Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

SEN

Software Engineering

Software ENgineering

Identification clouds: aspects of the implementation

C.L. Blom, M. Hazewinkel

REPORT SEN-E0508 AUGUST 2005

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

Identification clouds: aspects of the implementation

ABSTRACT
This note is concerned with the automatic assignment of keyphrases to (short) documents, for instance abstracts. The assumption is that there is a standardized list of keyphrases that can be used. The emphasis is on implementation issues.

2000 Mathematics Subject Classification: 68P20
Keywords and Phrases: information retrieval, identification cloud, automatic key phrase assignment, parsing

Identification Clouds: aspects of the implementation

C.L. Blom, M. Hazewinkel

August 15, 2005

Abstract

This note is concerned with the automatic assignment of keyphrases to (short) documents, for instance abstracts. The assumption is that there is a standardized list of keyphrases that can be used. The emphasis is on implementation issues.

1 Introduction

The basic premiss behind this note is that it is important to be able to assign keyphrases to documents, and to do that more or less automatically. Also that the key phrases should come from a controlled (= standardized) list. The second author has argued this point extensively, see the references.

Let it be noted again that full text search, including approximate string matching, is but a poor beginning when looking for information. One is almost certain to find something, which may give the illusion that things are going well; what one finds is, however, almost surely contaminated, distorted, etc; also seriously incomplete. This is not the place to argue this point further, but see [4, 5]. Here we are mostly concerned with the matter of automatic key phrase assignment to documents, given a good large set of documents and an adequate list of keyphrases for a given field of enquiry. For all of mathematics, according to several independent calculations, that means something like 200000 keyhrases. And compiling such a list, plus the additional data needed to make good use of them raises other problems which are beyond the scope of this note.

Here we are mostly concerned with the matter of the implementation of an automatic keyphrase assigner given that all the necessary data are in place.

2 The idea of Identification Clouds

Basically the *"identification cloud"* of an item from a controlled list of standardized key phrases is a list of words and possibly other phrases that are more or less likely to be found near that key phrase in a scientific text treating of the topic described by the key phrase under consideration. For instance the key

phrase

Darboux transformation

could have as part of its identification cloud the list

soliton
dressing transformation
completely integrable
Hamiltonian system
inverse spectral transform
Bäcklund transformation
KdV equation
KP equation
Toda lattice
conservation law
inverse spectral method
exactly solvable
...
37J35, 37K (the two MSC2000 classification codes for this area of mathematics)
...

Of course the definition as given here is still pretty vague: both the terms 'more or less likely' and 'near' need to be made more precise. Before saying something to that point let us look at a concrete example.
Consider the following record that the second author saw some five years ago:

" ... using the Darboux process the complete structure of the solutions of the equation can be obtained.".

At first sight it looks like there is here a natural key phrase, viz. "Darboux process", to be extracted. Presumably, some sort of stochastic process like "Cox process", "Dirichlet process", or "Poisson process" is meant. The context made that rather doubtful; the surrounding sentences did not have in them the kind of words one expects in a paper on stochastic matters. The proper name "Darboux" is also not sufficient to identify what is meant; there are two many terms with "Darboux" in them: "Darboux surface", "Darboux Baire 1 function", "Darboux property", "Darboux function", "Darboux transformation", "Darboux theorem", "Darboux equation",....(these all come from the indexes of [6]).

The various words and phrases occurring in the surrounding sentences settled the matter. These included such words as 'soliton' and others from the example above and are typical typical for the surrounding words of the term "Darboux transformation" and typical for the area classified by 37K (and 37J35) (one of the classifications-indeed the main one-of "Darboux transformation"). Thus the 'identification cloud' of the term "Darboux transformation" made it possible to

extract the right term. What the authors meant is that repeated use of the process 'apply a Darboux transformation' should give all solutions.

A human mathematician, more or less expert in the area of completely integrable systems of differential equations, would have no difficulty in recognizing the phrase "Darboux process". Thus what identification clouds do is to add some human expertise to the thesaurus (list of key phrases) used by an automatic system.

The idea of an identification cloud is part of the concept of an enriched weak thesaurus as defined and discussed in [1], [2] and [3]. This will not be discussed further here. Let us just remark that the idea can be extended: for instance there can be identification clouds of formulas and of MSCS2000 nodes. Indeed an identification cloud of a MSCS node gives that node content and meaning far beyond the terse descriptions in the MSCS2000 itself.

