on the structure of the document, which is assumed to change often.

Logic Programming is the use of mathematical logic for computer programming. In this case it is used to learn L-wrappers how to extract data. [1][3][4][5]. this approach seems to be proper for presented task, nevertheless during work on the thesis, another technique was used.

4. Machine Learning

Machine Learning is a subfield of Artificial Intelligence [3], but in contrast, it does not try to simulate the way that human beings behave. Machine Learning tries to achieve the best result using currently available technologies and algorithms. Machine Learning is a science that tries to make desired tasks as automated as possible, hence it takes advantage of statistic methods. Several main types of Machine Learning are distinguished:

- Supervised learning – where algorithm maps input into desired output. In this case large set of examples is supported, together with desired result. Learning here is similar to learning process in Artificial Neuron Networks.
- Unsupervised learning – in this case desired output is not supplied, which means that algorithm presents its result based on observations of the set of inputs.
- Semi-supervised learning – combination of previous two types of learning. In some cases output is supplied, and in other test cases it is not.
- Reinforcement learning – technique where software agents learn how they should act to some action, in order to gain the most benefits.
- Transduction – technique where algorithm tries to predict output based on inputs and outputs used while learning.
- Learning to learn – algorithms learns its set if assumptions (inductive bias) used to predict given outputs [3][6]

From the techniques mentioned above I have decided to dedicate this thesis to the last – Machine Learning. I believe that this is the right approach – try to take advantage of techniques, not trying to make machines more intelligent than they really are.

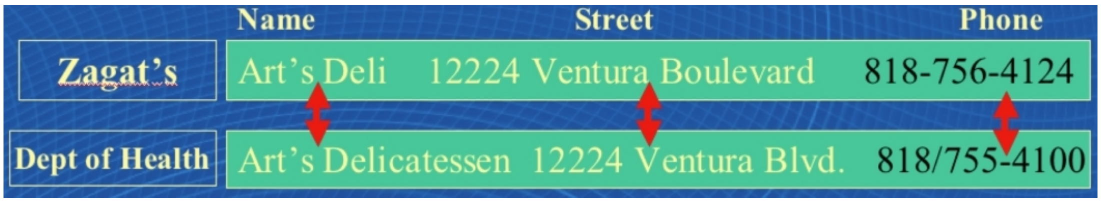
Moving in this direction I started to take a closer look at research work of Craig Knoblock, in the *Information Sciences Institute* at the *University of Southern California.* He is one of the leaders of a research group, which states that: "*Our research group is developing intelligent techniques to enable rapid and efficient information integration. The focus of our research has been on the technologies required for constructing distributed, integrated applications from online sources.*"[7]

Craig Knoblock and his achievements

As mentioned before Craig Knoblock led his group of researchers to discover way of data extraction from the web. The goal of this team was to create system capable of organizing and integrating huge amount of data. This system, if developed, could be a great assistance in travel planning, or analyzing biological data. Researchers have divided task into several sub-projects: Machine Learning, arbitrate whether records across web present the same data, automatic integration of data from multiple sources, managing execution plans in the Web and defining constraints for planning and integrating data. For the purpose of thesis, two projects are particularly interesting: Apollo [8] and Mercury[9].

1. The Apollo project

This project is devoted to finding techniques of linking records which represent the same object, but they are expressed in different ways; e.g. assume that there are two Internet sources that describe the same store, but their addresses are written differently:



	Name	Street	Phone
Zagat's	Art's Deli	12224 Ventura Boulevard	818-756-4124
Dept of Health	Art's Delicatessen	12224 Ventura Blvd.	818/755-4100

Image 8: Example of mapping of two addresses

Above there is presented example from the Apollo homepage. Address *Ventura Boulevard* can be also expressed in its shorter, yet understandable form: *Ventura Blvd.* As human beings can recognize those kinds of constructions without problem, then for machines these are two distinct sequences of characters.

The Apollo project deals with record inconsistency with system that uses two types of resolving such issues.

a. Building mapping tables

   A database-like table which contains records with information about possible mappings of one string into another

b. Mapping function

   A computer program that converts recorded data into another form.

Apollo joins both techniques in order to achieve semi-automated results – first of all system is building a simple mapping table from set of data to analyze, and then, using machine learning, system is trying to improve achieved mapping (by analyzing abbreviations, acronyms or word sequence in sentences.

This subsystem wasn't implemented in software included in this master thesis, but is seems natural way of evolution, and next step in automating Travel Support System.

2. The Mercury Project

Mercury Project is something like the Content Collection Subsystem in the Travel Support System – it is responsible for gathering data. Creators of the Mercury Project assumed that sources on the Internet appear all the time – extracting data from new source is connected with reprogramming it. In order to omit this part, there is a need for mechanism that would be capable of adapting data from new source.

The second of those projects utilizes variety of techniques that I have taken advantage of, for the needs of this master thesis.

DATAPROG algorithm

The Mercury project main task is to adapt newly created sources of data, in such way that implemented algorithms are capable of extracting valuable information. There is a variety of methods for accomplishing this task – Craig Knoblock together with group of researchers decided to use a machine learning approach.

Most of wrappers created by programmers are *landmark-based*, which is typical approach when extracting data from the web. *Landmark-based* wrappers have hot hard-coded structure of document – they are capable of extracting data from exactly one web page. This solution has major drawback – basing on outline of document means writing specific program extracting data from one specific page layout. Here, it is clear why HTML is not a good language to present data – every slight change in graphical layout of a web page means change in position where data is located. Since data is not separated from graphics, then a programmer must react to every such change, and then rewrite the wrapper.

When the dream of the Semantic Web will come true, then no wrappers will be needed – data on the Internet will be presented in machine-understandable form, probably in the RDF format. Since there are a few truly semantic data sources on the Web, then some mechanism should be supported, which extract data in semantic way – regardless to layout, can source, and data that user intends to receive. In order to approach towards Semantic Web, *content-based* wrappers must be considered. Those programs should be able to analyze type of data which user needs. Of course full, faultless analysis is not possible, but this can lead to promising results.

In this thesis we try to extract data from some semi-structured information sources. The goal of the wrapper algorithm will be to learn such structure. There are two ways of achieving this task: either learning algorithm can take into account negative examples, or positive examples. Full automation of the process is desired, but at the current state of development, system cannot run without user interaction. However, we assume that learning of the structure is done only once. Since there does not exist explicit source of negative examples, the algorithm will learn patterns based on positive examples. Programmer should be able to provide enough positive examples, to treat result as significant. Set of about one hundred test cases per type of data should be enough to talk about statistically significant data.

Since we are interested in address-related data, Let us note that each address in every country has some specific format. Let us consider example of few Polish addresses:

```
ul. Doroszewskiego 22/6 05-006 Łódź
ul. Chełmska 21/45 02-100 Chorzów
ul. 1 Maja 129/100 00-900 Lublin
ul. Niepodległości 16/23 01-222 Gdynia
```

Listing 5: Examples of Polish addresses

As it can be seen, they are not identical, but obviously share some similarities. The idea is to use pattern language, which will represent such records in set of regular expressions-like pattern.

