SMARTINDEXER – Amalgamating Ontologies and Lexical Resources for Document Indexing

H. Peter, H. Sack, C. Beckstein

Institut für Informatik
Friedrich-Schiller-Universität Jena
D-07743 Jena
Germany
{hpeter, sack, beckstein}@minet.uni-jena.de

Abstract

Document index compilation is a sophisticated task that requires text understanding capabilities. SmartIndexer supports the author in the process of index compilation. By providing information about the general structure of an index in combination with the lexical and semantic resources of WordNet, SmartIndexer gives suggestions for arranging potential index entries according to their semantic relationships and according to the requirements of the author. In addition, the process of index compilation can be reversed in the sense that an existing document index can be used for automated semantic annotation of the underlying document.

1. Introduction

The index is an essential part of any document, no matter if we consider a book, an issue of a magazine, a web page, or any other information source. It allows fast and efficient random access to any important topic within the document. The process of index creation is not trivial and thus requires extensive intellectual efforts: Appropriate headings must be chosen, index entries must be defined sophistically, synonymy, ambiguities and other relationships between index entries must be detected and handled properly. In the end, the creation of a sound index also affects the corresponding document because it provokes text restructuring and disambiguation of the used vocabulary.

Current indexing software (e.g. LaTeX's MakeIndex (Lamport, 1987) or MACREX(Calvert and Calvert, 1997)) supports the author only in mechanical indexing tasks, e.g. simple management or sorting of index entries. This type of software also does not assist the author in the much more complex and creative task of originating accurate and sound index entries. An entirely automated indexing process requires text understanding capabilities that are beyond the ability of prevailing computer systems.

Our goal was to develop an architecture – the SMARTINDEXER – that supports the author in the creative tasks of the indexing process. For this purpose, we designed an ontology (in the following referred to as *Index Ontology*), which contains general knowledge about index elements and their relationships. Index quality strongly depends on the amount of its inherent semantics. An index can be regarded as a network, where the index entries represent the nodes. Subentry relationship between two index entries as well as different cross-references among index entries constitute the arcs. This network embodies the semantic interrelationships inherent in the index. SMARTINDEXER facilitates the creation, expansion, and management of this network and thus, enables the generation of a high quality index.

Providing semantic relationships between words, as e.g. hyperonymy or meronymy, is the main task of the electronic lexical database WordNet (Fellbaum, 1998). SMARTINDEXER employs WordNet in connection with its Index Ontology to assist the author at the intellectually sophisticated indexing task. Supplementary, domain ontologies – if available – provide useful information about a document's subject. SMARTINDEXER can use these ontologies as significant input beyond the knowledge offered by WordNet. The paper is structured as follows: Section 2 and Section 3 introduce the reader to the basic principles of indexing. Section 4 covers the architecture of the SMARTINDEXER, while Section 5 gives a short overview of the SMARTINDEXER algorithm. In Section 6 a possible transformation of a document index into a domain ontology is shown. Section 7 concludes the paper with an outlook on ongoing and future work.

2. Index and Index Elements

According to the British Indexing Standard (Mulvany, 1994) an index is a systematic arrangement of entries designed to enable users to efficiently locate information in a document or specific documents in a collection.

Index entry: An index entry consists of a heading (or main heading) and at least one of the following components: a subentry, a reference locator (in the following referred to as locator), or a cross-reference. A heading is a term – normally a noun or a noun phrase – which reflects a concept in the document.

Subentry: A subentry is similarly structured as an index entry. It is composed of a subheading, one or more locators, and – only rarely – cross-references. The corresponding concepts of the subheadings are always related to the concept of the superordinated main heading. In the majority of cases subheadings represent subdivisions or more specific aspects of the main heading.

Sub-subentry: A subentry can have further index entries – so called sub-subentries. The above mentioned statements about subentries hold analogously for sub-subentries. In general it is not recommended to go beyond the level of sub-subentries.

Locator: Locators follow a heading and indicate that part of a document, where information related to the heading can be found. In printed media, reference locators are usually page numbers, section numbers, or line numbers.
Cross-reference: Cross-references establish a relationship between one heading and another. This makes it possible to connect scattered information within the index. A book index usually provides two kinds of cross-references: *see* references and *see also* references. The first kind is used for variant spellings, synonyms, aliases, abbreviations, and so on. *See also* references are used to guide the user to another closely related heading that supplies additional information.
A high quality index is an essential prerequisite for efficient information retrieval. Direct access to specific information within a document becomes hardly viable without an index.

