Charles University in Prague
Faculty of Mathematics and Physics

Master's Thesis

Bc. Martin Suchan

Semantics Detection in Partially Structured Sources

Department of Software Engineering
Supervisor: RNDr. Filip Zavoral, Ph.D.
Specialization: Computer Science, System architectures

2010
I would like to thank RNDr. Filip Zavoral, Ph.D. for supervising my master’s thesis, tuition and many valuable advices, and also thanks goes to my family for ongoing support.

I hereby declare that I wrote my thesis independently and exclusively using the referenced sources only.

I agree with lending of my work and its publication.

In Prague, 1st August of 2010

Martin Suchan
## Contents

Contents .................................................................................................................. 3

1 Introduction .......................................................................................................... 6
1.1 Data extraction ............................................................................................. 6
1.2 About this thesis ......................................................................................... 7

2 Methods of semantic detection ........................................................................... 9
2.1 Basic types of methods ............................................................................. 9
2.2 Advanced methods .................................................................................... 10

3 Analysis of data sources ................................................................................... 13
3.1 Structure of an email message .................................................................. 13
3.2 Structure of a HTML document ............................................................... 13
3.3 Other noticeable data formats .................................................................. 14
3.4 Implicit and explicit data structure ........................................................... 15
3.5 Sample CFP mailing list ........................................................................... 16
3.6 Analysis of the used semantics .................................................................. 16
3.7 Analyzing referenced web pages .............................................................. 18

4 Model of our semantic analysis ......................................................................... 19
4.1 Designing data processing model .............................................................. 19
4.2 Data gathering ............................................................................................. 20
4.3 Data filtering ............................................................................................... 21
4.4 Data parsing ................................................................................................. 22
4.5 Property generation .................................................................................... 23
4.6 Data matching ............................................................................................. 23

5 Processing documents ......................................................................................... 24
5.1 Processing emails ....................................................................................... 24
5.2 Processing web pages ................................................................................ 27
5.3 Data extraction ............................................................................................ 30
5.4 Other remarkable methods ......................................................................... 38
5.5 Common extraction problems .................................................................... 40
5.6 Time factor of gathered data ..................................................................... 41
5.7 Problem with data adding/replacing/removing ........................................ 42
5.8 Data matching and pairing ......................................................................... 43
6 Summary of results ................................................................. 45
6.1 Ideas for future work ......................................................... 45
7 Conclusion .................................................................................... 47
8 References .................................................................................... 48
A. Attachments ............................................................................... 50
B. Example application run .......................................................... 51
C. Document statistics .................................................................... 52
Title: Semantics Detection in Partially Structured Sources

Author: Martin Suchan

Department: Department of Software Engineering

Supervisor: RNDr. Filip Zavoral, Ph.D.

Supervisor’s email address: zavoral@ksi.ms.mff.cuni.cz

Abstract: The goal of this thesis is the comparison of methods for analysis of structured data sources, such as emails or HTML pages. The work focuses on practical assessment of common characteristics of these documents, which can be used for analysis, data extraction and cataloging for subsequent use. The work also includes a sample implementation of a program for cataloging data from emails and tracing changes in online sources.

Keywords: semantic analysis, data extraction

Název práce: Sémantická analýza částečně stukturovaných zdrojů

Autor: Martin Suchan

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Filip Zavoral, Ph.D.

Email vedoucího: zavoral@ksi.ms.mff.cuni.cz

Abstrakt: Obsahem této práce je porovnání možností analýzy strukturovaných zdrojů dat, jako jsou emaily či HTML stránky. Práce se zaměřuje na praktické zhodnocení spočetných znaků těchto dokumentů, které lze využít k analýze, extrakci dat a katalogizaci pro následné využití. Práce také obsahuje ukázkovou implementaci programu pro katalogizaci dat z emailů a dohledávání změn ve zdrojích dostupných online.

Klíčová slova: sémantická analýza, extrakce dat
1 Introduction

The semantic analysis used in computer science can be described as a process of relating syntactic structures from the input stream of text, dividing it into paragraphs, blocks or clauses and giving meaning to each of these objects. In compilers the semantic analysis has similar purpose – to specify and give meaning to symbols stored in parse tree, finding out which statements are legal and which ones are invalid. Special kind of semantic analysis is also called Latent semantics analysis [1] and it is a technique used in natural language processing used for finding and gathering relations between sets of documents. Semantics is basically a study about meaning, and Semantic analysis is a method for understanding the pattern, content and relation of a data in the context of a document.

Semantic analysis is used mostly by computers in processing various documents on the Internet but it is also used by humans in everyday activities as choosing a meal from menu in restaurant, when searching for a new book in a library or basically for understanding any written text or for comprehending new perceptions based on already learned experience. Although this process works for humans quite in a natural way, it is not simple to implement similar behavior as a computer algorithm.

1.1 Data extraction

The inspiration for this thesis is a common problem of extracting useful data from various data sources, especially partially-structured data sources. It is a problem mostly solvable using manual approach but finding a better, automatic algorithm based on findings from semantic analysis of the data source, that is what we are going to do. An example of such problem is the information extraction from documents of type “Call for papers” (CFP).

Typical source of CFP messages produces several documents every day. These documents share often similar structure and content type. Their content and purpose is straightly similar – inform all interested recipients about new planned events or send notification about changes in already proposed events. These messages generally contain information about the main topic of such event and also the date and place, where this event is taking place, invited speakers, detailed topics, and other data. One of the contained information in each email is also a reference to a web page or other related data sources. These referenced sources sometimes contain updated and more actual data related to the original information.
In the case of CFP documents, just reading hundreds of documents each month, sorting and updating them could be really exhaustive. Therefore it is a great idea to design some mechanisms for gathering such data automatically, sorting them, cataloging or updating. Such mechanism should work preferably fully automatically, or at least only with tolerable level of user input.

For accomplishing this idea we need to find a way, how to keep track on all those changes and how to track new information in referenced data sources. The question also is how to update found values in our local storage? This task can be generalized as “keeping track about important news from a specific news server and looking for relevant facts elsewhere”.

For accomplishing this task we are going to use several methods of semantic analysis and also various data extracting, filtering, matching and classifying algorithms.

1.2 About this thesis

The goal of this thesis is to propose possibilities for solving the problem mentioned above – extraction of valuable information from partially structured data sources and updating them thorough the process, especially targeting data extraction from CFP mailing list DBWorld [2]. Next task is also a wider analysis of such methods – which types of methods work, what kind of data structures to use, is it possible to incorporate existing algorithms or frameworks working with predefined grammar or semantics? The question also is: could the result of our work be used in Semantic web [3] as proper source of RDF triplets [4], mapped to some ontology language [5]? Although this thesis is targeted to one specific data source, it should be possible to re-use our methods in wider, generic area of interest.

For this purpose we design and implement several simple methods but when used in clever framework and context even those simple methods can offer great results. First of all we analyze the semantics of documents from [2] and determine, what typical structure they have and what are the signs common to most of these messages. The next part is devoted to finding out what kind of information we actually need to extract and store from these documents – identifying key areas of interest and most often mentioned subjects like typical dates and related events, locations, names, hosts, topics.

In the next chapter we design processing model for our application – framework with several layers containing data gathering methods, filtering procedures, methods for parsing and splitting the input text into applicable
stripes of text, methods for gathering interesting values from these strips and finally data matching and classifying methods for pairing and storing found samples. The last part of our work also describes the methods used for crawling related web pages, searching for actual or updated facts and storing them together with the previous data.

These described methods have been implemented in the application framework we have developed, tested on various sized data sources and evaluated by their effectiveness and completeness and also identified possible places for further improvement.

The last part also includes some interesting statistics and results acquired during the testing and programming, as analysis of occurrence of valuable data in documents we have processed and analysis of typical problems related to gathering data from partially-structured data sources.
2 Methods of semantic detection

There are several types of semantic detection methods that are currently widely used or implemented. Basically they could be divided into two groups – automatic methods, which are working on their own with no need of user interaction and semi-automatic methods working with human assisted cooperation, mostly using user-assisted learning.