Picture below shows the structure of such pattern language:

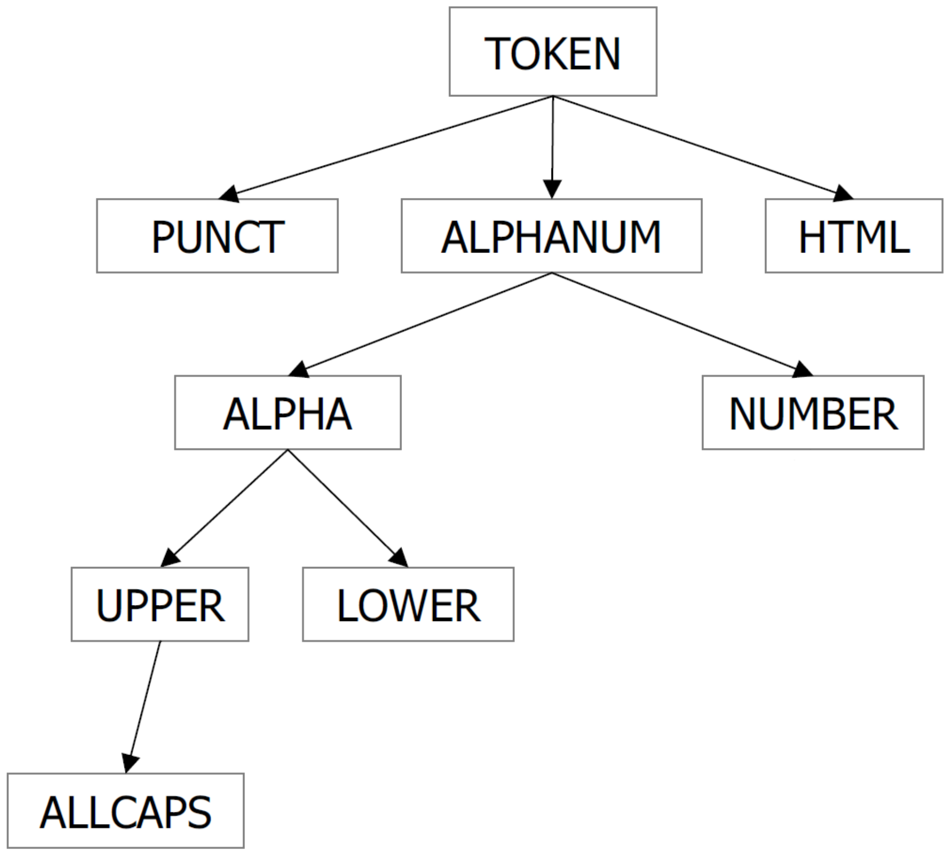


Image 9: Structure of pattern language used in DATAPROG algorithm

Several types of tokens (words that are considered) can be distinguished. They are organized into hierarchical structure, where each element of the tree means class of characters:

- TOKEN – abstract class of characters meaning any kind of character
- PUNCT – class of characters representing punctuation marks, which are one of following:
  - . (period)
  - , (comma)
  - ; (semicolon)

- : (colon)
- – (hyphen)
- _ (underscore)
- ! (exclamation mark)
- ? (question mark)
- ' (apostrophe)
- " (quotation mark)
- / (slash)
- \ (backslash)
- & (ampersand)

- HTML – class of characters, which are describing constructs specific to Hyper Text Markup Language. This class of characters means everything that is contained between < and > signs (opening and closing tags, respectively)

- ALPHANUM – class of characters that represent alphanumeric characters, meaning digits and letters of alphabet, regardless to letter case.

  Direct descendants of ALPHANUM class of characters are:

  - NUMBER – class of characters that represents any sequence of digits

- ALPHA – class of characters describing letters of alphabet, regardless to letter case

Direct descendants of ALPHA class of characters are:

  - LOWER – class of characters where ale letters in token are lower-case

  - UPPER – class of characters where the first letter of token is upper-cased, rest of the characters can have mixed-case

    The token UPPER has one direct descendant:

    - ALLCAPS – class of characters where all letters are capitalized

HTML has built-in specific character entities [13], for example: character < (less than) is represented by series of characters &lt;

HTML has about one hundred of such constructs – nevertheless none of them should appear in address, so those constructs are omitted..

Since pattern tree is organized in a hierarchical structure, then it is obvious, that token belonging to one class of characters belongs also to all parent classes.

Originally Craig Knoblock together with his group suggested slightly different token tree structure. It contained further division of NUMBER class into small numbers (e.g. those consisting of one number), medium and large.

What is interesting in presented structure is that it can be easily adapted, or extended to meet domain-specific requirements.

To the presented token classes new class is added, called *specific*, for every string that appears more than some fixed number of times. So it means, that if during processing presented earlier examples, algorithm will determine that tokens *ul* and . are appearing frequent enough, then structure will look like:

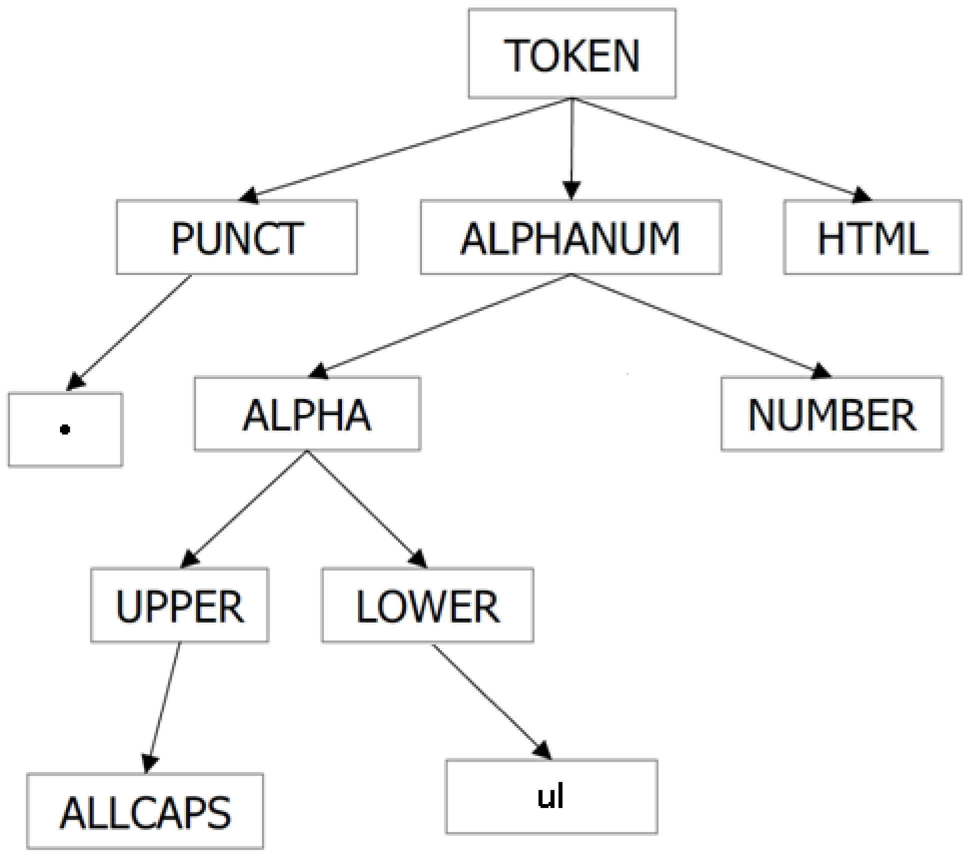


Image 10:Pattern tree after processing some examples

From above, after applying certain algorithm, one would come to fact that Polish address format consist of sequence of tokens:

<*ul*,., UPPER, NUMBER>

Please note that first two characters are specific token types – *ul* and “*.”* (full stop).

As Craig Knoblock remarks: "*a sequence of specific and general token types is more useful for describing the content of information than the character-level finite state representations...*".[12]

Such description of *data prototype* is easy to implement, and understand by human. What is more, it allows storing data in compact form.

The main part of data extraction is appropriate algorithm, capable of analyzing data, using pattern language described above. This algorithm, created by Craig Knoblock is called DATAPROG, it is successor to previous version called DATAPRO.

DATAPROG finds statistically significant sequences of tokens. Token is statistically significant whenever it occurs in examples more frequent than expected by chance. By phrase 'by chance' one can understand situation when tokens are generated in random way by uncontrolled source. In order to decide whether token's occurrence is significant, one needs to compute probabilities of each token type.

Since assumption was made that learning will be based on positive examples, then learning examples are treated as source of nearly 100% correct data. Computed probabilities are stored in a special structure called HashTable [14], which is an object mapping keys to values. What is more, it has been implemented in way that optimizes time of searching and inserting data. Having specified maximum load factor, and automatic change of size, it is powerful tool in computation. It can be useful for counting how many times specified word occurs in examples, and after that in counting probabilities of each token.