3. Index Compilation

Compiling an index is an intellectually sophisticated process. The difficulty of that process lies in capturing the essence of a document by means of only a few short, expressive and predictable headings or heading phrases. Furthermore, synonyms, ambiguities, and various relationships between terms have to be detected and handled properly.
The index compilation process usually consists of the following six steps:

1. Terms are highlighted that are considered to be main headings or subheadings in the index. Each highlighted term is augmented with additional and more specific information. This information will be used in a subsequent step for the generation of subheadings.

2. A corresponding locator is assigned to each highlighted term.

3. Then, highlighted terms and locators are arranged in order within the existing index. There are several possible index orderings. The most commonly used index order is the alphabetical order.

The remaining three steps generate a consistent document index from the collected temporary index entries:

4. It has to be decided, which term is transformed from a set of synonyms or closely related terms into a main heading. Furthermore, appropriate cross-references have to be created that reflect the existing semantic relationships.

5. Then, an index level has to be chosen, where the index entries have to be placed.

6. Finally, it has to be verified that all cross-references relate to an existing index entry that offers a locator.

The mere mechanical aspects of index creation (step two and three) can be accomplished easily with current indexing software. However, the author usually does not obtain any support in the intellectual aspects of index creation. Thus, the goal of our SMARTINDEXER architecture is to assist the author in the creative tasks of index compilation – especially in steps four to six.

4. The SMARTINDEXER Architecture

The SMARTINDEXER architecture is based on a two component framework: the *Index Generator* and the *Ontology Processor* (for an outline of the SMARTINDEXER work flow see figure 1).
The Index Generator receives as input a potential index entry from an arbitrary word processing application (1). Additionally, an already existing document index is passed to the Index Generator (2). After a preprocessing step containing (among other things) spell checking and word stemming, the author has to mark up the sense carrying substring (SCS) of the potential index entry. Then, the SCS is passed over to the Ontology Processor (3). The Ontology Processor recalls the entire lexical field (LF[SCS]) of the SCS by means of WordNet (4,5). LF[SCS] contains synonyms, hyponyms, hypernyms, holonyms, meronyms, and sister terms of the SCS. After this lookup, the SCS is passed back to the Index Generator (6). The Index Generator uses the general knowledge about indexing represented by the Index Ontology for making suggestions about new potential index elements, as e.g. cross-references or subentries (see section 5. for a more detailed specification of the indexing algorithm).
In addition to lexical resources as e.g. WordNet, SMARTINDEXER can use different knowledge repositories. The author has the possibility to supply domain ontologies referring to the subjects discussed in the document to be indexed. If there is no suitable ontology available, standard WWW search engines as e.g. Google or specialized semantic search engines as e.g. Swoogle (Ding et al., 2004) can be used for searching better suited ontologies (see figure 2). Typically, domain ontologies describe domain entities and various relationships between them. In particular 'IS-A' or 'PART-OF' relationships are good candidates for possible index elements, especially for cross-references.
With the help of the Index Ontology the Ontology Processor filters the found relationships and transfers them to the Index Generator. Depending on this information the Index Generator suggests suitable index elements and lets the author decide which of them to include in the document index. Finally, the Index Generator returns the chosen index elements to the word processing application (7) that inserts the new index entry into the document index (8). In certain situations inserting a new index entry requires complex index rearrangement.
SMARTINDEXER is being implemented as a Java application independent of specific hardware or operating systems. For the management of semantic information provided by RDF, RDFS, and OWL ontologies, we use the JENA application programming interface (McBride, 2002). The preprocessing of possible index entries requires word stemming, which is performed with the Java implementation of the Porter stemming algorithm (Porter, 1980). In order to access lexical information provided by WordNet we use the Java Word Net Library (JWNL) (Didion, 2004).

