6 Pre-processing

Having defined the system architecture and construction of Search Profiles, each single component in system has to be implemented. Pre-processing is the first step of IE in CapturePlus. The main role of pre-processing is formatting or converting the input documents, so that later extracting can be performed easily. Typical pre-processing tasks are converting documents in one uniform format, indexing, segmenting and structuring documents.

Document converting can be operated in different ways. Documents saved with different formats (such as ASCII Text, Word, PDF, Postscript etc.) must be converted in one uniform format (for example RTF), so that later processing is based on one standard and becomes possible. Besides converting, reformatting is in some cases necessary too. For instance, HTML documents can be converted to well-formed XML documents, so that later extracting process is able to analyse the hierarchical relationship of documents using standard XML-parser. Note, that converting can cause loss of information. For example, if all documents are converted to ASCII format, meta-information as well as style information and pictures, table etc. are dropped away. Hence, converting process has to guarantee that the main information, or information which does not belong to main information but is helpful for extracting processes (e.g. tag information in HTML), does not get lost.

In document indexing, documents are analysed at the word level. The results are stored then in an internal document structure. This internal document structure can be used for later processing and provides rich information about properties of occurring of words in documents.

Segmentation undertakes the task to identify region of text in documents. Typical segmentation is to delimit documents in chapter, section, paragraph, sentence etc. Segmentation is useful for later processing because the segments in different levels can help later processing to detect and exploit text zones more easily.

Structuring is used for online HTML documents. Although HTML documents have implicit structure defined by HTML tags, they do not have a clear structure, because HTML documents are not SGML well-formed. That is, syntaxes of HTML are developed for practical issues and therefore can not be parsed by a structured parser (such as SGML or XML parser). Hence, HTML has to be cleaned up and converted to XML-style documents, so that the implicit structure in HTML can be parsed by a structured parser (such as XML-parser).
In this section we will introduce some pre-processing techniques for two types of documents, namely normal documents and online HTML documents. For normal documents, pre-processing consists of two functions: indexing and segmentation, while for HTML documents pre-processing means mainly structuring.

6.1 Pre-processing with normal documents

6.1.1 Indexing of documents

The goal of document indexing is to allow rapid access to individual words/strings within a document. Document indexing is a standard technique in Information Retrieval and can be used in Information Extraction for better processing performance. In the indexing phase, a document is delimited at first at words level. In information retrieval, this step discards all space characters and stop words (such as “is”, “not” etc.). Unlike IR, indexing here keeps all words in documents, because each word can be potential useful in the later extracting phase. Afterward, information about the occurrence of words is analysed. The frequency of occurrence of each word is calculated. Position information for all occurrences of one word is also determined. Such information can provide a rapid access to words in the later extracting phase. Finally, after analysing, results are stored in an internal document structure. The structure consists various string access lists to allow rapid access to wanted string. For example, one string list contains descriptions about the occurrence of words and their positions in documents, and another is implemented for rapid finding similar words. Because indexing is a computational intensive task, the total internal document structure can be stored in an intermediate file, so that indexing can be processed only once. Internal document structure can be reconstructed quickly by loading such intermediate file, if the document is already indexed (Brown 1997).

6.1.2 Segmentation of documents

Another possible pre-processing task for normal documents is Segmentation. In the pre-processing phase, documents can be segmented in different levels. Segments can be table of contents, which define the construction of documents. They can be also boundaries, which delimit texts in different topics. Segmentation information is very helpful for later text zones detecting. As well as word-level indexing methods, various methods are developed for automatic segmentation of documents (Hearst 1994, Beeferman 1997, Richmond 1997, Katz 1996). Normally, automatic segmentation of documents relies on calculating similarity of texts in one document (Hearst 1994).
However, such automatic segmentation can not be used to extract fine pieces of information because the detected regions are not exact enough. Hence, automatic segmentation is used normally in document classification, or for information retrieval. For information extraction, more exact segments are required. Some IE systems need manual processing to format the input documents in suitable form (Soderland 1999). Manual processing is time consuming and error prone. Here we introduce an half-automatic tool to segment documents in clause, paragraph and sentences level.