Table showing statistics about examples is shown below:

Token Type	Representant	Count
Alphanum		29
Number		17
Punct		12
Alpha		12

Upper		8
Lower		4
SpecificType	/	4
SpecificType	.	4
SpecificType	-	4
SpecificType	ul	4
SpecificType	100	2
SpecificType	Łódź	1
SpecificType	Doroszewskiego	1
SpecificType	23	1
SpecificType	22	1
SpecificType	21	1
SpecificType	Niepodległości	1
SpecificType	006	1
SpecificType	900	1
SpecificType	222	1
SpecificType	129	1
SpecificType	6	1
SpecificType	16	1
SpecificType	45	1
SpecificType	1	1
SpecificType	Lublin	1
SpecificType	Chorzów	1
SpecificType	Maja	1
SpecificType	05	1
SpecificType	02	1
SpecificType	01	1
SpecificType	00	1
SpecificType	Gdynia	1
SpecificType	Chełmska	1
AllCaps		0
HTML		0

Table 1: List of token types from training examples

Knowing number of occurrences of each token type, and total number of tokens, probabilities of occurrences can be computed of each token type. Results below:

Token Type	Representant	Probability
Alphanum		0,71
Number		0,41
Punct		0,29
Alpha		0,29
Upper		0,20
SpecificType	ul	0,10
SpecificType	/	0,10
SpecificType	.	0,10

SpecificType	-	0,10
Lower		0,10
SpecificType	100	0,05
SpecificType	Niepodległości	0,02
SpecificType	Maja	0,02
SpecificType	Łódź	0,02
SpecificType	Lublin	0,02
SpecificType	Gdynia	0,02
SpecificType	Doroszewskiego	0,02
SpecificType	Chorzów	0,02
SpecificType	Chełmska	0,02
SpecificType	900	0,02
SpecificType	6	0,02
SpecificType	45	0,02
SpecificType	23	0,02
SpecificType	222	0,02
SpecificType	22	0,02
SpecificType	21	0,02
SpecificType	16	0,02
SpecificType	129	0,02
SpecificType	1	0,02
SpecificType	05	0,02
SpecificType	02	0,02
SpecificType	01	0,02
SpecificType	006	0,02
SpecificType	00	0,02
HTML		0,00
AllCaps		0,00

Table 2: Probabilities of occurrences of each token type

After computing overall probabilities of each of token's calculation can be made how many times token <NUMBER> is expected to follow token <UPPER> completely by chance. If number of occurrences is larger than that value, then one can say that token <UPPER> is also significant, thus token <NUMBER, UPPER> is a data prototype.

In order to decide whether patterns are significant *hypothesis testing* is used [15]. This is a technique for determining whether given hypothesis is correct. Process of deciding of correctness is done by statistical computation.

Hypothesis testing consists of four steps:

1. Formulate the *null hypothesis* often represented by H_0 symbol. *Null hypothesis* is a case when observations are result of pure chance. *Alternative hypothesis* is formulated – that observations are 'real effect'.

2. Identify a *test statistic* used to test whether *null hypothesis* is true.

3. Compute the *P-value* – probability that test statistic would be obtained from null hypothesis. The smaller the *p-value*, the stronger evidence against *null hypothesis*.

4. Compare the *P-value* to a significance value α, called *alpha value*. This is a value from range $0 \leq \alpha \leq 1$, such that probability of observing at least *k* cases among *N* observations is less than *alpha value*. Mathematically this is denoted by equation:

$$P(k \geq N) \leq \alpha$$

   Choice of *alpha value* is crucial to the results of algorithm – it will decide whether to accept given token sequences or not. In other words – it will decide of significance. A variety of different *alpha values* may be used. Usually it is assumed that value is set to 5%. Nevertheless, based on observations and specific type of data used, I have decided to set significance level to value of 10%, which is sometimes called *almost significant*.

   If *P-value* is less than *alpha value* then observation is statistically significant, so the alternative hypothesis is valid.

Knowing the theory-behind, the definition can be made what particular mathematical terms mean in the presented task. Starting from the beginning, definition of *null hypothesis* must be provided. In presented case it means that sequences of tokens were generated randomly and completely by chance. The *test statistic* will be observing that given token sequence is plausible. Next point is to compute the *P-value*. Let us make assumption that there exist have *n* identical sequences, which were generated randomly. After computation of token's probabilities, it is known that specific, currently observed token type *T* has probability of occurrence *p*. Probability that this token will be the next token in *k* of those examples has got a binomial distribution. It can be proved that for large *n* binomial distribution approaches *normal distribution*[17]. *Normal distribution* is often referred to as *Gaussian distribution.*

Normal Distribution is defined as:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)/(2\sigma^2)}$$

Where x is variable, μ is *mean*, and σ is *variance.*

In this case, *mean* can be referred to as average value. Average number of occurrences of token whose probability is *p,* in *n* sequences is expressed by multiplication:

$$\mu = n * p$$

Variance is measure of statistical dispersion [3]. This value tells us of average squared distance of values from *mean*. *Variance* is calculated as:

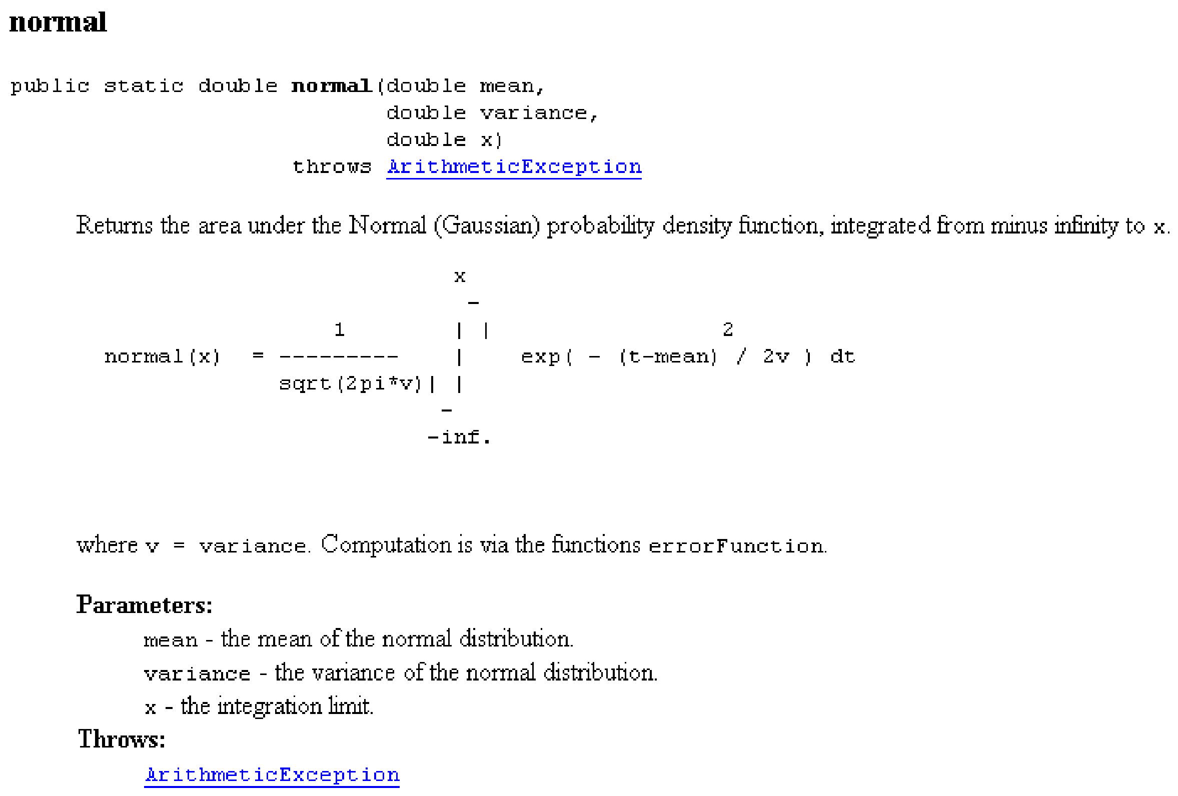
$$\sigma^2 = n * p * (1 - p)$$

The cumulative probability of observing at least n_1 events is given by equation:

$$P(k \geq n_1) = \int_{n_1}^{\infty} P(x, \mu, \sigma)$$

In the standard edition, JAVA does not contain any libraries providing tools for computing neither integrals, nor stochastic functions. Moreover, due to multiple definitions of word *integral*, it has hard to find appropriate libraries on the web. Here one can see how *Semantic Web* could help us in finding information about *calculating integrals* in JAVA, and not about *integral data types* in JAVA. I have managed to find COLT library [18]. This is open-source library that was created for the needs of High Performance Computing. The COLT project is carried out at CERN (European Organization for Nuclear Research), by scientists for scientists. JAVA is considered as slow environment for executing applications, and lack of advanced mathematical libraries is a proof that creators of JAVA think alike. Nevertheless developers at CERN who created COLT library claim that latest release breaks up the 1.9 Gflop/s barrier (floating point operations per second). The library is free to use and modify, with restriction that license agreement appears in source code of application taking advantage of COLT, as well as in supporting documentation.