5. SMARTINDEXER Algorithm

The SMARTINDEXER has to be able to detect relationships between index entries to properly assist the author with index compilation. This requires that the concept of a heading

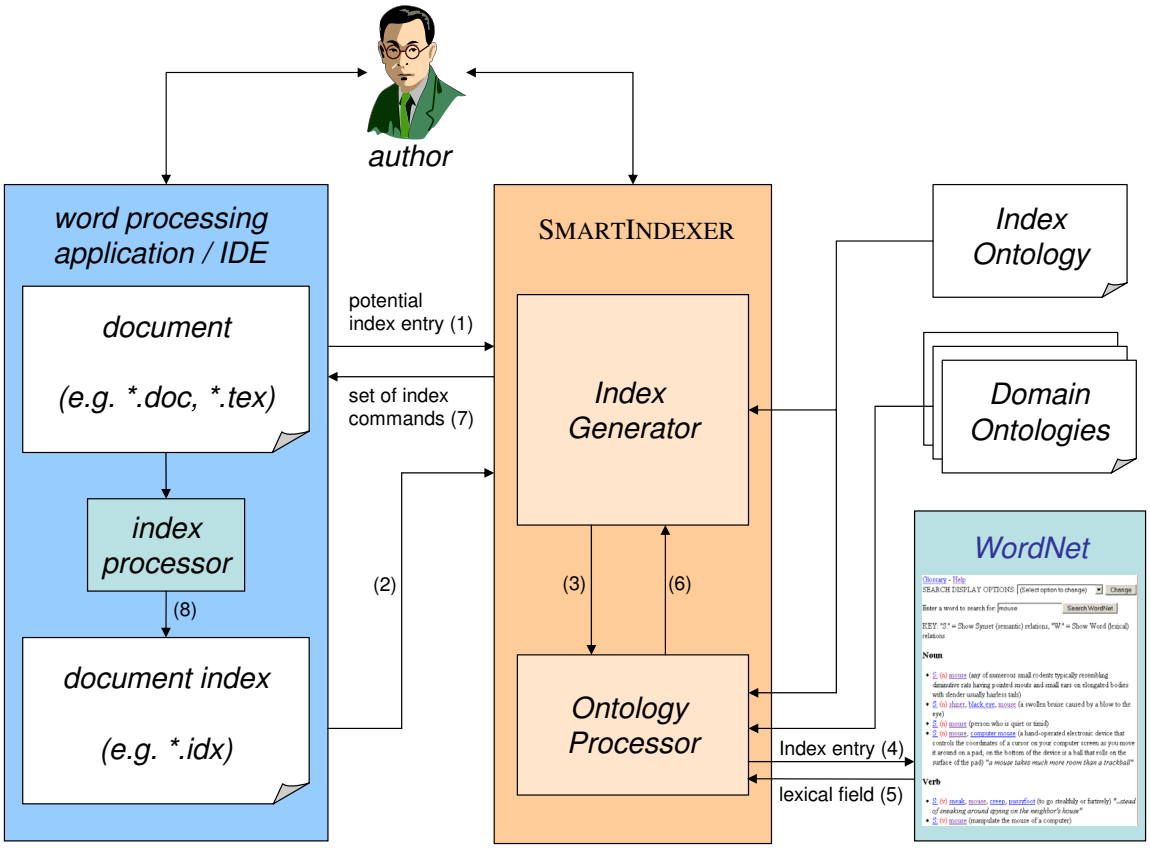

Figure 1: Indexing Process with SMARTINDEXER

is known. By knowing the concept of a heading SMARTINDEXER is able to identify relationships between index entries by the combined use of the Index Ontology and lexical resources as WordNet. The Index Ontology provides general knowledge about the components of an index and their relationships with each other.

As already mentioned, the Index Generator uses the general knowledge about index creation and the information offered by WordNet to make suggestions for a potential index entry i. First, the underlying concept of i is determined in a preprocessing step in cooperation with the author. The preprocessing comprises the following operations:

1. Perform spell checking of i and stop word removal from i.
2. Ask the author to mark up the sense carrying substring (SCS) of i.
3. Perform word stemming of i according to the porter stemming algorithm (this step is already provided by WordNet).
4. Use WordNet or available domain ontologies to determine the underlying concept of i, which will be used in the main index processing algorithm. This step has to be directed by the author.

Preprocessing must be guided by the author because the SMARTINDEXER algorithm is not able to determine the underlying concept of i. In order to realize this step in an autonomous way text understanding capabilities are indispensable.

WordNet contains so called synsets representing concepts that are identified with the help of so called sense keys. The sense key resulting from preprocessing of i is a prerequisite for the identification of the semantic relationships between i and the existing document index.