2.1 Basic types of methods

For processing plaintext input streams, we can use these simple methods for semantics detection. These methods are based only on a simple lookup in the input text and are applicable for statically occurring semantics.

**Simple key-word lookup** – in this process we are just looking for the occurrence of any pre-selected *key-words* from a group of words. With a finite number of choices we have quite simple algorithm – all it does is string lookup.

Although this algorithm is quite simple and straightforward, it has also several drawbacks - with this algorithm we cannot find any new information. All we can find is which documents contain which key-words. This occurrence might give us some hint about the distribution of key-words across the input data set but that is probably all. Although this method in its basic form is not very powerful, is widely used for many simple purposes like in *profanity filters* word search used in various message boards, for generating *tag clouds* using keyword count or generating popular topics on websites. Also in a really simplified example – this is basically what web search engines and crawlers do. When user enters some word, the web crawler returns web sites containing such word and sorts them by a computed relevance factor.

In reality this is much more complicated process. Great introduction into the problem, i.e. how web search engines and web-crawlers work, is described in [6]. This work studies currently used web crawlers, their methods and effectiveness; also it is targeted for developing custom web crawler for finding science publications on the Internet and information extraction from those pages.
**Pattern based lookup** – as a slight upgrade from the previous simple method this one is used for finding strings matching specific input patterns. These patterns are called *regular expressions* or sometimes expressions with *wildcards*.

By using regular expression lookup we have a much stronger tool for finding requested data from data sources. The main difference from key-word lookup is that regular expressions are used not only for finding expected strings but also for finding new values matching the used pattern. Regular expressions are for example usable as simple tools for picking values of specific tags used on web pages.

The strong side of pattern lookup is the great flexibility of used patterns and collected results; the bad side is here still the fact that we need to design these regular expressions manually, so the results are still limited by the coverage of used regular expression.

### 2.2 Advanced methods

For processing more structured data sources we might use methods based on grammar analysis or even methods using advanced classifying or decision algorithms.

**Grammatical analysis** – is a process of correlating the line sequence of lexemes (words) of the language with its formal grammar. The result of this process is usually a parse tree or an abstract syntactical tree.

Using parser based on specific grammar for processing and analyzing input documents is probably the most powerful way how to process completely the document but unfortunately parsers could be used only for processing structured or at least heavily structured documents like program source codes. Using parsers for processing generic plain text documents i.e. email messages is not feasible because without proper applicable grammar we are in a dead end.

**Simple classifying methods** – when analyzing partially structured sources, there is a lot of possible scenarios for which we might need some semantic analysis. The most obvious scenario, when processing documents, is probably tagging the documents by their content or gathering interesting data – for these we might just need the keyword and regular expression methods.
For more advanced problems we might probably need more advanced methods like naive Bayes classifier [7] or statistics-based machine learning [8].

The classifying methods are great in situations as assigning input documents into several target classes – like identifying spam messages from proper emails. Bayes classifier in this case ranks all found words from input messages by the probability of assigning the whole message into one of several target classes (in the spam filter there are only two classes – spam and legal emails, sometimes also with a third class called “not sure”). When each word get individual rank for each target class, it is easy to compute the total index of each message. The total index says how high the chance this message belongs to each group is. At the end the message is assigned to the group with the highest index.

For methods like Bayes classifier it is necessary to teach the method first which messages belong to which group. This should be done using user-assisted learning – user assigns each message to the proper group manually, or by using some predefined learning set of data. About this method is also great that with more processed data there is a higher success ratio of assigning each message into valid class.

Other classification algorithms applicable in similar situations are cascading classifiers or decision trees [9].

Semantic web

Semantic web [3] is one of the most interesting approaches of incorporating semantics and semantics data processing into everyday life.

The problem with today’s computers and the Internet is that when computers and programs are communicating with each other, they are just passing the information without the understanding what the information actually means. They understand only the syntax of pages and information but not the semantics.

Semantic web is an extension of World Wide Web providing description of stored content – semantics, making it possible for machines to process and understand the content. It is basically a vision how it could be great, if semantics was a standard part of any document on Internet.

Fulfilling the goal of the Semantic web is rather a vision – it would require huge changes in the structure of World Wide Web to achieve the full understanding of contained data. A program capable understanding the semantics of stored data would be able to provide related data actively, based on the content and preferences, not just passively based on the keywords.
The current structure of the Semantic web is based on two cornerstones – standard format for storing and interchanging semantic data and connectivity among separate data sources.

The used data format, also called The Semantic web stack, consists of these standards/layers:

- XML and XML schema for defining the syntax and structure of Semantic web documents
- RDF schema [10] describing the properties and classes of RDF-based resources
- Unifying logic and Proof layers, which are not yet fully specified

In short the structure of the Semantic web data format is based mostly on modified XML data format called Resource Description Framework, which is used by Web Ontology language and is queried by SPARQL.
3 Analysis of data sources

Before implementing any algorithms we need to analyze the structure of some common data sources, especially the structure used in emails and web documents.

3.1 Structure of an email message

Email is a well-known partially structured document format used for sending electronic messages. Each email consists of several parts, where the first one is the header containing specific control information like email address of the sender, list of servers used for transferring the message, some spam-check related data and mostly also some email-server related specifics. These data are mostly used only during the process of sending and receiving and are generally not shown to the user. The second part of each message is the content itself – mostly it is just a document in plain text formatted only by the authors, sometimes it is HTML page or even other type of document. Optional part of any email is variable number of attachments but they are not in our interest in this case.

Although the email message structure is quite well defined, there are several problems and limitations involved with parsing and understanding the message. In general, the most common problems with email messages are problems with character encoding. Although the most common encoding today is UTF-8, it is not rare to get emails encoded in Win-1250 or other specific character sets. Together with character encoding is sometimes used the body encoding, which is mostly Base64. This encoding is used either for the email body or sometimes even for the email Subject.

3.2 Structure of a HTML document

HTML is a well-known structured format used for storing and presenting the content of Internet web pages. HTML is based on a XML-like format defined in several distinct recommendations or specification like HTML 4.01, XHTML or upcoming HTML5. We do not actually need to handle these formats separately because the used structure is mostly similar.
The content of each HTML message can be divided into head and body. The head contains several distinct tags like title and meta, and also references to various script or style files. The meta-tags contains mostly information like used encoding, language, author and also sometimes indexing rules for web crawlers.

The content of HTML body is structured using several well-known HTML tags like h1, h2 for document headings, table, div, frame or span tags for defining the structure of the document and tags like ul, ol and p for common paragraphs and lists.

For accessing and extracting data from web page document it is easy to use some HTML parser for finding tags most likely containing important data, or in the case of invalid HTML code where HTML parser in not fully applicable, we might want to use simple regular expressions for matching specific or even damaged tags.

3.3 Other noticeable data formats

There is lot of other partially-formatted data formats used more or less globally. These ones are probably the most interesting for our semantic analysis. Although they are not covered by this thesis, they could be used for analysis in future work.

RSS [12] is a format for “delivering regularly changing web content”. It is basically a hybrid between web page and email, widely used for notifying about news in specific sites or sending “public” messages. The structure is similar to emails – the RSS feed is basically an XML file structured into single messages. Each message contains several tags with title, date, link and also the content. The content could be plaintext or HTML, or even other format just like in the email. RSS feed messages are even sometimes by their nature interchangeable with email messages or other kinds of documents.

Extracting data from RSS feed is really just a matter of design simple XML parser and using tag based lookup for gathering requested information.
Wikipedia [13] is a well-known open Internet encyclopedia, providing free, up-to-date information about various topics. The wiki pages are often a great source of information, so looking for a related data here might not be a bad idea, while keeping in mind the possible issues with trustworthiness.

Using Wikipedia as a data source is not hard. We could use the search form for finding the requested topic or even use the referenced pages or documents for further search. When searching the target site, we could extract data using same methods as when extracting data from any other page or we may use the specific wiki editing pages, which are written using special meta-language and processed on a server-side for generating the final wiki HTML pages, as the source for our data.