Segmentor is a tool with interactive functionality for half-automatic segmentation of documents. Segmentor assigns one (or more) hierarchical structures to the input document. The assignment of a hierarchical structure requires four main tasks to be carried out:

- Determine all candidate boundaries between structure parts in the text (100% recall)
- Clean up the list of boundaries (100% precision)
- Assign a level (with respect to the structure tree) to each boundary
- Extract/Generate a name for each boundary

The workflow of segmentor is shown in Figure 21. In following we introduce each step briefly.

![Workflow of Segmentor](image)

**Figure 21: Workflow of Segmentor**

6.1.2.1 Find All Boundaries

The Find All Boundaries task is defined to be a search through the text that delivers all candidate boundaries at all levels in the intended hierarchical
structure. In this step segmentor must reach 100% Recall. But the precision can be lower than 100%.

In general, a candidate boundary for a particular level will be defined by a set of possible delimiter strings - e.g. sentences end with "", "?", "!", and so on. In order to cope with the normal section-sub-section structure of documents, it will also be necessary to detect numbers within a document. In order to deal with most standard document structure formats, a set of standard components for boundary finding is employed.

Where the required boundaries are represented by more complex string patterns (e.g. application specific labels to denote requirement clauses), then, in general, a pattern recogniser must be defined to find these boundaries.

The following table summarises the standard boundary finders that should be included in the Segmentor tool:

<table>
<thead>
<tr>
<th>Standard Component</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence Detector</td>
<td>Finds possible ends of sentences based on a delimiter list</td>
<td>Delimiters: {',', '?', '!', ...)</td>
</tr>
<tr>
<td>Paragraph Detector</td>
<td>Finds possible ends of paragraphs based on a delimiter list</td>
<td>Delimiters: {'newline', ...)</td>
</tr>
<tr>
<td>Numbered Section Detector</td>
<td>Looks for nested numbers in the text</td>
<td>Finds the pattern &lt;int&gt;,&lt;int&gt;, etc.</td>
</tr>
</tbody>
</table>

Table 3 Pattern for detecting standard boundaries

Note that for “numbered section detection”: the standard format for section description is variegated. The most common form is “alpha numeric” (such as “2.3”, section “4.6” etc.). However, other formats are not unusual either. For example, the Latin description (such as “Section IX.II”) or the mixed form (such as “part 3.5.a”) are also used in industrial documents. Because in this step, the system tries to achieve 100% precision, all possible representation forms for section description have to be investigated. We use regular expressions to defined the patterns for recognition numbered section. The following is a sample regular expression for detecting numbered sections and paragraphs.

"(((\[0-9]+)([\.|\)])*)((\[[\-\w]+\])([\.|\)])*)((\[\d\d\]|(\[0-9 a-z A-Z]+)\[\d\d\]))".

6.1.2.2 Clean-Up Boundaries

The Clean-Up Boundaries task takes the list of candidate boundaries generated by step “Find All Boundaries” and eliminates any and all invalid entries. In this step 100% Recall and 100% Precision must be reached.
In principle, this task could be eliminated by incorporating the required constraints into the various boundary recognisers used in “Find All Boundaries”. However, it is a general truism in classification tasks that a “filter and test” approach is easier to implement than a “strict one-pass classification” approach. Moreover, by having two distinct phases to the generation of the list of boundaries, weak and strong constraints on the boundary instances can most elegantly be defined.

Actually, several related tasks fall under the category of “Clean-Up Boundaries”:

- **Exactly Position Each Boundary**: The list of boundary positions returned by “Find All Boundaries” may be only approximate. For example, sentence recognisers may return the position of the last ‘.’ of a sentence. This position may need to be moved (white space skipping) so that the actual boundary is the first character of the next sentence.

- **Merge Equivalent Results**: In many cases, a single (logical) boundary may be recognised multiply by the application of simple delimiter lists in Find All Boundaries. For example, given a section heading such as “1.2.4.1 Sectionname” it is likely that each of the integers in the first string are recognised as a possible section boundary. Similarly, if newlines are used to delimit paragraphs then problems arise when more than one line is occasionally used (e.g. for formatting reasons). Various types of constraint can be employed to overcome these problems, e.g. Only boundaries at string start, minimal separation constraints for parts, white space skipping, etc.