3 The TRIAL-SOLUTION project

The basic idea of the TRIAL SOLUTION project (IST-1999-11397: Febr. 2000-May 2003) is the reuse and individual retayloring of textbooks in the following way. Take a number of textbooks, slice them into suitable chunks, called slices, and fit together a collection of slices to obtain a text individually taylored to the job or individual at hand.

Besides the basic text of a slice many more data are needed to make this possible. One of these is a set of metadata in the form of key phrases. Within the project therefore there was needed an automatic key phrase assignment software package that works on the idea of identification clouds. As is indicated in the example above one cannot rely on '(approximate) string recognition' to find all relevant standardized key phrases.

One thing that came out of the project was that the idea of identification clouds needs further refinements; for instance weights and even negative weights (to rule out spurious key phrases). For details on this see [4] and [5]

3.1 Components of the TRIAL-SOLUTION project

The software package for the TRIAL-SOLUTION project consists of the following components:

1. Splitter. The Splitter decomposes a documents (e.g. mathematical textbooks) into semantical units called slices, stores each slice in a separate file and produces an XML formatted file (called "imsmanifest.xml") that contains a complete description of the original text: in which chapters and paragraph slices of text occur, how the relate to each other, and in which file a slice is stored. A slice can be as small as a single line (e.g. a mathematical definition) or it may span over several pages (e.g. a complex mathematical proof).

2. Automatic Keyphrase Assigner (AKA). The AKA takes a sliced document and assigns keyphrases to relevant slices by adding these as extra information (meta-data) to the "imsmanifest.xml" produced by the Splitter. It doesn't modify the original text.

3. Meta-data Server. The Meta-data Server maintains a database called "thesaurus" containing all keyphrases used within the entire project and checks the keyphrases for consistency, dependencies, uniqueness etc.

4. Authoring Tool. The Authoring Tool can be seen as a human interface to manage the other components of the project, e.g. manually adapt the structure generated by the Splitter, revise meta-data assigned by the AKA, edit content of slices. It is intended to be used for small modifications in the thesaurus and the enriched texts by expert users.

5. Delivery Tool. The Delivery Tool is the human interface for the end-user (e.g. college-students). A query about a subject is matched with a database with many sliced and annotated books and those slices from many sources are then formatted and presented to the end-user.

All components exchange information in a standardized way, by means of XML encoded files with Data Type Definitions (DTD's) or XMLSchema's.

4 Automatic key phrase assignment software.

The "Automatic Key Phrase Assigner" deliverable of the TRIAL SOLUTION project is a piece of software that pictorially can be depicted as in Fig. 1.

Thus, there are two inputs:

- a 'sliced text'. That means a scientific text, say an undergraduate textbook on analysis (such as were actually used during the project), sliced, that means cut-up in chunks that are small enough to be coherent and reuseable and large enough to have internal meaning. The sliced text comes in a special packaged format that is universal for all TRIAL SOLUTION components.

- a list of key phrases together with for each key phrase its identification cloud. This concept will be described in more detail below in section 2. But, roughly the identification cloud of a standard key phrase is a list of words (and maybe very short other key phrases) that one expects (often) to find in the neighbourhood of the key phrase concept that is being discussed.

There are also two outputs

- the same sliced text, packaged in the same way, enriched with key phrases that have been assigned to each slice.