Twitter [14] is a quite new and popular messaging service used for posting small messages about “what is happening right now”. Twitter has been also proven as a valuable source of real-time information.

The format of Twitter feeds is really simple – string of a plain text of a maximum size 140 characters. This string often contains links to related web pages, pictures, words with “#” symbol, so called topics or hash words, used for finding similar content or references to other users of Twitter marked with “@” sign. The analysis of typical Twitter messages could give us really up-to-date information about important events, people or affairs, all we need is to use either the Twitter web page or some implementation of the open Twitter API [15] for searching of keywords or reading messages from a specific source.

3.4 Implicit and explicit data structure

After analyzing several emails and web pages, we have found that there isrecognizable implicit and explicit data structure in both data sources.

The implicit structure is a stable structure defined by the data format itself – it is the header structure in emails and basic html structure in web pages. This structure mostly contains valuable meta-data for us, like the title of the related event or even some basic dates or deadlines.

The explicit structure is a specific structure used across documents sharing the same goal, recipients, type of people, topics or similar data sources.

In the case of [2], the explicit structure is the structure used in most CFP emails. Although there are variations in certain documents, the basic structure of those mails is simple – there is a title, abstract, some basic
information, block with dates and references and also footer. If we were able to identify these blocks in each mail, we could speed up the analysis and overall improve the quality of extracted information.

3.5 Sample CFP mailing list

The selected data source for our research is [2]. It is a standard mailing list with a simple web interface for sending new documents to the whole community of recipients. The frequency of receiving new emails in this conference varies but it is about several messages a day, which makes more than a hundred messages each month, mostly it is even more! Most of the emails there are CFP emails, sometimes there are also of other kind of documents – proposals for post-gradual study, new job offerings, etc. but such types of emails represents only a minority of the whole content.

As we have found, most of CFP documents there share similar structure – they contain a header with the name of the proposed action, then brief description, some important facts like dates, persons, places, and lastly a more detailed description or list of events, topics and sometimes even a picture. Considering such explicit structure gives us the first idea, where to start.

Another noticeable feature of these documents is also the structure of the email header – most Subjects of these emails start with “[DBWorld]” keyword, followed by the name of the proposed event. What is even more interesting is the presence of specific meta-tags in the email header. These tags contain valuable information, which is used in one of our methods of data gathering.

3.6 Analysis of the used semantics

When extracting important data from CFP emails, we need to specify which ones we are interested in and also what kind of data are easy to extract.

Most of CFP messages share a similar structure because of the common goal – to inform the recipients about upcoming event. The most important information is what the main topic of the proposed event is. Other important data in each email are names of invited speakers, the location of the event, the actual date of the event, important dates for submitting papers or sending reservation, name of the organizing authority and also references to web sites related to the event.
If we write down all these specific interesting points from each CFP email, then we should classify which ones are more important for us than the others and also which ones are “easy to find” and which ones are not.

With this information on mind, we are ready for proposing the target areas for our data extraction methods and algorithms.

<table>
<thead>
<tr>
<th>Type of information</th>
<th>Importance</th>
<th>Easiness to extract</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the event</td>
<td>Very high</td>
<td>Medium</td>
<td>Every time</td>
</tr>
<tr>
<td>Date + event</td>
<td>High</td>
<td>Easy</td>
<td>Very often</td>
</tr>
<tr>
<td>Web address</td>
<td>Medium</td>
<td>Easy</td>
<td>Very often</td>
</tr>
<tr>
<td>Topics, areas of interest</td>
<td>Medium</td>
<td>Hard</td>
<td>Often</td>
</tr>
<tr>
<td>Location, names</td>
<td>Low</td>
<td>Medium</td>
<td>Often</td>
</tr>
</tbody>
</table>

Figure 3.1 Importance of information in CFP messages

After analyzing the semantics of our data source and finding the most appealing targets for our data extraction methods we have also proposed the actual goal for our data extraction framework. We need to be able extract information of type event name, referenced web addresses and also dates with related information. Also using the found web addresses we should be able to download the referenced web pages and process them in a similar manner – try to gather similar types of information and using some matching mechanisms update our previously found values from emails with the actual ones from web pages. This process should basically replace the manual annotation process used by human recipients for storing and indexing received event proposals.
3.7 Analyzing referenced web pages

After downloading and processing emails from our data sources and annotating important information from each email we have also found lot of referenced web pages with information related to the original CFP events. For gaining more actual results we need to analyze the semantic structure of these web pages and gather more actual data there. It is actually common that there is more current information on official web pages than in received emails.

For designing the web page data extraction process we have manually downloaded tens of referenced web pages and searched them for common signs. The first interesting finding was the fact that the directly referenced pages contained mostly generic information only about the event, not detailed info we were looking for. The actual dates and deadlines for the event were mostly located on a sub-page or just deeper in the global web structure. We have also found out that the referenced pages were sometimes not related to the original event at all, contained only passive link or active redirection script or the web address was not working at all, so when designing automatic data extraction process we should keep on mind all these findings.

The goal of this process is also the ability to download periodically new emails and actual web pages and use new contained values for updating already stored information. This process should be mostly automatic with minimum user interference.
4 Model of our semantic analysis

4.1 Designing data processing model

When designing data processing model for our semantic analysis, we have first developed solid algorithms covering all phases of data processing from downloading emails, parsing and finally gathering important data and storing them in specific tags. Although this process worked pretty well, it was not flexible at all and it was also not suitable for further improvements and adding new features.

We have also come across the issue with reusability of our existing methods for parsing different data sources. We were unable to adapt our processing methods used in email parsing for further use in web page processing because of the different data format.

With these facts on mind, we have developed new 5-layered application processing model, see Figure 4.1, containing data providing methods for downloading the initial documents, data filtering methods for removing unwanted results, data parsing structures for stripping possibly valuable strips of text from the original documents, property generating layer containing most of the data gathering algorithms and finally the data matching and pairing layer. The last layer is used for assigning found properties into the right event tag and also for merging tags related to the same event.
### 4.2 Data gathering

Data gathering is the first of our processing layers. We are using this layer for downloading emails from email inbox using POP3 protocol [16] or loading emails dumped in a file. This layer is also used for downloading the source code of referenced web pages. Documents from these sources are then passed to the other processing stages.

Although the document downloading from the Internet is quite simple and automatic process, there are several places for further improvements like adding support for accessing encrypted email boxes using SSL, downloading web pages from automatically redirected addresses or accessing other types of data sources such as downloading RSS messages.

![Figure 4.1 Model of the semantic analysis](image)

| Data provider | • Raw emails from POP3/dumped in a file  
|               | • Raw web pages |
| Data filter   | • Pick only related emails/web pages. Drop all others.  
|               | • Key-word search in email header, HTML Title, document body. |
| Data parser   | • Find all possibly interesting strips of data.  
|               | • Email meta-tags, picked web page tags, lines with interesting values. |
| Property generator | • Generate properties from found strips.  
|                 | • Specific properties like web/email addresses, dates or titles. |
| Data matcher  | • Join found properties with already existing tag or create new one.  
|               | • Match similar properties together using several processing methods. |
4.3 Data filtering

When processing documents from a specific mailing list, we should be aware that some of the documents might not be related to the others at all, in short, they are spam. As spam we mean not only typical unrelated messages but also documents partially related to the email conference but somehow not containing any data we are currently focused on, for example in our mailing list it is quite common to get emails not related to CFP events, such as the proposals for post-gradual study, new job offerings, etc. but such emails represents only a minority of the whole content.

We do not need to handle in a special way those messages, if the ratio expected vs. spam is high enough. Otherwise we need to filter such messages in our data filtering layer.

The data filtering layer is probably the youngest one we are using. At the beginning we were processing only emails from a specific email box containing only messages from our data source and we had no need for filtering those messages – all of them were related for our processing. Later we thought that this is probably not the most common situation. The recipients of this mailing list are mostly receiving these emails using their major message box and such messages make only a fraction of the whole content. Processing only those really relevant messages is therefore necessary.