The COLT library is divided into several packages. The one containing appropriate for us functions is called *cern.jet.stat.* Class responsible for computing statistical functions is called *Probability*. In that class there is defined static method *normal:*



normal

```
public static double normal(double mean,
                            double variance,
                            double x)
                     throws ArithmeticException
```

Returns the area under the Normal (Gaussian) probability density function, integrated from minus infinity to `x`.

```
                             x
                              -
                   1         | |                 2
normal(x)  = ---------        |    exp( - (t-mean) / 2v ) dt
             sqrt(2pi*v)| |
                             -
                           -inf.
```

where `v = variance`. Computation is via the functions `errorFunction`.

Parameters:

`mean` - the mean of the normal distribution.

`variance` - the variance of the normal distribution.

`x` - the integration limit.

Throws:

`ArithmeticException`

Image 11: description of computation of normal distribution in COLT library

This function will compute integral on range from minus infinity to variable *x*. In DATAPROG, there is need to calculate integral based on the interval from *x* to infinity. Knowing that:

$$\int_{-\infty}^{\infty} P(x,\mu,\sigma) = 1$$

This can be written down by:

$$\int_{-\infty}^{\infty} P(x,\mu,\sigma) = \int_{-\infty}^{x} P(x,\mu,\sigma) + \int_{x}^{\infty} P(x,\mu,\sigma)$$

Hence:

$$\int_{x}^{\infty} P(x,\mu,\sigma) = \int_{-\infty}^{\infty} P(x,\mu,\sigma) - \int_{-\infty}^{x} P(x,\mu,\sigma)$$

This is denoted by:

$$\int_{x}^{\infty} P(x,\mu,\sigma) = 1 - \int_{-\infty}^{x} P(x,\mu,\sigma)$$

From above comes out fact, that COLT library can be used to calculate probability, but from the result number 1 should be subtracted.

On next page pseudo code of the DATAPROG algorithm is shown:

```
DATAPROG MAIN LOOP
Create root node of tree;
For next node Q of tree
        Create children of Q;
        Prune nodes;
Extract patterns from tree;


CREATE CHILDREN OF Q
For each token type T at next position in examples
    Let C = NewNode;
    Let $C.token$ = T;
    Let $C.examples$ = Q.examples that are followed by T;
    Let $C.count$ = $|C.examples|$;
    Let $C.pattern$ = concat($Q.pattern\,T$);
    If Significant($C.count$, $Q.count$, $T.probability$)
                AddChildToTree(C, Q);
    End If
End T loop


PRUNE NODES
For each child C of Q
    For each sibling S of C s.t. $S.pattern \subset C.pattern$
        Let $N = C.count - S.count$
        If Not(Significant($N, Q.count, C.token.probability$))
              Delete C;
              break;
        Else
              Delete S;
        End If
    End S loop
End C loop


EXTRACT PATTERNS FROM TREE
Create empty list;
For every node Q of tree
    For every child C of Q
        Let $N = C.count - \sum_i (S_i.count | S_i \in Children(C))$
        If Significant( $N, Q.count, C.token.probability$)
            Add $C.pattern$ to the list;
Return (list of patterns);
```

Image 12: pseudo code of the DATAPROG algorithm

Algorithm starts with an empty tree – meaning that the only element is the root node, which has no influence on calculations of the algorithm. Next, algorithm iterates through all positions in examples – for each encountered token type a new *token* object is created. Please note that since creation of children for each token type is made, then for every string derivation of all possible token types must be made – from most general to most specific (including specific type representing the currently processed token). Each token type is tested whether it is significant, with respect to the parent node. If positive, then it is added as a child to the current node. After creation of children, nodes are pruned. DATAPROG is comparing every two sibling nodes, and eliminates the less significant. Above procedure is repeated for each element of the tree. Let us illustrate this procedure on the above introduced example.

First step is to split characters – especially punctuation marks from other signs. For needs of text-processing I have developed *Utilities* class. It contains method called *SplitTokensFromPunct*, its code is presented below:

```
/**
 * Method that splits tokens from punctuation signs. Eg: We have a Polish Zip-Code
 * passed as a string : "03-123", the result of this function is "03 - 123".
 * After this method, we can easily split string according to "space" characters
 * @param text
 * @return
 */
public static StringBuilder SplitTokensFromPunct(StringBuilder text)
{
    int punctIndex;

    // we iterate through punctuation signs, and check incoming string against it
    for (int i = 0; i < punctuation.length(); i++)
    {
        punctIndex = 0;

        // this while loop is for purpose, when a string contains more than one instance
        // of the same punctuation sign
        while (punctIndex >= 0)
        {
            // since we operate on StringBuffer, we have to operate on strings, not characters
            // so we have to "cut out" single character
            punctIndex = text.indexOf(punctuation.substring(i,i+1), punctIndex);

            // if there is a character
            if (punctIndex >= 0)
            {
                // we have found a punctuation sign in processed string

                // if it is not last character in string, and next character is not a whiteSpace,
                // then we put a "space" after the punctuation sign
                if (punctIndex + 1 < text.length() && Character.isWhitespace(text.charAt(punctIndex+1)) == false)
                {
                    text.insert(punctIndex+1," ");
                }

                // if sign befrore token is not a wihtespace, then we insert token
                if (punctIndex - 1 > 0 && Character.isWhitespace(text.charAt(punctIndex-1)) == false)
                {
                    // we do not forget about putting the "space" character before the punctuation sign
                    text.insert(punctIndex, " ");
                }
                // we have found punctuation sign at position punctIndex.
                // we have put a white-space at this place
                // so our new punctuation sign has position punctIndex+1
                // since we want to search string from the next position, then
                // we have to add 2 to the index
                punctIndex+=2;
            } // if (punctIndex >=0)
        } // while

    }// for

    return text;
}// end method
```

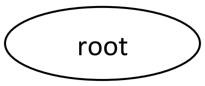
Listing 6: implementation of SplitTokensFromPunct method

After splitting, the following set of examples is obtained:

ul	.	Doroszewskiego	22	/	6	05	-	006	Łódź	
ul	.	Chełmska	21	/	45	02	-	100	Chorzów	
ul	.	1	Maja	129	/	100	00	-	900	Lublin
ul	.	Niepodległości	16	/	23	01	-	222	Gdynia	

Token tree consists of only the root node. Let us observe how data structures change when algorithm runs:

1. Token tree empty:



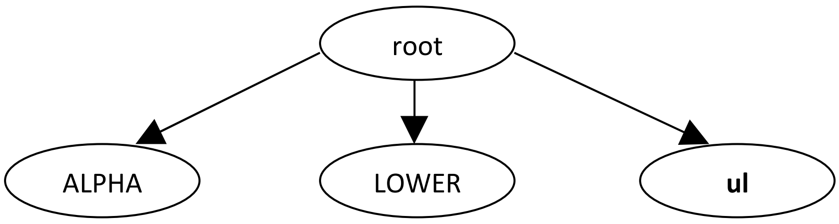
2. For each token at next position in examples:

ul	.	Doroszewskiego	22	/	6	05	-	006	Łódź	
ul	.	Chełmska	21	/	45	02	-	100	Chorzów	
ul	.	1	Maja	129	/	100	00	-	900	Lublin
ul	.	Niepodległości	16	/	23	01	-	222	Gdynia	

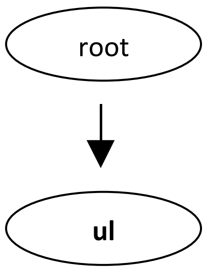
Current position is 1 – determination of each token type at this position is made. The result is:

<ALPHANUM, ALPHA, LOWER, **ul**>

Next, algorithm checks whether those token types are significant. In this case, it happens that all of them except ALPHANUM are significant. This token type is too general in this case, as it represents almost every token in examples. Something more specific is needed, to represent valuable data. Algorithm will treat the remaining 3 token types as significant. After this step the tree looks like:



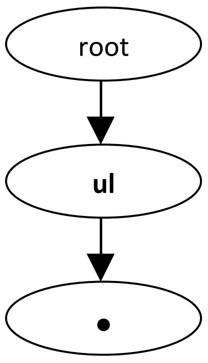
3. Next step is to prune the tree. Algorithm will compare every pair of sibling nodes, such that one of them is more general, and will leave only the more significant. Algorithm will leave the specific token, since it appears more times than expected by chance, and it fully covers the domain. The tree after pruning:



4. Repetition of the procedure for every node of the tree:

ul	.	Doroszewskiego	22	/	6	05	-	006	Łódź	
ul	.	Chełmska	21	/	45	02	-	100	Chorzów	
ul	.	1	Maja	129	/	100	00	-	900	Lublin
ul	.	Niepodległości	16	/	23	01	-	222	Gdynia	

This case is almost exactly the same as previous. The difference lies in obtained token types – here algorithm deals with <PUNCT> class of characters. Nevertheless – after pruning the algorithm will leave the specific type. The token tree is now:



5. The next position in examples is processed:

ul	.	Doroszewskiego	22	/	6	05	-	006	Łódź	
ul	.	Chełmska	21	/	45	02	-	100	Chorzów	
ul	.	1	Maja	129	/	100	00	-	900	Lublin
ul	.	Niepodległości	16	/	23	01	-	222	Gdynia	

Here, algorithm will recognize two different sets of token types

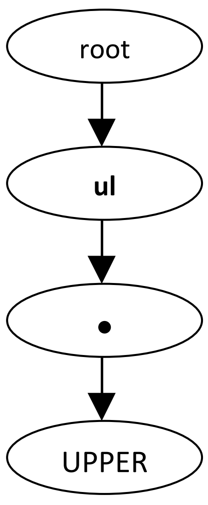
<ALPHANUM, ALPHA, UPPER, **Doroszewskiego, Chełmska, Niepodległości>**

And

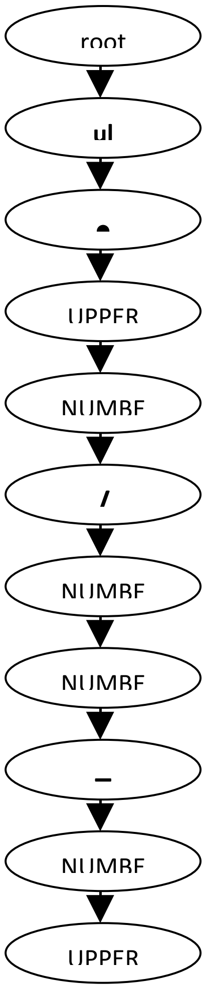
<ALPHANUM, NUMBER, **1>**

The latter of these sets will be treated as not significant – hence algorithm will reject the address "ul. 1 Maja 129 ...". This may lead to conclusions, that several Polish addresses will be treated as incorrect (Polish addresses mainly start with characters *ul.* , but this can be followed by a number or a word). As it was mentioned earlier – learning examples are treated as nearly 100% correct (from above one can see that algorithm will not crash when small percent of data is improper). If programmer is aware of possible address formats, then several learning examples should be supplied in order to create different token sequences.

After processing of this point, and calculations of significance, we receive following tree (after pruning):



6. Repetition of procedure for next node of tree – in this case for node containing type <UPPER>. After processing next records, tree has following shape:



From the above procedure the following token sequence is obtained:

<**ul**, •, UPPER, NUMBER, /, NUMBER, NUMBER,–, NUMBER, UPPER>

Presented pattern covers most of Polish addresses. After finding such sequence of patterns, the programmer's task is to apply the result to Internet pages in order to find desired information.

The use of DATAPROG algorithm in the Travel Support System

Presented method of obtaining patterns was used in Travel Support System. The Coordinator Agent has been equipped with a new behaviour – *FindMissingTokensAction.* This behaviour needs information – name of the object that search concerns, and an *ArrayList* object containing pattern which has to be searched. When called, it and uses the specially constructed class *PatternSearcher* that tries to find data on the Internet.

The *PatternSearcher* class queries *Google* for the name of the object. The class takes into account the first 10 results returned by search engine. Every link being returned is examined. To achieve this goal the *WebPageReader* class, which is responsible for reading contents of the web page, and supplying its source, was written. Furthermore, functionality that allows retrieving pure text from the web page (without HTML tags and special HTML symbols [28]) was also implemented. The resulting sequences of strings are put into the *HashTable*, in order to count number of occurrences. The result with greatest hit count is the one returned by to the Coordinator Agent. Testing the number of occurrences of specific string is a temporary solution. Development of trust-management based algorithm could be a good entry point to another master thesis.

Let us now take a look at example showing how the algorithm will try to search missing data.

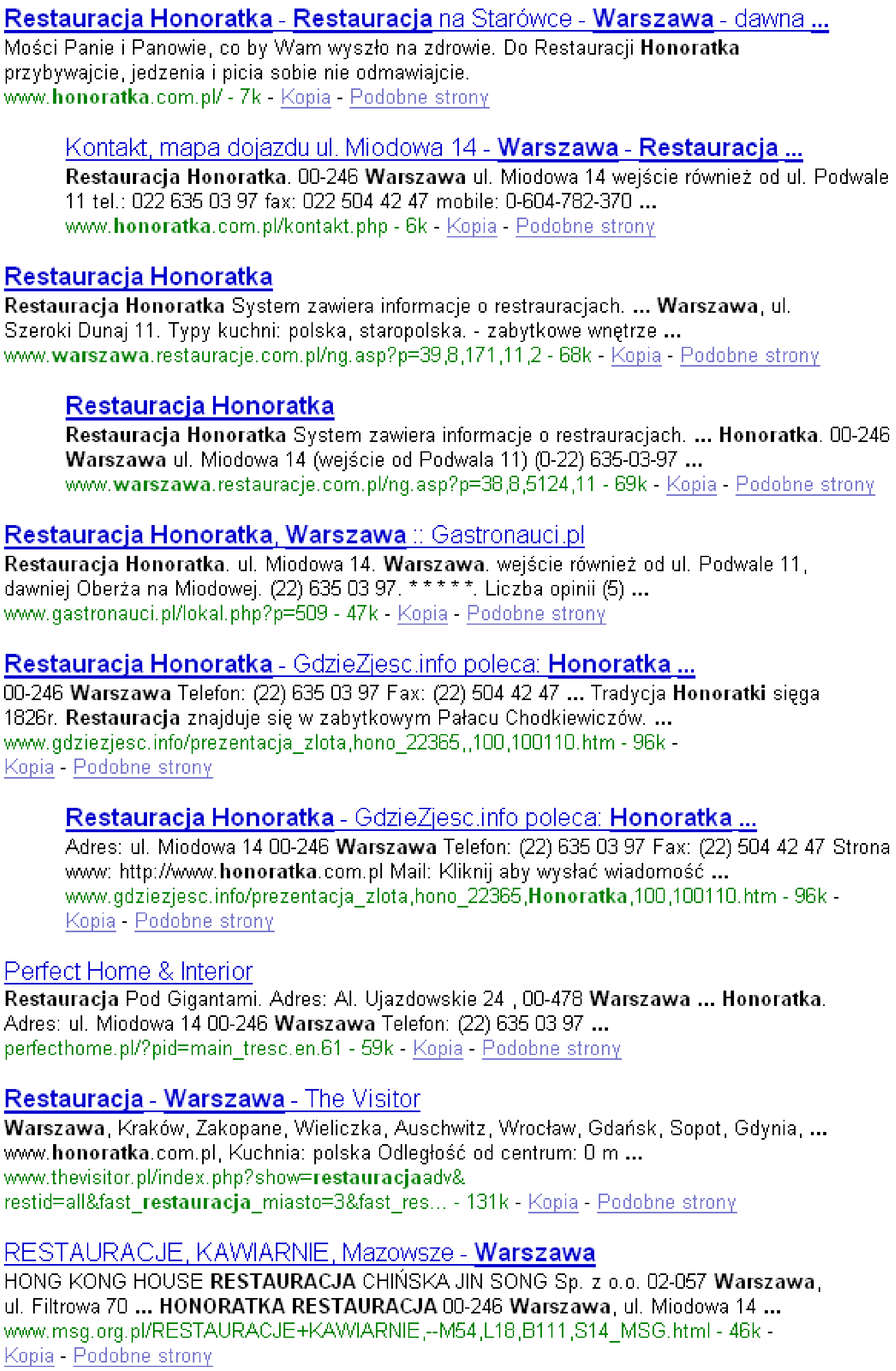
The test case is finding address of *Honoratka* restaurant, located in Warsaw, Poland. From DATAPROG algorithm it was discovered that Polish addresses are represented by sequence of tokens:

<**ul**, •, UPPER, NUMBER>

This is shortened, but also proper form of address. It does not contain information about zip code, nor city.