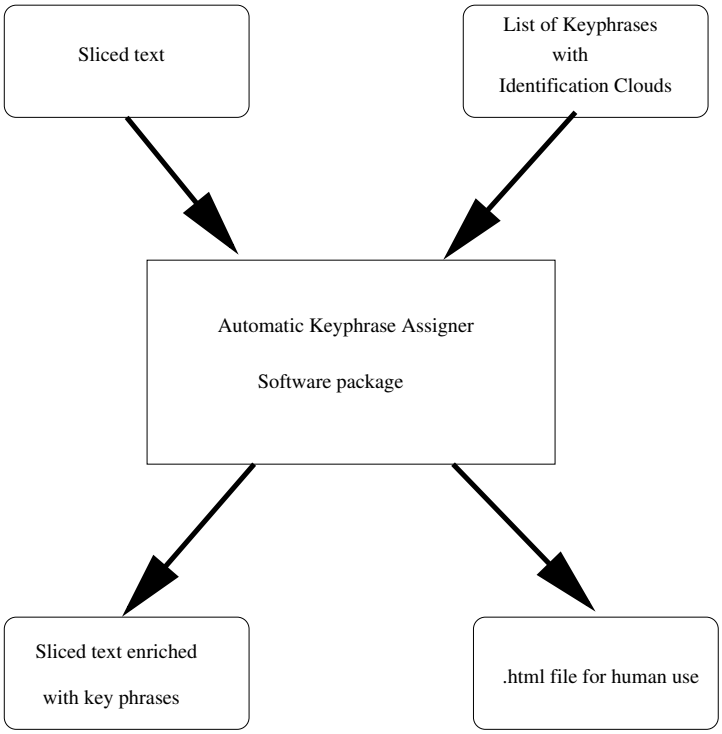


Figure 1: Automatic Key Phrase Assigner

- a `.html` file for human use that is to be used for expert validation of the the automatic key phrase assignment. This one, when displayed, highlights the key phrases that have been found and also gives the 'evidence' for assigning them in terms of the items from the identification cloud that were found and the percentage (weight) of the identification cloud that was found.

Most of the remainder of this note is about the internal structure of this piece of software as designed by the first author.

4.1 Keyphrase Recognition

Let us illustrate how the software works by an example Consider the following keyphrase definition (simplified, in German language):

```
<KEYPHRASE>
  <NAME THRESHOLD="7" LANG="de"> Universelle Algebra    </NAME>
   <WORD   WEIGHT="3" LANG="de"> universel              </WORD>
   <WORD   WEIGHT="3" LANG="de"> Algebra                </WORD>
   <WORD   WEIGHT="1" LANG="de"> Gruppe                 </WORD>
   <WORD   WEIGHT="1" LANG="de"> Ring                   </WORD>
   <WORD   WEIGHT="1" LANG="de"> Datentyp               </WORD>
   <WORD   WEIGHT="1" LANG="de"> Omega-Algebra          </WORD>
   <WORD   WEIGHT="1" LANG="de"> Omega                  </WORD>
   <WORD   WEIGHT="1" LANG="de"> Kongruenzrelation      </WORD>
   <WORD   WEIGHT="1" LANG="de"> Faktoralgebra          </WORD>
   <WORD   WEIGHT="1" LANG="de"> Homomorphismus         </WORD>
   <WORD   WEIGHT="1" LANG="de"> Homomorphiesatz        </WORD>
   <WORD   WEIGHT="1" LANG="de"> Termalgebra            </WORD>
 </KEYPHRASE>
```

The idea is that when in a slice of text some of these WORDs can be found, and the sum of their associated WEIGHTs match or exceed the THRESHOLD specified, then this keyphrase can be associated with the slice.

The program, written in 'C', starts by reading all keyphrases and store that information into 2 tables: `keywords` and `keyphrases`. An entry in `keyphrases` contains only references to entries in `keywords`. All keywords are normalized to lower case, UTF-8 encoding.

During this process an n-ary (multibranched) tree is build were each node represents a letter of a keyword, as depicted in Fig. 2. Each node contains besides a value (the letter), a (possibly empty) list of next nodes and a (possibly empty) information pointer field. In the last node of each keyword, this pointer is filled with, a reference to additional information concerning this keyword.

Next the text for which keyphrases are to be assigned is read, were the set of characters that was used in the nodes of the Keyword Tree (and the character sequences $'-'$ and $'\backslash-'$) determines the word boundaries. These input words are normalized in the same way as the keywords.

All input words are looked up in the Keyword Tree. Each next letter (or more precise byte in a multibyte UTF-8 sequence) is readily found in a set of next nodes that is associated with a particular node using a hashing scheme. The root node of the tree contains a '\0' byte which does not participate in matching, A partial match occurs when a node is encountered that has a non-empty information pointer. When this node matches the last byte in a word, it is a full match. Then, from the information pointer, a list of keyphrases is obtained were this keyword was used, possibly with different weights.

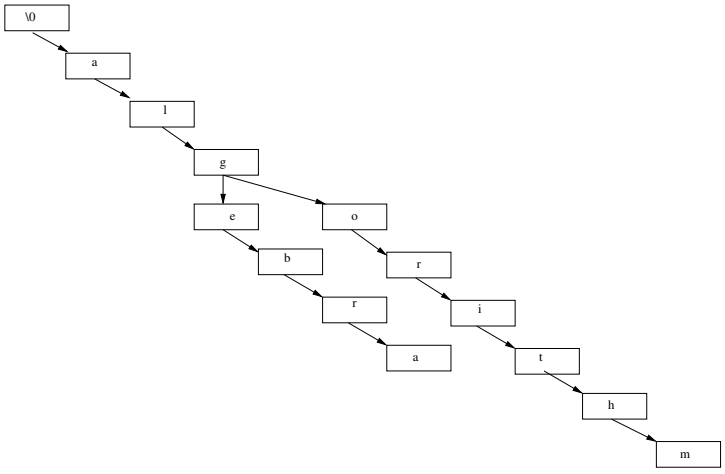


Figure 2: Keyword Tree for keywords “algebra” and algorithm”

In a “matchlist”, all potential matching keyphrases are stored together with, for each keyphrase, the keywords found in the text.