We also needed to analyze the content of referenced web pages and remove from further analysis those ones, which are not related to the CFP events – it is quite usual that some of the links found in emails were referencing not-related pages. As we found about third of all referenced pages contained no valuable data for us.

In the case of emails we are currently checking for specific meta-tag present only in valid messages from our data source. In the case of web messages we have implemented experimental checking method based on list of most common CFP keywords and also actually found keywords in the previous emails, so if the page contains usually words like “call, for, papers, abstract, deadline, conference, event, participate” etc. we keep such page for the further processing.
4.4 Data parsing

The next processing layers are probably the most interesting. After filtering only relevant document, we are passing these documents into document parsers. These parsers are used for stripping – extracting short lines of text from the original document.

This application layer was also introduced in the later part of the analysis process. At first we were just processing specific document places containing valuable data, like email meta-tags containing specific key-words. Later we changed this method to be more generic not dependent on specific keyword. Finally we added the document stripping methods for generating simple document strips.

The motivation for introducing data stripping was also the fact, that dates shared common structure in email and in referenced web page with the only difference – on a web page the string with date was contained in a specific tag, for example:

<li>May 24, 2010<span>Camera-ready papers due</span></li>

Good idea is to have only one method for matching dates in a text strip, no matter what document was the origin of such string.

The output of our document stripping method is list of strips – each strip is a short line of text, without any tags or meta-tag headers. Each strip also contains additional information where was taken and how important the original location was. This information helps us store lines with no explicit information but possibly with important keywords such as email Subjects and web Titles.
4.5 Property generation

The fourth layer contains the core data extracting algorithms – on this place we are using all of ours data extracting algorithms on the entire list of new strips from the input. The output of these methods is a list of extracted values. We are currently able to find properties of type web address, email address, date and date interval, including related data string, and document title abbreviation.

Most of these methods use in the process regular expression for matching pre-defined strings or search for context-specific strings. Adding new methods to this process is also quite easy – all what it necessary is creating new class sharing the same processing interface and registering it in the property generating layer.

4.6 Data matching

The input in this layer is the complete list of found properties in the current document, all we need is to store these properties within a certain tag – our data object. In the early phases this process was quite simple – we were just creating new tag for each email from the mailing list – the name of the tag was extracted from a specific meta-tag “Name” in the email header.

X-Dbworld-Name: MoMM 2010

As we have found, using this name as the tag name is a good idea. In most cases all emails related to the same event contained the same meta-tag “Name” in email header.

We also use simple solution for matching properties from web pages to tags – each downloaded web page is strictly bonded to a specific event tag. If an email related to a certain event tag contains web addresses, we assume these addresses are then related only to this single tag. So during the data extraction process, if we find the web page useful, we add properties found on the page to the existing related tag.

When matching data, finding appropriate tag is not the only problem. We should be also able to match similar properties with slightly different text together. How this is done and what are the other possibilities is shown in the chapter 5.8 Data matching and pairing.
5 Processing documents

5.1 Processing emails

For downloading email messages we have implemented simple email client using a POP3 protocol. Using this client we are downloading email messages from our auxiliary email box, which is receiving emails from [2] mailing list. This client is not full-featured; it can download only complete content of email inbox or predefined number of first n stored emails in the box. It is not capable of deleting any email from the box or sending new emails but that is not an issue. Receiving of emails is done asynchronously in a loop. Each downloaded email is saved and then processed in our parser.

5.1.1 Filtering emails

We are currently using only simple check for identifying unrelated messages in our data source – we are just checking for the presence of “X-Dbworld” tag in the email header.

For processing other possible data sources of email messages it is necessary to replace this check with more advanced one, possibly with search for predefined keywords in the email body, checking the email sender with whitelist of email addresses or using other typical spam-fighting methods.

Another occasional problem when filtering and processing email messages is the variable format of the body. In most cases it is just plaintext but it could also be HTML or even RTF. In a sample of 1347 CFP emails, we have found that about 7% of emails use HTML content and about 5‰ is in RTF, so filtering those non-traditional messages should not bring any major problems into our results, see Figure 5.1. Another possibility is to process email messages with HTML body in the same way downloaded web pages are processed.

<table>
<thead>
<tr>
<th>Content type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaintext</td>
<td>1243</td>
</tr>
<tr>
<td>HTML</td>
<td>97</td>
</tr>
<tr>
<td>RTF</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 5.1 Type of content in emails
5.1.2 Analysis of email header

The first part we have analyzed in emails was the email header structure. Email headers usually contain several more or less useful data, such as the sender email address, servers used for transferring the email, some spam-checking results and also specific meta-tags. We have found that messages from our mailing list contain specific meta-tags with valuable data that we could use. These meta-tags start with “X-Dbworld” prefix and contain mostly information as name and abbreviation of the related event, one or more web addresses and also several important dates, see Figure 5.2.

![Example of meta-tags in email header](http://www.iwias.org/conferences/momm2010/)

For processing these meta-tags we have first used specific simple string lookup for most often tags like “X-Dbworld-Name:” and “X-Dbworld-Web-Page:”. We have basically created simple rule for each unique meta-tag we found. Using these strings we have found lines with those tags, we have then stripped from each line the tag and kept only the valuable content.

After several algorithm iterations we have found this solution rather incomplete and stubborn. We changed the usage of direct strings and designed proper regular expression instead.

```
(?i)X-DbWorld-(\w\-\w\-\w\-\w)+:\ (\.*
```

Using this regular expression which looks for “X-DbWorld-” then captures the rest of the tag name into first numbered group and the real tag value into the second numbered group, there is no need for other manual stripping and processing. During the process each found meta-tag is then used as data strip for the property generation.
When analyzing email headers, we have also identified several related problems. The first possible problem, when processing header, is probably the necessity to dig into these mostly hidden structures – the presence of such data in email header is not obvious. The other problem is that email headers might not contain any valuable information for us aside the subject and sender – it is all source-dependent feature. In our case the DBWorld mailing list is based on Mailman Mailing List Manager [17], which adds these tags into created emails automatically. Other mailing list systems might add similar meta-tags into sent emails too but manual analysis is here still necessary.

It is also a noticeable problem if some of these expected meta-tags are sometimes missing, for example tag “X-Dbworld-Name” is missing in about 9% of all emails, and we should therefore not completely rely on its presence and have a backup plan like using email Subject or found event name from the email body as the event tag name in further processing.

5.1.3 Analysis of email body

In the second phase of processing the content of email message, we are generating strips from the email body. In this case we are using a simple approach – just striping the body into separate lines of text. Each non-empty line of text from email body is then used as a new strip, but with the difference that strips not containing any valuable data are dropped from further processing.

Although this current method is quite effective, we might get better results with use of regularities found in the structure during the semantic analysis of email bodies. In CFP emails it is common to have most important values in optically separated boxes at the beginning or the end of the document. These boxes are either separated by blank lines, or by lines made of special characters like “=" or “*”. These boxes contain either only values of the same type, such as all important dates or most important information related to the whole event. Processing only values from these hot boxes is therefore a good idea – it could save us some time and resources when processing lot of messages and also lower the number of found false positive results. This optimization is also mentioned by authors of [18] and it is implemented in their work.
5.2 Processing web pages

Before we actually download any web page, we need to get the list of possible interesting web addresses.

As we have found, see Figure C.1, each email in our source contains about 2-4 unique web addresses in the email body and email header, so at the end of the data gathering process we got a list of addresses about 3 times longer than the initial count of email documents. When processing hundreds of emails, this gives us a lot of web addresses to check for related data.

5.2.1 Downloading web pages

For downloading the source code of referenced web pages we use the built-in .NET WebClient class in an asynchronous way – using 8 concurrent threads. If the downloading of a certain page for some reason fails, no document is downloaded and processed. If there is no problem with the connection new object of type web page is created and sent to further processing.