- **Eliminating False Boundaries**: Apart from the above type of elimination, there may be a genuine need to eliminate some standard “false hit” type of boundaries. This is particularly pertinent for sentence detection where standard abbreviations (e.g. “etc.”, “e.g.”, “z.B.” and so on) lead to false hits. That the first word of a sentence should start with a capital letter may also be used to strengthen sentence boundaries. It may also be necessary to recognise and eliminate other numbered text items (e.g. list elements, diagrams etc) in order to achieve accurate section boundaries.

The following table summarises the set of standard methods for repositioning the boundary position, and/or for merging equivalent boundaries, and/or for eliminating false boundaries:
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move to Start of Next Word</td>
<td>Skips white spaces till the first string after the boundary</td>
<td>White space delimiters = {'',TAB,newline,...}</td>
</tr>
<tr>
<td>Move to Start of Previous Word</td>
<td>Skips white spaces till the first string before the boundary</td>
<td>White space delimiters = {'',TAB,newline,...}</td>
</tr>
<tr>
<td>Move to Start of Current Word</td>
<td>if the current boundary is in the middle of a string, move it to the start of that string</td>
<td></td>
</tr>
<tr>
<td>Move to Start of Current Line</td>
<td>for boundaries that occur somewhere in the middle of a line, move it back to the start of a line</td>
<td>Line Start Delimiters = {newline,...}</td>
</tr>
<tr>
<td>Set Minimal Part Size</td>
<td>If the distance between two boundaries is less than the user defined minimum, then one of them is chosen</td>
<td><em><strong>Not clear where the new boundary should be formed</strong></em></td>
</tr>
<tr>
<td>Ignore Standard Abbreviations</td>
<td>If the string that occurs at the point of the boundary belongs to the set of standard abbreviations, then delete that boundary</td>
<td>Standard Abbreviations include English and German = {etc., e.g., z.B., ...}</td>
</tr>
<tr>
<td>Check for Standard Section Labels</td>
<td>For integer-based boundaries, checks that the boundary conforms to a nested-number format</td>
<td>Acceptable formats = (&lt;\text{int}&gt;,(.&lt;\text{int}&gt;)&lt;\text{roman}&gt;(,.&lt;\text{roman}&gt;))</td>
</tr>
<tr>
<td>This Word starts with a Capital</td>
<td>takes the string which occurs at the boundary and sees if its first letter is a Capital letter</td>
<td>Capital Letters = {'A',...'Z'}</td>
</tr>
<tr>
<td>Next Word starts with a Capital</td>
<td>Skips white spaces till the first string after the boundary and sees if its first letter is a Capital letter</td>
<td>Capital Letters = {'A',...'Z'}</td>
</tr>
</tbody>
</table>

**Table 4 Standard methods for clean up boundaries**

6.1.2.3 Assign Boundary Levels

The “Assign Boundary Levels” task takes the list of boundaries generated from “Clean-Up Boundaries” and assigns a single, integer level to each boundary. Top-Levels are at level 1, all other levels are greater than 1. There should always be some defined top-levels. Where possible, level assignment should be consecutive.

After the first two tasks, a linear (and hopefully accurate) set of all required structure boundaries is produced. This should be sorted into a tree structure. This is achieved by assigning a level number from 1...n to each of the boundaries. The depth of the boundary represents the distance from the root of the tree (the whole of the document is assumed to represent level 0). Figure 22 shows relationship between document boundaries and hierarchical Levels.
The “Assign Boundary Levels” task processes stepwise: at first, system get the current boundary and tries to generate all possible following numbers for such boundary, either for upper level or lower level. For example, if the current boundary has the number “2.1.2” with level 3, all possible following numbers can be calculated as “2.1.3”, “2.1.2.1”, “2.1.2.2”, “2.2”, “level 2” and “3, level 1”. If the next boundary is one of the wanted following number, this boundary is set to the corresponding level. Otherwise, Segmentor prompt dialog to enable user to set the correct level, or to delete the current boundary. This operation is interactively processed until all extracted boundaries are assigned corresponding levels.