Finally, when the whole input text (a slice) has been examined, for each keyphrase in the matchlist the sum of weights is computed and compared against the threshold to determine whether or not the keyphrase can be associated with the slice.

This process is to be executed for every slice. A textbook typically contains several thousands of such slices, and a keyphrase definition file for a field such as undergraduate mathematics will contain several thousands of keyphrase definitions.

This matching software can also work with suffix lists by storing the suffixes in the tree starting at the last character backwards to strip of suffixes from a stem.

5 XMLparser

Extensible Markup Language (XML) is a W3C standard for document structure definition. For the parsing of the various XML formatted files in this project, such as the “imsmanifest.xml” file describing the sliced document and the list of keyphrases “keyphrases.xml” as well as several other control files, a parser was written in ’C’ called “XMLparser”. This parser is based on the“expat” XML parser library written by James Clark, who was one of the original inventors of XML.

XMLparser is able to parse any well-formed XML document into a parse tree, and repeatedly walk over this tree (visit all nodes). A node in this parse tree represents an XML entity. Associated with each entity (node) are its name, attribute list, content, etc. When visiting a node, an application defined function

can be called by XMLparser giving the application access to all information associated with a particular XML entity in the parsed file. These application defined function are to be stored in dynamical loadable function libraries, which XMLparser loads prior to parsing a particular file.. For example, before XMLparser parses "keyphrases.xml", it must first load the function library "keyphrases.so" containing functions able to handle entities with the names "KEYPHRASE", "WORD", etc. as in the example above.

All these dynamically loaded functions have access to a shared data space, that is not touched by XMLparser.

In the example, this is where the Keyword Tree, the keyphrase lists, etc. is stored.

Next, before parsing the file "imsmanifest.xml" describing a sliced mathematical textbook, another library is loaded with functions specialized in handling the XML entities to be found in that file. For example,

```
...
 <title>Definition  1.13</title>
   <metadata>
     <record>
       <sourcereference>6</sourcereference>
       <navtitle>Definition  1.13</navtitle>
       <types>
         <type>Definition</type>
       </types>
       <start href="slib/1/1/0/3/6/start.tex" type="latex"/>
       <main href="slib/1/1/0/3/6/math.tex" type="latex"/>
       <end/>
     </record>
   </metadata>
...
```

is a fragment of an "imsmanifest.xml" file. It contains quite different entities such as "metadata", "record" etc. than the ones that occur in "keyphrases.xml". In particular, the value of the "href" attribute in the "main" contains the filename containing the LaTeX formatted slice to be examined by the Automatic Keyphrase Recognizer program described in section 4.1.

Actually this program is embedded in these libraries, so that the text of each slice can be immediately matched against the keyphrase list stored previously.

At the end, when all slices have been processed, a new "imsmanifest.xml" is produced which is the original one, with "keyph" entities added to those "record" nodes that specify slices of text for which keyphrases were found. These "keyph" entities describe the keyphrases, and the reason why they were added: the keywords found and their weights.

An example of the `.html` output from the Automatic Keyphrase Recognizer mentioned in section 4 is shown in Fig. 3. Here a slice of German text is shown in LateX format, together with a keyphrase found, the keywords found and their

weights, and the keywords not found but defined for this keyphrase, without weights.

6 Tests and Results

The idea of identification clouds and the implementation was tested. The results are satisfactory in the sense that relevant key phrases were picked up nicely. However, what also happened is that a number of irrelevant (spurious) key phrases appeared. for examples of how this might occur see [5]. The idea of identification clouds needs further refinements, in particular the idea of negative weights, see loc. cit.

7 Future Work

Obviously it is a non feasible task to compile by hand a standard keyphrase list of some 200 000 keyphrases complete with identification clouds, weights etc. Ideas to automate this task are outlined in [4, 5]. But so far there are no implementations.

References

[1] M. Hazewinkel, "Enriched thesauri and their uses in information storage and retrieval." In: C. Thanos (ed.), *Proceedings of the first DELOS workshop*, Sophia Antipolis, March 1996, INRIA, 1997, 27-32.