If some web page from the list is currently unavailable, it is not downloaded in the current iteration but it is checked again in the next web page downloading process – it works equally for emails – new emails are downloaded next time the inbox is checked.

5.2.2 Filtering web pages

Choosing between relevant and uninteresting web pages is more complicated than in emails. Using manual analysis we have found that about third of the referenced web addresses actually leads to a place not interesting for further analysis. It could be just some uninteresting referenced web page only with redirection code to the real home page, just a welcome screen, home page of some institute with no direct information about the related event, or even custom “Error 404 page”.

When processing HTML documents, there is also lot of related problems with the document structure. The main problem when accessing and parsing HTML is probably the non-validity of most HTML pages. About 60% of all web pages containing DOCTYPE is invalid, and about 30% of all pages works in a Quirks mode [19] without a specific DOCTYPE [20]. Some of the
problems are just small differences from the standard; some of them are major issues like non well-formed XML – missing or invalid tags or attributes, invalid structure and so on.

Also with the use of new Web 2.0 [21] features such as dynamic, on-the-fly loading, JavaScript processing, using AJAX [22], Flash/Silverlight elements or other non-trivial parts, it became sometimes hard or even impossible to gather all requested data directly from the source code. In our case we are processing just the bare web page source code without exploring any advanced content.

For facing the issue with lot of unrelated pages in our application, we are checking the content of the downloaded page for specific keywords – we are using check for keywords from a pre-defined set of most common keywords in CFP emails and already found words in our emails. A web page containing at least 4 of these key-words (experimentally chosen) is then used for further processing.

For identifying interesting pages there are also other possibilities – we might classify found addresses by some rules, like addresses located in email hot boxes are probably the most interesting for us, and addresses found in large text paragraphs are probably not so related to the event. Using this method for dividing found addresses into several priority queues is a good idea for future work. Also looking for other linked web pages in the content of already downloaded pages, searching for \(<a href=""> tags, is a good way for finding information related to the initial event. Sometimes the information about the event is not on the first page but instead on a sub-page like “More information”. The only problem with this process is a lot of new pages and even higher ratio of related/unrelated pages.

5.2.3 Analysis of web page body

Web page source code processing is a bit harder, because web pages are not written in plaintext but using XML-like structure. Also for proper understanding of the web page structure we need to perform manual semantic analysis of several typical web pages related to CFP events.

When processing web pages our goal is to find related data to already found values in emails. With this fact on mind, we have analyzed dozens of referenced web pages and found these common structures:

There is a major difference between emails and web pages – email purpose is based on one-shot semantics: to send message, inform, transfer important facts and forget the message. Lifespan of a single email message goes from
hours to days. On the other hand, the purpose of web pages is providing actual, long-lasting source of information. The lifespan of web pages, especially pages related to certain events, is between months to years. Their purpose is offering valid information in the entire event timeframe. Closely related to this fact is the graphical layout of message / page. The effort used for creating single message is mostly way smaller than effort used for maintaining certain web.

When analyzing web pages, we have found that important data are located mostly on several pages or places, not in one text block as in emails. The target information is located mostly in specific html tags like this:

```html
<li> ... </li>
<p> ... </p>
```

Or in combined tags like this:

```html
<tr>
<td> ... </td>
<td> ... </td>
</tr>
```

The important data are also located in the typical important html tags like `<title>`, `<h1>` or `<meta http-equiv=...>` in page header.

After analyzing tens of referenced pages we have pin-pointed about 6 most common places worth further processing. For these places we have created specific regular expressions for stripping the contained value in each predefined tag as a plaintext. Such stripped values are then processed in the same manner as striped values from email messages.

The good part of this method is the simplicity of targeting specific places in HTML, the drawback is the need to design each regular expression manually – it is exhaustive and not so flexible.

Another way here how to extract target data is using actual HTML parser for checking the target HTML tags. We could use either the .NET built-in XML parser or use some third party parser designed especially for parsing (mostly invalid) HTML.
5.3 Data extraction

While using the previously mentioned methods we are able to find and extract strips of text containing possible interesting data from each document. These data strips are then processed using various data parsing methods. Each method was designed for finding specific type of information like web address, dates, date intervals, email address or the titles. For designing these methods we have mostly used specific regular expression with specific post-processing.

Because of the nature of these algorithms it is easy to modify each method separately and also to use all methods for processing strips from different data sources – there is no need for separate methods for matching dates in emails or web pages.

5.3.1 Extracting web addresses

One of the most important values in each CFP message is the web address pointing to the web of the related event. As we have found, average email message contains about 3 web addresses, excluding addresses in email header and blacklisted addresses, see Figure 5.1. When analyzing web pages for contained addresses we get a larger count. There are tens of addresses in each web page but mostly they are not referencing new web pages, only used images, JavaScript and CSS files – these types of addresses are not processed any further.

When extracting web addresses we are not accessing directly the content of emails or downloaded web pages – we are processing only gathered data strips. For extracting web addresses from the strips, we are using regular expression for matching the common format of all web addresses – it starts with the leading part http(s)://, mostly followed by www and then by hostname and domain, sometimes even followed with port number and the trailing part.

```
"(?ix)
\b
# Match the leading part (proto://hostname, or just hostname)
(  
# http://, or https:// leading part
(https?:://([-\w]+\.[\w]+([-\w]*))  
|  
# or, try to find a hostname with more specific sub-expression
# omit host-names from emails
```
When designing the best regular expression we need to take into our mind all mentioned variations – with too simple expression we might miss many addresses, or there is also a chance to match only a part of a specific address. Such partial match could bring us to a generic server page, not to a specific page devoted to the event.

With designing more and more complex regular expressions we might also come to another problem – false-positive matches. Sometimes wrong constructs in the original documents might be matched by the regular expression; even though they are not web addresses. Good example is a false matching of group of words separated with slash: “…deadline for accepting abstracts/drafts/samples of papers...”.

During the process of designing our regular expression for matching web addresses, we have testers several different expressions with various results. The final one, see Figure 5.3, is pretty complex and yields best results so far. After small final tweaks for excluding domain names from email addresses we are now able to match more than 99% of all contained addresses with only tiny number of false positives.
5.3.2 Web address validation

Validation of found web addresses is an important process. In the early phase of development our regular expression for finding addresses was rather poor so there were a lot of invalid or incomplete found addresses. The regular expression used currently is quite stable but still we need to validate addresses if they are pointing to a live web server.

After finding each address we are using several procedures to make sure each address is valid. Each web address found by the regular expression is normalized with adding the leading part, if it is missing. Also all addresses ending with slash are stripped of this character – this is a precaution, we do not want in our results two web addresses with the only difference at the ending slash. The third check is a check against our blacklist of addresses. Currently there are only two addresses in this blacklist:

http://www.cs.wisc.edu/dbworld
https://lists.cs.wisc.edu/mailman/listinfo/dbworld

These addresses are present in each email from our source at the bottom of the document body, these are links to the home page of this mailing list and we definitely do not want them in our results.

At the end a new Uri object is created for each found address using the built-in method Uri.TryCreate. This method is our last resort for finding web addresses, which are not written correctly.

Each web address found by the property extraction process is also probed using our WebAddressChecker service. This service uses the WebRequest class for checking whether the target server actually exists and responds in a reasonable amount of time, in our case 10 seconds for each check. If the Http request does not responds in a given amount of time, the address is thrown away and is not used any more. If the request is successful, the address is added as a new data source into list of available web data sources.

In the case the address of the server is valid but the server is just currently unavailable it is possible to re-run manually this procedure. All addresses previously thrown away are checked again using the same semantics.
5.3.3 Extracting dates / date intervals

Other important values located in CFP emails are the dates or intervals of dates, see Figure 5.4. In comparison with web addresses, we do not need only bare dates from our documents; we also need to find what kind of events or deadlines are related to these dates.