First thing that code logic does, is to query the Google homepage about data supplied as one of the parameters. In this case this would be: Honoratka Restaurant Warsaw. This string should be obtained from ontology. The name of the restaurant which should be checked is supplied by Data Management Agent (which monitors consistency of data).

When searching through Google one will obtain following results:



Restauracja Honoratka - Restauracja na Starówce - Warszawa - dawna ...
Mości Panie i Panowie, co by Wam wyszło na zdrowie. Do Restauracji **Honoratka** przybywajcie, jedzenia i picia sobie nie odmawiajcie.
www.**honoratka**.com.pl/ - 7k - Kopia - Podobne strony

Kontakt, mapa dojazdu ul. Miodowa 14 - Warszawa - Restauracja ...
Restauracja Honoratka. 00-246 **Warszawa** ul. Miodowa 14 wejście również od ul. Podwale 11 tel.: 022 635 03 97 fax: 022 504 42 47 mobile: 0-604-782-370 ...
www.**honoratka**.com.pl/kontakt.php - 6k - Kopia - Podobne strony

Restauracja Honoratka
Restauracja Honoratka System zawiera informacje o restrauracjach. ... **Warszawa**, ul. Szeroki Dunaj 11. Typy kuchni: polska, staropolska. - zabytkowe wnętrze ...
www.**warszawa**.restauracje.com.pl/ng.asp?p=39,8,171,11,2 - 68k - Kopia - Podobne strony

Restauracja Honoratka
Restauracja Honoratka System zawiera informacje o restrauracjach. ... **Honoratka**. 00-246 **Warszawa** ul. Miodowa 14 (wejście od Podwala 11) (0-22) 635-03-97 ...
www.**warszawa**.restauracje.com.pl/ng.asp?p=38,8,5124,11 - 69k - Kopia - Podobne strony

Restauracja Honoratka, Warszawa :: Gastronauci.pl
Restauracja Honoratka. ul. Miodowa 14. **Warszawa**. wejście również od ul. Podwale 11, dawniej Oberża na Miodowej. (22) 635 03 97. * * * * *. Liczba opinii (5) ...
www.gastronauci.pl/lokal.php?p=509 - 47k - Kopia - Podobne strony

Restauracja Honoratka - GdzieZjesc.info poleca: Honoratka ...
00-246 **Warszawa** Telefon: (22) 635 03 97 Fax: (22) 504 42 47 ... Tradycja **Honoratki** sięga 1826r. **Restauracja** znajduje się w zabytkowym Pałacu Chodkiewiczów. ...
www.gdziezjesc.info/prezentacja_zlota,hono_22365,,100,100110.htm - 96k - Kopia - Podobne strony

Restauracja Honoratka - GdzieZjesc.info poleca: Honoratka ...
Adres: ul. Miodowa 14 00-246 **Warszawa** Telefon: (22) 635 03 97 Fax: (22) 504 42 47 Strona www: http://www.**honoratka**.com.pl Mail: Kliknij aby wysłać wiadomość ...
www.gdziezjesc.info/prezentacja_zlota,hono_22365,**Honoratka**,100,100110.htm - 96k - Kopia - Podobne strony

Perfect Home & Interior
Restauracja Pod Gigantami. Adres: Al. Ujazdowskie 24 , 00-478 **Warszawa** ... **Honoratka**. Adres: ul. Miodowa 14 00-246 **Warszawa** Telefon: (22) 635 03 97 ...
perfecthome.pl/?pid=main_tresc.en.61 - 59k - Kopia - Podobne strony

Restauracja - Warszawa - The Visitor
Warszawa, Kraków, Zakopane, Wieliczka, Auschwitz, Wrocław, Gdańsk, Sopot, Gdynia, ... www.**honoratka**.com.pl, Kuchnia: polska Odległość od centrum: 0 m ...
www.thevisitor.pl/index.php?show=**restauracja**adv&restid=all&fast_**restauracja**_miasto=3&fast_res... - 131k - Kopia - Podobne strony

RESTAURACJE, KAWIARNIE, Mazowsze - Warszawa
HONG KONG HOUSE **RESTAURACJA** CHIŃSKA JIN SONG Sp. z o.o. 02-057 **Warszawa**, ul. Filtrowa 70 ... **HONORATKA RESTAURACJA** 00-246 **Warszawa**, ul. Miodowa 14 ...
www.msg.org.pl/RESTAURACJE+KAWIARNIE,-M54,L18,B111,S14_MSG.html - 46k - Kopia - Podobne strony

Image 13:Results from Google

Presented results may vary upon location, from which query was generated (one will probably see different results for machines located in USA and Poland). The first result returned by Google leads to main page of Honoratka restaurant.

Code locates all hyperlinks in obtained text. In HTML hyperlinks are represented by the <a href=”...> tag. For example – let us see the source code of first two results from Google:

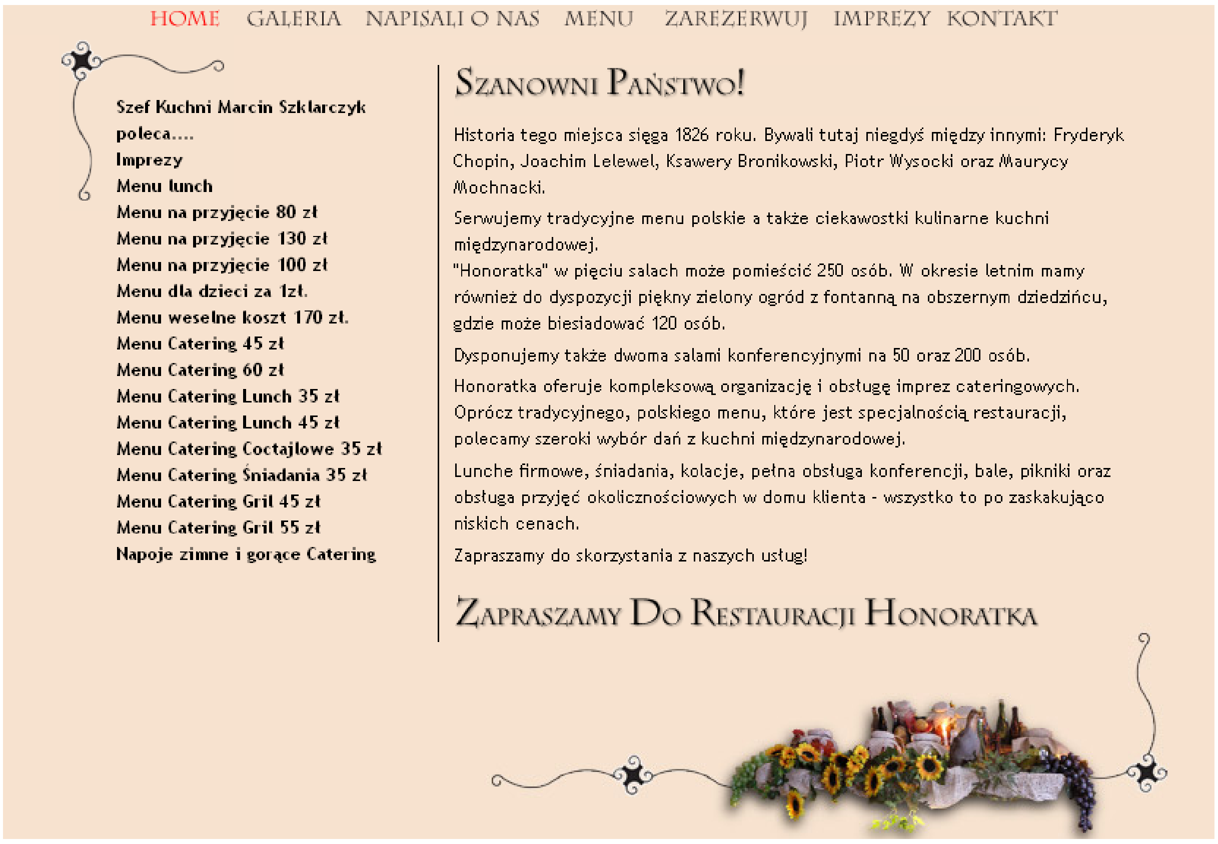
```
<h2 class=r>
    <a href="http://www.honoratka.com.pl/" class=l
       onmousedown="return clk(this.href,'','','res','1','')">
        <b>Restauracja Honoratka</b> - <b>Restauracja</b>
        na Starówce  - <b>Warszawa</b> - dawna <b>...</b>
    </a>
</h2>

<table border=0 cellpadding=0 cellspacing=0>
<tr>
    <td class="j">
    <div class=std>
        Mości Panie i Panowie, co by Wam wyszło na zdrowie. Do
        Restauracji <b>Honoratka</b> przybywajcie,
        jedzenia i picia sobie nie odmawiajcie.<br>
        <span class=a>www.<b>honoratka</b>.com.pl/ - 7k</span> -
        <nobr>
            <a class=fl
               href="/search?hl=pl&amp;lr=&amp;q=related:www.honoratka.com.pl/">
               Podobne strony</a>
        </nobr>
    </div>
    <!--n-->
    </td>
</tr>
</table>

<h2 class=r>
    <a href="http://www.honoratka.com.pl/kontakt.php" class=l
       onmousedown="return clk(this.href,'','','res','2','')">
        Kontakt, mapa dojazdu ul. Miodowa 14 - <b>Warszawa</b> -
        <b>Restauracja</b> <b>...</b>
    </a>
</h2>
```

Image 14: Source code of Google results

From above text, algorithm extracts the <a href=”..> tags, which will result in obtaining two hyper links:

- **http://www.honoratka.com.pl/** - the main page of the restaurant
- **http://www.honoratka.com.pl/kontakt.php** - page containing information about address.

For each obtained link, the *PatternSearcher* class is instantiated. The first page is shown on following screen (unnecessary Flash content has been cut out):



Szef Kuchni Marcin Szklarczyk poleca....
Imprezy
Menu lunch
Menu na przyjęcie 80 zł
Menu na przyjęcie 130 zł
Menu na przyjęcie 100 zł
Menu dla dzieci za 1zł.
Menu weselne koszt 170 zł.
Menu Catering 45 zł
Menu Catering 60 zł
Menu Catering Lunch 35 zł
Menu Catering Lunch 45 zł
Menu Catering Coctajlowe 35 zł
Menu Catering Śniadania 35 zł
Menu Catering Gril 45 zł
Menu Catering Gril 55 zł
Napoje zimne i gorące Catering

Szanowni Państwo!

Historia tego miejsca sięga 1826 roku. Bywali tutaj niegdyś między innymi: Fryderyk Chopin, Joachim Lelewel, Ksawery Bronikowski, Piotr Wysocki oraz Maurycy Mochnacki.

Serwujemy tradycyjne menu polskie a także ciekawostki kulinarne kuchni międzynarodowej.
"Honoratka" w pięciu salach może pomieścić 250 osób. W okresie letnim mamy również do dyspozycji piękny zielony ogród z fontanną na obszernym dziedzińcu, gdzie może biesiadować 120 osób.

Dysponujemy także dwoma salami konferencyjnymi na 50 oraz 200 osób.

Honoratka oferuje kompleksową organizację i obsługę imprez cateringowych. Oprócz tradycyjnego, polskiego menu, które jest specjalnością restauracji, polecamy szeroki wybór dań z kuchni międzynarodowej.

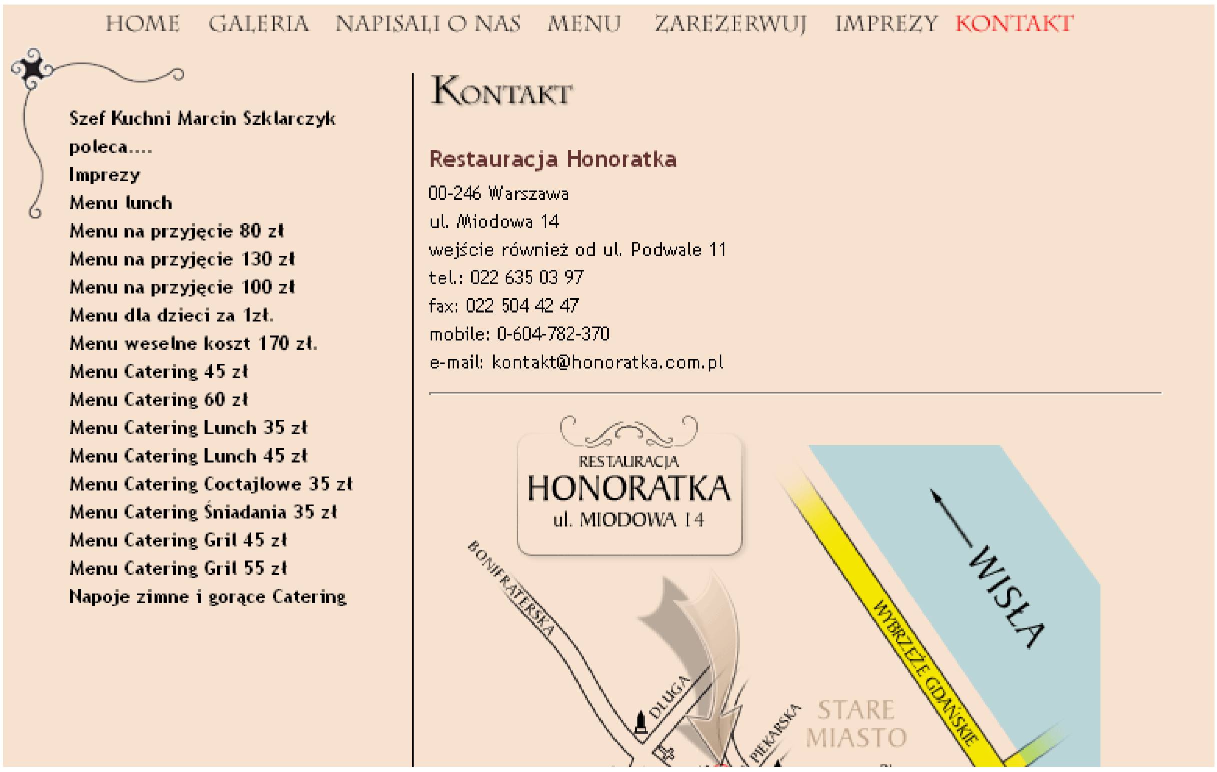
Lunche firmowe, śniadania, kolacje, pełna obsługa konferencji, bale, pikniki oraz obsługa przyjęć okolicznościowych w domu klienta - wszystko to po zaskakująco niskich cenach.

Zapraszamy do skorzystania z naszych usług!