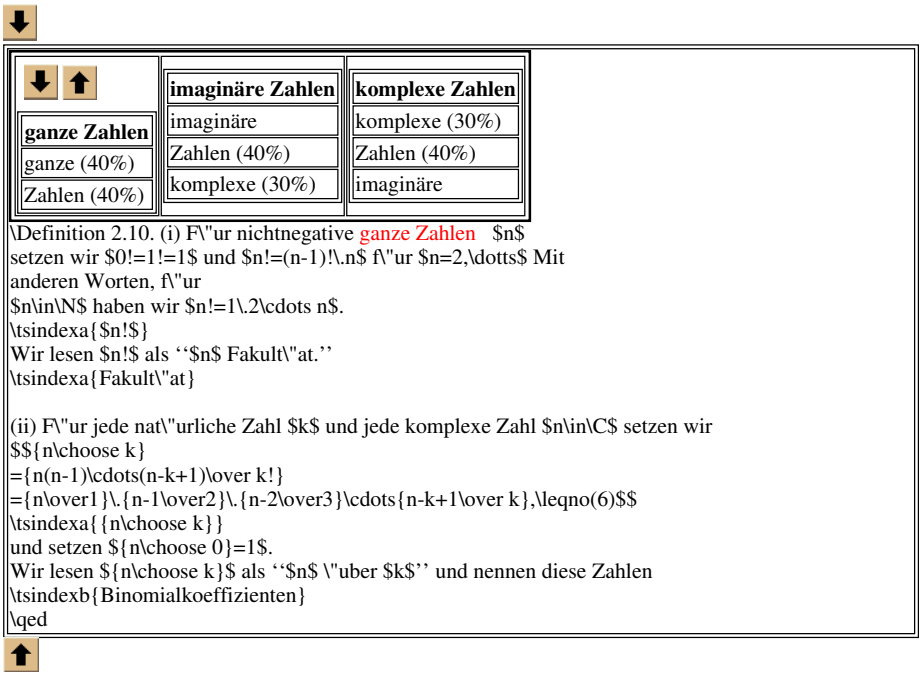
[2] M. Hazewinkel, "Topologies and metrics on information spaces." In: J. Plümer, R. Schwänzl (ed.), *Proceedings of the workshop: "Metadata: qualifying web objects"*, 1997. Available on-line: `http://www.mathematik.uni-osnabrueck.de/projects/workshop97/proc.html`

[3] M. Hazewinkel, "Topologies and metrics on information spaces," CWI Quarterly 12:2(1999), 93-110. Preliminary version available on-line: `http://www.mathematik.uni-osnabrueck.de/projects/workshop97/proc.html`

[4] M. Hazewinkel, "Dynamic stochastic models for indexes and thesauri, identification clouds, and information retrieval and storage." In: H. Gzyl (ed.), *Surveys from IWAP 2002*, KAP, 2003,

[5] M. Hazewinkel, "Identification clouds and automatic keyphrase assignment. Lessons learned from the TRIAL SOLUTION project." Deliverable for the IST project TRIAl SOLUTION, IST-1999-11397, (2003),

[6] M. Hazewinkel (ed.), "Encyclopaedia of Mathematics", Volumes 1-13. Kluwer Acad. Publ. 1988-2001.



ganze Zahlen	imaginäre Zahlen	komplexe Zahlen
ganze (40%)	imaginäre	komplexe (30%)
Zahlen (40%)	Zahlen (40%)	Zahlen (40%)
	komplexe (30%)	imaginäre

\Definition 2.10. (i) F\"ur nichtnegative ganze Zahlen n
setzen wir $0!=1!=1$ und $n!=(n-1)!\.n$ f\"ur $n=2,\dotts$ Mit
anderen Worten, f\"ur
$n\in\N$ haben wir $n!=1\.2\cdots n$.
\tsindexa{$n!$}
Wir lesen $n!$ als ‘‘n Fakult\"at.’’
\tsindexa{Fakult\"at}

(ii) F\"ur jede nat\"urliche Zahl k und jede komplexe Zahl $n\in\C$ setzen wir
$${n\choose k}
={n(n-1)\cdots(n-k+1)\over k!}
={n\over1}\.{n-1\over2}\.{n-2\over3}\cdots{n-k+1\over k},\leqno(6)$$
\tsindexa{{n\choose k}}
und setzen ${n\choose 0}=1$.
Wir lesen ${n\choose k}$ als ‘‘n \"uber k’’ und nennen diese Zahlen
\tsindexb{Binomialkoeffizienten}
\qed

Figure 3: Example of .html output from AKA