IMPORTANT DATES
======================
Paper submission: April 21, 2010 (23:59 Hawaii Time)
Notification: May 21, 2010 (23:59 Hawaii Time)
Camera-ready version: June 7, 2010 (23:59 Hawaii Time)
Workshop day: July 5 or 6, 2010

Figure 5.4 Example of dates in typical CFP

When checking dates from multiple emails related to a single event, it is also an interesting finding that these proposed values are sometimes not fixed. It is common that some of the terms are, due to various reasons, extended and new date is sent in a new message. In this case we need to figure out, how to update our data in based on the value in the new email. This updating and pairing is mechanism described in chapter 5.8.

When looking for dates we do not have such comfort as when looking for web addresses. Dates are often written in many different formats, according to national or regional specifics. Sometimes the date is written using only numbers, sometimes using names of the month. It is also quite common to use intervals of dates instead of just one date.

For better understanding the semantics used for writing dates into documents, we have analyzed several messages manually and found these common rules that apply for most scenarios.

Mostly there is only one date or one date interval located on a single document line. Also in vast majority of all email messages we came across the month was written using English name for the month instead of just the number. There is a good reason for this – there are different regional rules for writing dates using just numbers. In the United Kingdom the short date format is “dd/MM/yyyy”; in the United States it is “M/d/yyyy” and there are even other more confusing date formats in most of other countries. So writing a date for international audience using just “3/8/2010” is rather confusing; using “3rd of August, 2010” is much better. Another finding is that majority of dates in documents contain the year as well – there is a simple reason for this,
mostly the first CFP email is sent way ahead the term of the event, often one year or more. With this keeping in mind it is therefore a good idea to include the year in the message as well, for avoiding confusion.

What is also interesting discovery – dates are located in most emails together in a special *hot blocks*, so targeting only those hot blocks is a good idea.

Taking into account our findings, the first idea how to find dates and associated data is based on lookup for specific strings of text – numbers for years, names for months and numbers for days.

Our implemented method works in this way – each found data strip is scanned using regular expression for year number, month name (full word or just the month abbreviation) and day number. After selecting all such strips we are trying to identify the right triplets in each strip. This is not as easy as it sounds – this process should identify the occurrence of multiple dates, such as date intervals, and also exclude all false positive results, like in this case:

*CIKM 2010, Toronto, Canada, October 25 to 30, 2010.*

Matching procedure finds in this line two year numbers, one month name and two day numbers. The result of this data strip should be date interval 25. – 30. October, 2010 and the relevant info should be “*CIKM 2010, Toronto, Canada*”. This is accomplished with a smart algorithm calculating the distance of substrings in each possible triplet on each line. The shorter the distance is, the higher is the chance the date is proper. In this example there are four possibilities how to pair year, month and day together. The smallest distance got the substring “October 25 to 30, 2010” – this string is used for constructing the date interval and the rest of the string is used as the relevant information.

After creating DateTime object with the extracted values and validating it, there is a next task – we need to find the relevant text to this date. This is done by checking if there is more text on the right or on the left side of the date – the longer text is used. This text is stripped of special characters and the result is the final information relevant to the date, see Figure 5.5.
Even though this method works in most situations, there is still place for further work and improvement. Also our methods are tuned especially for formats used in our CFP emails and related web pages, it might therefore not work properly for other different data sources.

Future work on date matching should probably focus on these areas: better handling of date intervals (using more advanced context based lookup), support for other languages (updating the regular expression with month names from other languages), support for month numbers (this is actually simple to implement, the only problem is identifying which number is day and which is month, maybe using some heuristics), looking for dates only in selected hot boxes (find information about line numbers with dates and process only the large group of lines) or creating more robust, context-based method for identifying the related text to each date (handle typical delimiters when parsing the relevant text).

In this case is also good to mention method used in [18] – simplified Bayes classifier for putting found dates into preselected classes for most common types of dates in CFP messages, see chapter 5.4.2.

### 5.3.4 Extracting email addresses

Another type of important value found in most CFP emails is the email address used for contacting the organizing institution. When we were designing our data extraction goals, we have thought that email address is a type of information present in most of emails, even in multiple occurrences but later after designing and testing our extraction rules we have found out that it is common to have only one or even no email address in most of input emails. That was an interesting finding.
Extracting email addresses from our document sources is actually quite simple comparing to web address or date extraction methods. Email addresses have a more specific format and designing proper regular expression was not hard, see Figure 5.6.

The extraction process works similar as web address extraction, the only difference is no need for further validation. Currently we are also capturing only the bare email address from the document but it is probably a good idea to capture also the information what is each email related to.

```
"(?ix)
\b([a-z0-9!#$%&'*+/=?^_`{|}~]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~]+)*\@\b(\:[a-z0-9]+\?\:[a-z0-9]+\?\:[a-z0-9]+\?\:)?(?:[A-Z]{2}|aero|asia|biz|cat|com|coop|edu|gov|info|int|jobs|mil|mobi|museum|name|net|org|pro|tel|travel|arpa)
"
```

*Figure 5.6 Regular expression used for matching email addresses*

### 5.3.5 Extracting event titles

When extracting information from some email or web page, the probably most important information is the name of the target event or message topic. This event name or event title is mostly well defined and unique across all similar messages. It consists of several words, sometimes of numbers too. Identifying the real and complete name of such event is the key area of our information extraction, because we use the event name, especially the event abbreviation, as the identification string of such event.

After manually analyzing semantic structure of tens of documents we have identified several key places where to find the proper event names. In most CFP emails the event name is located as abbreviation in the “X-Dbworld-Name” meta-tag in header; also event name or abbreviation is present mostly in email Subject. In the email body the event name and/or abbreviation is present within the first few lines in the document or in case of HTML pages the event name is mostly contained within the *title* or *h1* tag.

Extracting event abbreviation from email messages was quite simple – about 91% of all emails contained the “X-Dbworld-Name” meta-tag in email header. This tag contained the proper event abbreviation in most cases. Also the used abbreviation was mostly the same for each email referencing the same event. We have also found that this abbreviation is present in about
40% of emails in the document body – it was located within the first 10 document lines and wrapped in round brackets, for example:

Mediation Services in Computing Environments
(MeSC 2011)

As we have found, there is only a small chance for matching false positive result within the first 10 document lines when using regular expression for matching event abbreviation but sometimes there is more than one used abbreviation, mostly referencing other similar events.

This event name abbreviation found either in email header or in document body is used as the unique identifier of each CFP event. For matching document related to the same event we have also introduced a small tweak – all found event abbreviations are stripped of all special chars. Consider names like “PDC2010” vs. “PDC 2010” – two different strings for the same event. Using this stripping method we are able to match more documents with the proper event without actually losing much information.

As far as extracting event abbreviation is pretty simple, extracting full event name from the document is a harder problem. The variability of places used for positioning the event name was too big and also there was not any typical layout used for event names.

Full event name extraction was not implemented because of the complexity but we have identified several strong points possibly applicable in the future work – the full event name is located within first 10 lines in about 80% of all documents or within first 5 lines of about 60% of all documents. Most of these lines are either empty or containing only special characters like “*” or “=” or common sentences like “Call For Papers” or “Apologies for multiple posting...”. So when looking for the full event name and excluding previously mentioned lines (creating exclude rules for each mentioned type) we could get from each document about 2-3 unique lines containing highly probable the full event name. Using some statistics or heuristic based algorithm we might get the real event name from these 3 lines with a proper chance to success.
5.4 Other remarkable methods

Recently we have discovered that a similar work to ours was started. The paper [18] is targeting a close sub-area of our work – gathering dates and places of events from CFP emails, and the current status of this work is “work in progress”. The authors are describing two interesting data extraction techniques.

5.4.1 Gate online

The first described method uses the Gate online component, which incorporates the GATE framework [23]. Gate online works as a web service and is useful in many areas of natural language processing. When using predefined parameters and scripts written in JAPE language [24] as inputs for the Gate online service it is possible to retrieve annotated XML structure with extracted tags and content from the input documents. The authors are currently able to extract information like important dates with places and names of persons and hope for improvement these methods in future work.