Besides the interactive operation, system can also change intern boundaries with some heuristic approaches automatically. For instance, the “look forward” action searches all similar boundaries and deletes them, which are direct after the currently deleted one. For example, If user does not want to assign the number “part 3.5.a” as section, it could be assumed that direct following boundaries such as “part 3.5.b”, “part 3.5.c” etc. are also to be deleted. This operation is repeated until all similar boundaries are eliminated or a new wanted boundary is reached.

Another approach is so called “look back” strategy. For instance, following the boundary “5.3.2”, which is assigned as level 3, is a boundary with the value “part A”. System is not able to determine the level of the current boundary and
then ask the user. User observes that this boundary is part of the section 5.3.2 and then assign it as level 4. Now the next coming boundary is “part B”. System looks backward and find that there is a similar boundary “part A” direct before the current boundary. The boundary “part B” is then assigned the same level as the previous one.

After assign levels to all boundaries, Segmentor represents boundaries as hierarchical document structures.

6.1.2.4 Assign Boundary Names

The Assign Boundary Names task takes the list of boundaries generated from “Clean-Up Boundaries”, combined with the level values assigned from “Assign Boundary Levels”, and constructs and assigns an appropriate string-based identity for each boundary. Boundary names can be assigned from document text or automatically generated with numbered prefix. This step can be processed full automatically. The final result of segmentation is output to an intermediate file, so that such hierarchical structure can be loaded in the later processing.

6.1.3 Related Work

Pre-processing is the first step for Information Extraction. The main role of pre-processing is converting, formatting and preparing for underlying IE processes. Currently, IE Systems use various pre-processing mechanisms for defined preparations. RoboTag’s pre-processing (Bennett, Aone, & Lovell, 1997) includes segmenting the text, performing morphological analysis and lexical lookup. Pinocchio (Ciravegna 1999) receives as input the result of a pre-processor that performs text zoning, morphological analysis, part of speech tagging and Named Entity recognition. Pre-processing in LaSIE performs rule-based part-of-speech training (Brill Tagger); English morphological analyser with exception table based on WordNet.

The pre-processor in FACILE (Ciravegna et al. 1999) transforms a stream of characters into analysed tokens. It provides morpho-syntactic information and a disambiguated syntactic tag for each token in the input, treating proper names as single tokens and assigning them a (disambiguated) semantic tag. The key steps are the same for all the languages considered (currently English, Italian, German and Spanish) and use the same machinery with language-specific resources. The steps are:

1. Tokenisation (splitting of input text, assignment of orthographic features)
2. Morphological analysis (stemming and assignment of morpho-syntactic information)
3. Part of speech tagging (disambiguation of multiple morpho-syntactic categories)
4. Database lookup (assignment of semantic features)
5. Named entity analysis (NEA). The pre-processor input is a zoned text, i.e. with parts to be analysed identified and marked up. Its output is a text where each (group of) word(s) is associated with orthographic and morphological features, normalised form and semantic labels (e.g. “John Smith” is a “proper name” with orthographic class “capitalised”, feature “masculine” and semantic tag “person”).

Indexing is also necessary in Text Mining. For Text Mining task, pre-processing extracts useful words and terms with NLP techniques for future association analyse (Feldman and Hirsh 1997).

Segmentation is also widely used in the phase pre-processing. Methods for Segmentation are various. WHISK (Soderland, 1999) uses manually created segmentation for multi-slots documents. Each instance of template is separated by special token strings manually.

Hearst (1994) introduced TextTiling approach for partitioning expository texts into coherent multi-paragraph discourse units. A cosine measure is used to gauge the similarity between constant-size blocks of morphologically analysed tokens. First-order of change of this measure are then calculated to decide the placement of boundaries between blocks, which are then adjusted to coincide with paragraph segmentation, provided as input to the algorithm. This approach leverages the observation that text segments are dense with repeated words. Relaying on this fact, however, may limit precision because the repetition of concepts within a document is more subtle than can be recognised by only a “bag of word” tokenizer and morphological filter. Beferman (1997) combines Hearst’s algorithm and decision tree technique and presents a statistical model for segmenting unpartitioned text into coherent fragments.