This main indexing process can be divided into two main steps: First, a possible position p of i within the already existing document index I has to be determined. Then, the new index entry i has to be inserted at position p either with its locator or as a cross reference. To accomplish both steps the semantic relationships between i and the existing index entries $j \in I$ have to be located. This can be achieved in the following way:

1. Determine the position p of the new index entry i within the existing document index I. This is done depending on the type of information available about i:
   - If i is a synonym of an already existing index entry $j \in I$, then the position p of the new index entry i can be the same as the position of j.
   - If there are already known subordinated relationships (e.g. hyponyms, meronyms) of the new index entry i and the already existing index entries $j \in I$, then i can be positioned at the position of j, while j has to be relocated below the new index entry i.

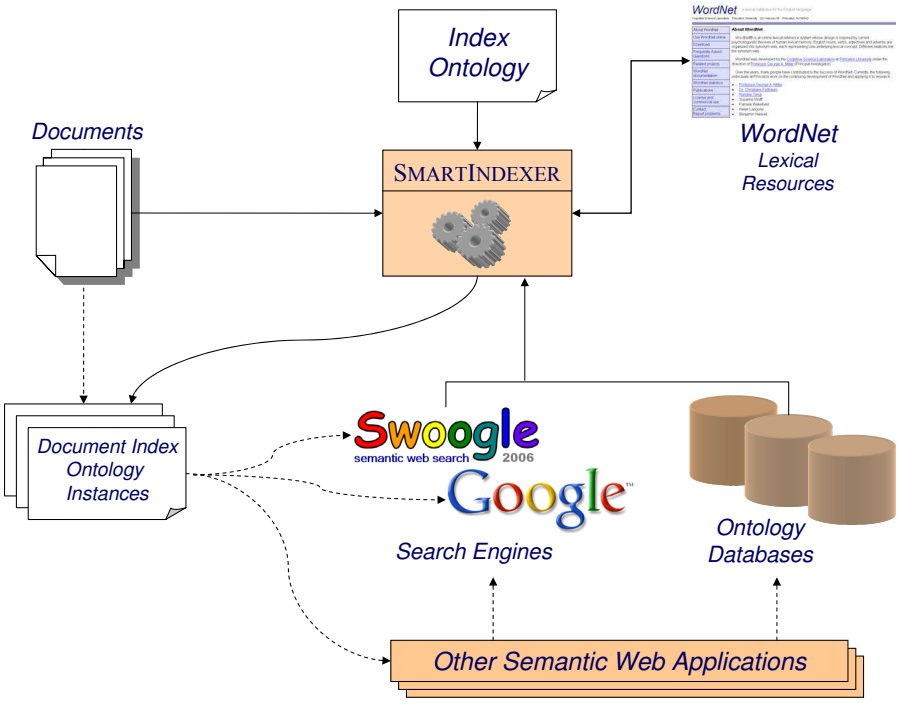

Figure 2: SMARTINDEXER Embedded in Semantic Web Framework

- If there are already known superordinated relationships (e.g. hypernyms, holonyms) of the new index entry i and the already existing index entries $j \in I$, then i can be positioned below the already existing superordinated index entry j.
- Otherwise, if there are already known associated terms (e.g. sister terms) of the new index entry i and the already existing index entries $j \in I$, they can be used to find a suitable position for the new index entry i in I. If i is a sister term of j, i can be positioned at the same index level as j.

2. Insert the new index entry i at position p with its locator or as a cross-reference:

   - *see* references can be already existing index entries $j \in I$, which have a synonymic relationship with i.
   - *see also* references can be already existing index entries $j \in I$, which have any semantic relationship with i.

SMARTINDEXER only gives suggestions, where to insert a new index entry into the existing document index. The final decision, where to supply the new index entry is up to the author.

The index compilation process is illustrated with the following example (see figure 3). The new index entry *mouse* has to be inserted into an existing index. After the preprocessing step SMARTINDEXER determines *mouse* to be a direct hyponym of the existing main heading *rodent*. Additionally, *mouse* also is determined to be a direct hypernym of the existing main heading *field mouse*. SMARTINDEXER suggests two possible arrangements to the author, who determines which of the proposed variants should be used. Choosing the second variant requires the rearrangement of already existing index entries. The new index entry *mouse* becomes a main heading, while *field mouse* and its subordinated index entries become subentries of *mouse*.

6. Embedding SMARTINDEXER within the Semantic Web Framework

A document index provides direct access to specific information within the document. It can be considered as a very condensed summary of the underlying document and thus, also providing access to essential concepts within the document.