5.4.2 N-gram based extraction

Another described approach in this paper is about extraction dates from plain text documents using n-grams representing the date structures and classification of related data using modified Bayes classifier. As the result, they were able to automatically retrieve and classify about 90% of dates from the CFP emails and assign them to proper classes in comparison with manual email annotations.

Although these two methods targets only a specific data source and both methods are designed only for processing typical CFP events, they have shown an elegant way how to process this type of data source. Instead of designing simple parsing methods they have used a thick framework for processing their input. We found especially the second method with Bayes classifier really interesting and it might be a good idea to focus on such classifier algorithm in the future work.
5.4.3 Semi-automatic machine learning

When designing our currently used methods we have also thought about different way how to gather data automatically – using user-assisted machine learning and classifying.

This method is based on several ideas – first one is the idea, that the application should not need predefined regular expressions for gathering important data, the application just needs to know what kind of important data wants the user himself. The user will teach the algorithm by manually selecting each important string from input emails and after several learning iterations the algorithm should be able to pick similar values from the text on its own.

For the start the application needs a thorough framework for generating statistics for each word, line, group of lines and document. The statistics framework should also generate metadata like the common neighborhood, typical words in current selection and typical words and lines before and after the selected line. The program should also identify the composition of currently selected word – characters, capitals, numbers or special symbols. After providing this kind of information for each word or line we could start the user assisted learning. Basically user will manually select important words in each document; the algorithm will generate statistics data for each selected string and create classifying rules representing the common signs of selected data.

With such learned information the framework should be able to select similar data from new incoming documents by classifying all input words and lines and picking those ones with high computed value. Also, in the case of wrong selection, user should be allowed to fix the selection and give the framework another valuable feedback.

This idea basically incorporates similar mechanism used in association rule learning [25], which is used for discovering interesting relations between variables in large databases using rules like “if item X and item Y is bought by the user, then also the item Z is bought”. In our case of document processing, the generated rules might look like “if word is composed only with capitals and is located within first n lines, then it is selected”.

Although this proposed method is rather complicated to implement, it is an interesting idea what area of data extraction to explore in future work.
5.5 Common extraction problems

Even though we are able to gather most of data from our input documents, there is still lot of issues common to most of input documents and sources.

5.5.1 Types of errors

There are basically two types of errors occurring in the data extraction process.

*Errors in the original documents* – when processing any type of document, there is the usual chance that the original document contains several errors. The errors could be just small misspelling issues but also factual errors like invalid date of the related event, missing important facts or swapped values. If the error is purely semantic such as invalid date of the event, bad year, missing or invalid name of a guest, there is only a limited way how to fix it. The first possible method could be identifying such errors in the first place and not including such invalid facts to our results. The other possible scenario is to search for other relevant sources, like information from the related web pages, and find the valid values there.

During our processing we have faced lot of these errors in the original documents. The most common errors were different abbreviations used for the same event, factual errors in the target dates like bad year – 2010 vs. 2011 or invalid months or even year number 1020 instead of 2010. Other typical errors were incomplete or misleading web addresses or wrapped document lines damaging the original document structure. Most of these errors were also fixed in the next email related to the same event.

*Errors made during the processing* – these ones are created during the process of data gathering – they could be caused by unexpected data format, improper document layout or even error in the algorithm. The result of such error could be either false-negative results – items missing in the results or false-positive results like invalid found values. Lowering the rate of such errors is possible for example by implementing proper unit tests for testing unexpected input data, testing expected results and current results.
5.5.2 Different data formats, country-related specifics

Typical problem when processing emails or any other type of data documents is with regional or country-related specifics – the way document is structured, what kind of date, time, number, currency format is used. One way how to face this problem is just by expecting all documents in the same format but this is mostly not acceptable. The other, opposite way how to face it is by creating variants of the algorithm for all most likely occurring input formats. This might be exhaustive but mostly it works.

The problem on its own is when some documents are written in whole other language. We could just ignore such document or possibly identify the actual language and use some automatic method such as Google translate for translating such document. In our case this is not needed, all emails we have seen were written in English.

Another problem with handling documents from many sources is sometimes the problem with different default document encoding. Identifying other encodings than the most common UTF-8 is possible either by the information in email/web header or manually by checking the occurrence of special characters in document but sometimes it is just easier to handle all documents in UTF-8.

The last noticeable problem is the document format. Currently we are processing only plaintext or HTML emails or web pages but for broader data gathering we might want to process also the referenced or attached documents. In this case we should be prepared for processing wider range of document formats, like PDF (Portable Document Format), PS (Postscript) or DOC(X) Microsoft Word document. For accessing or processing such documents we might use some 3rd party API or just some tool for converting the whole document into plaintext.

5.6 Time factor of gathered data

When analyzing data from different sources there is a problem we call the “time relevance factor”. The time relevance factor is in other words the problem that some documents are stable but some are periodically changing. Stable documents which could also be called snapshots are in this case such documents, which are once released and never changed afterwards. Emails are in this case good example of such stable documents. The changing documents are mostly documents accessible online only like web pages. Web pages could
change dramatically over just a small period of time so monitoring of new versions is therefore necessary.

Speaking of time factor, we could also find some kinds of hybrids in this case. The best example is the Wikipedia. Every page could be seen as dynamic data source, changing from time to time but also any page could be seen as the newest document in a row of stable documents – all previous versions are still accessible.

With this time relevance factor is closely related the next problem – how to update, add or replace changing data.

5.7 Problem with data adding/replacing/removing

When processing data and storing them into tags representing the related event, we need to know how to handle updates of existing values, for example the “paper submission” date has been changed or there is a new web address available. In the first case we should just change the stored value with the new one; in the other case we should add the new value into the tag.

Basically there are three possible types of update:

- New information replacing old obsolete value, for example a “paper submission deadline”
- New information adding new value of the same type, for example a new web address
- New information removing previously introduced information, for example new, shorter list of invited speakers

Implementing automatic algorithm for identifying types of update is not an easy task, because this is purely semantic problem. In our algorithm we have implemented manually several rules for handling updates of each type of document property:

- In case of new web addresses, just check, if this address is already present in tag. If not, we add it as a new property. If there is such address, we save the new one as a duplicity.
- In the case of new date, each date also has the information about event related to the date. In case there is already a date with the same information and the new date comes from more actual email or web page snapshot, we mark the old value as Obsolete and the new one as New.
In the case of plaintext information (like those ones gathered from email header), the same rule is applied as for dates.

This process is pretty straightforward but it has several drawbacks. For example we suppose if the new information got different value than the previous one, the previous is no longer valid – what if the new one contains erroneous information? Then the valid one is replaced with the new, invalid one. In our algorithm we have solved this problem using a smart approach. By default all found properties are stored in the target tag but only the most actual and unique ones are show. User is also allowed to unhide all hidden properties to see changes during the history of each event.

5.8 Data matching and pairing

As we have shown so far, we are able to process emails and web pages from various sources, search these documents using several rules designed with the respect to the identified semantics and process and store found properties in our data model. But there are still two major problems – matching of documents related to the same event into single event tag and data pairing of properties related to the same event tags, for example properties like “Paper submission deadline”.

As we have shown in the chapter 5.1, it is not a rare situation to find two or more emails related to the same event tag but using different abbreviation for the event in email subject or meta-tag. Also the property names related to event dates are sometimes different in CFP emails and on referenced web pages. At the result we got several similar properties in our tag properties table but what is the proper way of matching same values within one tag property?

We have used as the reference name the event abbreviation for matching documents related to the same event tag. This was found as an applicable solution; about 90% of all documents were matched to the right event tag and in the case of misplacing some document, user has the ability to merge one tag into another one. We have also used a simple tweak for increasing the success ratio – when comparing new abbreviation with existing ones we are stripping all non-alphanumeric characters from the string and also ignoring character case. With this tweak we have lowered the number of created unique tags for 1542 emails from 916 to 842 tags – about 6% of previously created duplicates were merged into existing tags.
For pairing tag properties there are several ways how to do it. The simplest one is just keeping all unique property names and not matching them into specific groups. This is an applicable solution for highly structured documents or for situations with only limited number of properties in each email and for each event. The other way is using some similarity check like looking for existing property containing long enough substring similar to the name of the new property.