Zapraszamy Do Restauracji Honoratka

Image 15: Main page of the ‘Honoratka’ restaurant

The second page overview:



Szef Kuchni Marcin Szklarczyk poleca....
Imprezy
Menu lunch
Menu na przyjęcie 80 zł
Menu na przyjęcie 130 zł
Menu na przyjęcie 100 zł
Menu dla dzieci za 1zł.
Menu weselne koszt 170 zł.
Menu Catering 45 zł
Menu Catering 60 zł
Menu Catering Lunch 35 zł
Menu Catering Lunch 45 zł
Menu Catering Coctajlowe 35 zł
Menu Catering Śniadania 35 zł
Menu Catering Gril 45 zł
Menu Catering Gril 55 zł
Napoje zimne i gorące Catering

KONTAKT

Restauracja Honoratka
00-246 Warszawa
ul. Miodowa 14
wejście również od ul. Podwale 11
tel.: 022 635 03 97
fax: 022 504 42 47
mobile: 0-604-782-370
e-mail: kontakt@honoratka.com.pl

Image 16: Page containing 'Honoratka' address data

First page does not contain any data connected with addresses – algorithm will not find any sequence of tokens beginning with text "ul .". Let us move then to the second page found by Google.

Source code of the part containing addresses is presented below:

```
<div class="opis">
    <img src="grafika/kontakt.gif" border="0" alt="">

    <h3>Restauracja Honoratka</h3>

    00-246 Warszawa<br />
    ul. Miodowa 14<br />
    wejście również od ul. Podwale 11<br />
    tel.: 022 635 03 97<br />
    fax: 022 504 42 47<br />
    mobile: 0-604-782-370<br />

    e-mail: <a href="mailto:kontakt@honoratka.com.pl">kontakt@honoratka.com.pl</a>
    <hr />
    <img src="grafika/mapa.gif" border="0" alt="Plan dojazdu - restauracja Honoratka" />
</div>
```

Image 17: Source code of page containing 'Honoratka' address data

Algorithm traverses through source code, searching for supplied pattern. What is important – search is performed on source code, which has got HTML tags removed. I have also developed method for splitting tokens from punctuation marks (so they are treated as separate characters). The above presented source will have following form:

```
Restauracja Honoratka

00 - 246 Warszawa
ul . Miodowa 14
wejście również od ul . Podwale 11
tel . : 022 635 03 97
fax : 022 504 42 47
mobile : 0 - 604 - 782 - 370

e - mail : kontakt@honoratka . com . pl
```

Image 18: Source code of 'Honoratka' restaurant contact page with HTML tags removed

The algorithm will examine such input, and will correctly recognize two addresses:

- ul. Miodowa 14
- ul. Podwale 11

Continuing with successive 8 web pages return by Google, algorithm will find various valid street addresses (from pages of other restaurants, or yellow pages). Results are in table below:

Found Address	Hit count
ul . Miodowa 14	11
ul . Podwale 11	4
ul . Berezyska 39	2
ul . Wiertnicza 96	2
ul . Długa 52	1
ul . Marszałkowska 55	1
ul . Marszałkowska 10	1

ul . Huculska 1	1
ul . Jezuicka 1	1
ul . Sadowa 2	1
ul . Grzybowska 47	1
ul . Wilcza 8	1
ul . Wilcza 43	1
ul . Królewska 2	1
ul . Mokotowska 45	1
ul . Wilcza 35	1
ul . Wiolinowa 14	1
ul . Zgoda 1	1
ul . Stawki 2	1
ul . Zgoda 4	1
ul . Chłodna 34	1
ul . Senatorska 3	1
ul . Wierzbowa 9	1
ul . Waliców 9	1
ul . Poligonowa 30	1
ul . Wrbla 3	1
ul . Różana 14	1
ul . Rejtana 14	1
ul . Wsplna 62	1
ul . Połczyńska 126	1

Table 3: Number of occurrences of patterns recognized as addresses

The address “ul. Miodowa 14”, which is the real address of “Honoratka” restaurant, occurs most often. Number of occurrences between this address, and the second on the list is big enough, to talk about statistically significant result.

Summary

In this Master thesis I have described what Semantic Web is. I have mentioned technologies used in development of SW. The goal if this thesis was to develop a mechanism, which will be capable of searching desired information on the web. I have done quick overview of possible ways of achieving this functionality.

I have successfully configured and deployed the Travel Support System. Into the system, I have embedded functionality written by Szymon Pisarek (Content Collection Subsystem – subject to his master thesis).

For the needs of the task, I have developed several classes and data structures, which are widely used in my implementation of DATAPROG algorithm – originally invented by Craig Knoblock. The algorithm is responsible for ‘learning’ patterns from the supplied examples. It uses hypothesis testing and uses advanced mathematical library COLT.

Results obtained from the DATAPROG are passed to the new behaviour added to Coordinator Agent (which can be found in CCS).

I have developed functionality that takes as input simple query and pattern to search. Algorithm searches the top 10 results from Google, and then for each encountered result, it searches for supplied pattern.

What is more, I have developed several useful tools that are used in text processing, HTML processing and statistical mathematical equations computing.

The task that was planned has been achieved, although there still are places where actions could be more automated.

This master thesis, and any software developed for the needs of it, or based upon it will bring us closer to the Semantic Web.

Bibliography:

1. Implementing Logic Wrappers Using XSLT Stylesheets, Amelia Bâdicâ, Costin Bâdicâ, Elvira Popescu

2. Google, http://www.google.com

3. Wikipedia, http://en.wikipedia.org

4. ontoX - A Method for Ontology-Driven Information Extraction, Burcu Yildiz and Silvia Miksch

5. Improving scalability of multiple crawlers based on contextualized query-sampling, Jason J. Jung, Yun-Sang Oh, and Geun-Sik Jo

6. Mathworld, http://mathworld.wolfram.com

7. Craig Knoblock homepage, Craig Knoblock, http://www.isi.edu/~knoblock/

8. Apollo Project homepage, Craig Knoblock et al., http://www.isi.edu/integration/Apollo/

9. Mercury Project homepage, Craig Knoblock et al.

   http://www.isi.edu/integration/Mercury/

10. Java Agent for Development (JADE) Framework, http://jade.tilab.com

11. Foundation for Intelligent Physical Agent (FIPA), **http://www.fipa.org**

12. Wrapper Maintenance: A Machine Learning Approach, Kristina Lerman, Steven N. Minton, Craig A. Knoblock

13. World Wide Web Consortium http://www.w3.org

14. JAVA API http://java.sun.com/javase/6/docs/api/

15. Mathworld - Hyphothesis testing, http://mathworld.wolfram.com/HypothesisTesting.html

16. MathWorld - Decisions based on P-values and Significance Level Tests

   http://demonstrations.wolfram.com/DecisionsBasedOnPValuesAndSignificanceLevels/

17. MathWorld - Binomial Distribution

   http://mathworld.wolfram.com/BinomialDistribution.html

18.COLT http://dsd.lbl.gov/~hoschek/colt/

19. Systemy agentowe w technologii RDF   Wawrzyniec Hyska

20.JADE Administrator's Guide   Fabio Bellifemine, Giovanni Caire, Tiziana Trucco (TILAB S.p.A., formerly CSELT). Giovanni Rimassa (FRAMeTech s.r.l.), Roland Mungenast (PROFACTOR GmbH)

21.JADE Tutorial   Giovanni Caire (TILAB, formerly CSELT), David Cabanillas (Technical University of Catalonia - UPC)

22. Transforming Arbitrary Tables into F-Logic Frames with TARTAR Aleksander Pivk, York Sure, Philipp Cimiano, Matjaz Gams, Vladislav Rajkovi, Rudi Studer

23. Zastosowanie ontologii do organizacji informacji pozyskiwanych z Internetu, Rafał Gąsiorowski

24. Modelowanie użytkownika na podstawie interakcji z systemem opartym o technologie WWW, Maciej Gawinecki

25. Utilizing Semantic Web and Software Agents in a Travel Support System Maria Ganzha, Maciej Gawinecki, Marcin Paprzycki, Rafał Gąsiorowski, Szymon Pisarek, Wawrzyniec Hyska

26. Ontologicznie zorientowane przeszukiwanie Internetu, Szymon Pisarek

27. Regular Expressions, http://www.regular-expressions.info/

28.HTML 4.01 Specification World Wide Web Consortium (W3C)

http://www.w3.org/TR/html401/

Warszawa, dnia ....

Oświadczenie

Oświadczam, że pracę magisterską pod tytułem; „Towards Semantic Web” ,

której promotorem jest dr Marcin Paprzycki wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

....