By reversing the index compilation process, SMARTINDEXER can also be utilized to transform an already existing document index into an ontology that captures important semantic knowledge about the document. For this purpose, the already mentioned Index Ontology has to be considered to be a generic class framework for the index at large. Accordingly, a document index has to be considered to be a specific instance of the general Index Ontology.

By making use of this consideration, we have the possibility to transform any document index file into an RDF file reflecting all the relationships defined by the underlying document index instance. The resulting RDF file can be used to provide a traditional index representation, i.e. an

Index (before insertion)

fieldmouse, 13, 15
- prairie vole, 16
- meadow vole, 16
- habitat, 15
- *see also* rodent

rodent, 1
- beaver, 10, 11
- dentision
  - incisor, 4
  - rotation of teeth, 5
- hamster, 6, 8 – 10
- *see also* field mouse

Index (after insertion)

mouse, 12
- **fieldmouse, 13, 15**
  - **prairie vole, 16**
  - **meadow vole, 16**
  - **habitat, 15**
  - ***see also* rodent**

rodent, 1
- beaver, 10, 11
- dentision
  - incisor, 4
  - rotation of teeth, 5
- hamster, 6, 8 – 10
- *see also* field mouse

Figure 3: Example of Index Entry Insertion of the new Index Entry *mouse*

alphabetically ordered list of index entries.
In addition, one can use the RDF data structure to display the index in different alternative ways that provide supplementary information. It is possible to display the document index as a topic map or as a graph, clarifying the relationships between the index entries by graphical visualizations that can be used for inner document navigation. Furthermore, the RDF index instance of the document index identifies the significant keywords of a document, thus providing information about what is important and what is not. According to this kind of interpretation, index entries with a large number of references in the document can be considered to be of higher significance than index entries with only a single reference.
Index entries also reflect how index keywords do interact, e.g. by giving information about hyperonymy, meronymy, homonymy, synonymy, and other kind of associations. This additional semantic information can be used to draw new links between different sections of the document. It enables the reader to break out of the linear text flow of the document by using cross connected index keywords (*see* and *see also* references) like the hypertext links.
The semantic relationships provided by the index can be utilized as a starting point for further semantic annotation of the document related to the index. Also corresponding domain ontologies, which match the concepts provided by the document can be identified more easily.

7. Conclusions and Outlook

Document index compilation is a sophisticated task. It requires smart knowledge processing. SMARTINDEXER supports the author during the process of index compilation. The compilation of a sound document index requires the identification of circles or blind references. This is accomplished by using a semantic index description (Index Ontology) in combination with the lexical resources provided by WordNet. The document and the index ontology together with WordNet's semantic relationships fosters the emergence of a new ontology from the document's index. This ontology can be used for visualization and navigation issues. Furthermore, it can as well supply additional semantic information for the underlying document. Therefore, SmartIndexer can be considered as being a first step towards semantic document annotation, which is mandatory for enabling the semantic web.

8. References

Grigoris Antoniou and Frank van Harmelen. 2004. *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts.

Hilary Calvert and Drusilla Calvert. 1997. MACREX Manual for Version 6.5.

John Didion. 2004. The Java WordNet Library.

Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. 2004. Swoogle: A Search and Metadata Engine for the Semantic Web. In Grossman et al. (Grossman et al., 2004), pages 652–659.

C. Fellbaum. 1998. *WordNet – An Electronic Lexical Database*. MIT Press.

David Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors. 2004. *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*. ACM.

Leslie Lamport. 1987. MakeIndex: An Index Processor for LaTeX.

Brian McBride. 2002. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59.

Nancy C. Mulvany. 1994. *Indexing Books*. The University of Chicago Press, Chicago.

Martin F. Porter. 1980. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137.

Laurent Prévot, Stefano Borgo, and Alessandro Oltramari. 2005. Interfacing Ontologies and Lexical Resources. In *Proceedings of OntoLex 2005 - Ontologies and Lexical Resources*, Jeju Island, Republic of Korea, 15 October.

Frank van Harmelen, Jeen Broekstra, Christiaan Fluit, Herko ter Horst, Arjohn Kampman, Jos van der Meer,

and Marta Sabou. 2001. Ontology-Based Information Visualisation. In *International Conference on Information Visualisation, IV 2001*, pages 555–562, London, England, 25-27 July.