Another interesting approach is shown in [18]. The authors have used modified naive Bayes classifier to automate the process but the drawback is the solution is too tightly coupled with the CFP documents. The authors have basically created predefined classes of date events which are most common in CFP documents. Then they found keywords on each line with a date and used modified Bayes classifier for assigning the date into the date event class with highest computed value based on the relevance of keywords to each of classes.

This method was found as highly successful in assigning dates to proper classes, the only drawback is the necessity to manually preselecting all target classes. For other type of partially structured data sources it is necessary to create new set of rules and new target classes.

An idea how to automate this process is using machine learning – user will create the actual target classes manually in the application, not in code, and then, with a help of user-assisted learning, he will teach the classifier which keyword sentences assign into which classes. This is probably the most remarkable method worth of implementing in future work – it uses predefined expression lookup for finding dates (or other properties) and relevant data in documents, and then it combines the advantages of machine learning on a limited set of data with limited set of rules for matching the value with a proper target class.
6 Summary of results

Summarizing the capabilities of our application we are now able to download emails from any user defined email box with POP3 protocol (currently only using unencrypted connection), save all emails in our object model and analyze them using our data extraction methods. These methods consist of data filtering – removing all emails uninteresting for our analysis; stripping email body and email header to short text strips; property generation on found strips using web address and email address regular expression lookup, smart date lookup using various regular expressions with considering the semantic structure of typical CFP emails and also lookup for event name abbreviation in top part of each email; matching found properties to target event tag and replacing older values with new ones found in more actual documents. During this data extraction process is also running the web address checker service capable testing all found web addresses if the referenced web server is accessible.

In the second part of processing we are able to download the content of all referenced web pages, filter pages not containing any relevant data using a simple filter and strip values from predefined HTML tags with the respect to the most common semantics of web pages referenced from CFP emails. These strips are then processed using same property generating mechanism and found values are then matched with the appropriate event tags.

The resulting data model could be saved and opened later, updated with current emails or values on updated web sites.

We have also proposed possible improvements in each area of our work, either small possible improvements relative only to our data source or generic proposed methods, even experimental ones, capable improving the overall gain of our application.

6.1 Ideas for future work

Although our implemented application is capable extracting most of important data stored in CFP emails, there is still place for further improvements.
Improving extraction and filtering rules

The first place for improving the overall results is by enhancing already existing data filtering and property generating methods. For example there are few uncovered scenarios when parsing date intervals from complex line sentences which could be covered by more detailed parsing rules. Also the extraction of full event name proposed in chapter 5.3.5 was not fully implemented. Another idea for improving the overall results is gathering and checking web addresses from downloaded web pages – digging deeper to the actual web structure.

Another possibility is also adding rules for gathering new types of data, like names of cities related to each event and names of referenced people. Parsing these values should not be hard using English dictionaries of most common names of cities and people.

User scriptability

In current implementation it is possible only to change extraction and filtering rules directly in application code. Future work could cover implementing user interface for modifying regular expressions and other extracting rules directly in running application and saving such rules together with the project.

User assisted learning

In chapter 5.8 we analyzed existing method for data classifying and proposed a new one for defining data source specific extraction classes and a way of teaching the classifier which strips of data to assign to which classes. This is probably the most interesting topic for further work, the possibility of defining source-specific classes for automatic data gathering and saving these rules and classes together with to project.

Export into new data formats

Although the current data format used for storing project status is not compatible with other applications, the actual project object model is ready for saving into other formats. Implementing saving into formats like SQL database or even RDF file is therefore just a matter of creating new serialization capable classes.
7 Conclusion

The goal of our study was to explore the possibilities of analysis of partially structured data sources such as emails or web pages. In the first part of this work we described existing methods and proposed their possible use in our work and also explored more advanced approaches of semantic-based document processing.

In the next section we described the specific data source we used. Using manual semantic analysis of metadata and the most frequently occurring structures in the mailing list, we found interesting places for data extraction, for which we designed a set of rules and the most interesting of them implemented as automatic algorithms in our framework. Furthermore by analyzing other possible ways of optimization, identifying potential problems and applying additional adjustments to our methods, we managed to achieve applicable level of automatic data extraction, which can be used in practical use. We also proposed several ways for improving current methods in future work.

In the second part we focused on utilizing gathered web addresses for additional search of related data. In the first phase we implemented a system for filtering uninteresting and misleading addresses, in the other phase we proposed and implemented methods for obtaining valuable data from referenced web pages. The result of our work is application capable analyzing data from specific mailing lists, which can be easily adjusted for the analysis of other types of data sources. We were also able, although just partially, to analyze the content of linked websites and complement earlier data from emails with updated information from the Web.
8 References


A. Attachments

Application framework Seeqer

The attached disk contains source code and binary executable of the application framework Seeqer [28] we have developed. Application framework Seeqer was designed for simple testing of our designed methods. It is based on a structured data model which is serialized and persisted in XML, application logic containing various data gathering methods like POP3 email downloader, web page downloader etc. It also contains parsers for filtering and extracting data from gathered documents.

The UI of the application is built upon WPF. It consists of simple tab and panel layout and it is designed mainly for testing purposes.

For further instructions how to use the application please read the User’s Guide describing the installation prerequisites and requirements.

Application documentation

Application documentation consists of three parts – first one is generated MSDN style documentation from XML comments in source code. For basic understanding how the framework works this is a good place to start.

The second part is User’s Guide designed for easy start with the application. It describes the UI parts and also the basic workflow used by the application – how to create project, connect to a POP3 server, download emails and check the results.

The last part is the Architecture overview manual designed for exploring and improving the application infrastructure and implemented methods.

Test data and projects

The attachment also includes folder with test data for the application – small and large set of dumped emails and also an empty project and small preprocessed project with emails and downloaded web pages.

The last part of the attachment is this thesis as Adobe PDF document. The current version of the application, documentation and test data is also available online on the project website [28].
B. Example application run

These are the example results when running our application with 300 CFP messages received during 25 days loaded from a file:

- 8 seconds took the processing of these emails, including header processing, address lookup and date lookup on Core2Duo 2.3 GHz.
- 225 event tags were created during the process, including 8 un-named ones.
- 1222 web addresses were found during the process, 743 unique, excluding blacklisted ones. From this number 208 was gathered from email headers, 535 from email bodies.
- 1545 dates were found during the process. From this number 485 was gathered from email headers, 1060 from email bodies.
- 225 emails were found in email bodies, including duplicates.
- 154 abbreviations were found in email bodies, including duplicates.
- 1126 pieces of other plaintext information were gathered during the process from email headers.
- 598 unique addresses responded using WebRequest testing within the 10 seconds timeout for each request. Using 8 threads this test took about 115 seconds.
- 581 unique web documents were downloaded using WebClient within 10 seconds timeout for each request. Using 8 threads this process took about 40 seconds.

The Figure B.1 shows the typical event tag grid from our application – we can see gathered web addresses, dates and other information from email headers and bodies. This example also illustrates the situation, when event Deadline was extended.
C. Document statistics

During the data processing we have gathered statistics from about 1300 CFP emails. In the Figure C.1 only addresses and dates from email body are counted, including duplicities, excluding HTML emails – there was too many unrelated web addresses in each email message. The statistics generation of length, lines and words count was originally created for the forthcoming machine-learning classifier but it was never actually implemented.

It is also possible to generate similar statistics for each active project. More information about statistics generation is in the User’s guide.
These generated box plots show median line between 1\textsuperscript{st} and 3\textsuperscript{rd} quartile. The error bars represent 9\textsuperscript{th} percentile and 91\textsuperscript{st} percentile.

![Box plot for Web addresses and dates](image1)

**Figure C.1** Addresses, dates and length per email statistics

![Box plot for Words and lines](image2)

**Figure C.2** Words and lines per email distribution