Richmond (1997) improves the Hearst’s model by introducing a new measure of word significance. Richmond’s algorithm improves accuracy without sacrificing language independence. However, such statistical methods are not able to provide exact segmentation of documents. Moreover, Hearst’s algorithm segments documents in a flat structure. That is, segmenting hierarchical structures is not possible.

Compared with full manual and full automatic segmentation, Segmentor provides on the one hand an exact segmenting of hierarchical structure, on the other hand, a quick and easy processing. Therefore, Segmentor is more useful for practical applications.
6.2 Pre-processing with Online (HTML) documents

6.2.1 Converting HTML to XML

Pre-processing with HTML document is different to that with normal one. Methods for indexing and segmentation used for normal documents are not more suitable for HTML, because the original HTML source documents contain rich tag descriptions. One way to use the conventional pre-processing in HTML is to convert HTML at first to normal text documents. However, if HTML documents are converted at first to normal documents and then extracted as the normal one (this operation can be achieved for example by using “save as text file” command in Web browser), all useful tag information will get lost, which can help IE systems enhancing the performances. Another possible indexing technique is decomposing HTML in different tokens including tags. With this approach, not only underlying texts but also tags are considered as tokens.

Methods for segmentation of HTML are also possible, if HTML document contains multi-slot instance. In this case, additional tags can be inserted into the source document, in order to separate the instances. In normal case, however, the segmentation is not used.

As discussed in Section 4.2, there is implicit hierarchical structure in HTML documents. If such hierarchical relationship can be analysed, such hierarchical relationships can be very useful to define text zones. As (Kushmerick 2000) mentioned, emerging standards such as XML will simplify the extraction of structured information from heterogeneous sources. (Knoblock and Minton 1998) observed also, that wrapper induction algorithms may be able to use XML as a source of supervised training data. Our consideration is, that we can analyse the (implicit) structured information in HTML sites with help of XML-Parser and find out the structure properties of the contents. Therefore, in the pre-processing phase, the HTML documents are converted to XML-style.

The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web (W3C 2001). As discussed above, although the tags of a HTML document imply its structure, we can not use this information reliably, because we can not follow the structure exactly. By contrast, XML documents have a complete and clean hierarchical structure specified by start tags and end tags. Therefore, to analyse the structure we need to convert HTML sites to well formed XML-style documents, which is the task of Pre-processing. HTML documents are not XML conformed because some tags have only start tag but do not have end tag (such as <br>, <hr> etc.). So that normally a HTML site can not be parsed by an XML-parser directly. The converting from HTML to XML is simply: eliminate all single tags and enclose
all attribute values in quotes ("). This converting can be made automatically with help of tools. We use the standard open source TIDY provided by W3C organisation (http://www.w3.org/People/Raggett/tidy) to convert HTML to XML automatically. Tidy is able to fix up a wide range of formatting problems and syntax errors. The main functions of TIDY is to clean up HTML pages and recover various syntactical errors in HTML.

One limitation of TIDY is that TIDY does not convert special characters (prefixed by "&") to a normal one. These special characters can cause parser error because the parser does not accept "&" in text. To work around this problem, we replace all "&" by the string “\amp”. Once the documents are well formed with XML-style, they can be parsed by an XML-Parser.

Note that these documents are not real XML-documents because they do not have Data Type Description (DTD) or XML-schema. However, such converted documents are well-formed and therefore can be parsed by an XML-parser. Hence, it is enough to use them to get the structure information. Figure 23 gives an simple example for converting HTML to XML.

6.2.2 Related Work

In general, pro-processing for HTML documents has two major tasks: indexing and recovering. For the first task, system discomposes HTML in different tokens for later processing. STALKER (Musela 1998), for example, uses token-based Wrapper and therefore needs to tokenise HTML texts into a sequence of tokens (e.g., words, numbers, HTML tags, etc.).
The later task tries to format HTML to error free or well-formed documents. XWRAPPER (Liu L, et al. 2000) cleans up HTML documents at first in the pre-processing phase. The bad tags are deleted and the uncompleted tags are recovered. The cleaned HTML document is fed to a source language compliant tree parser.

SRV, on the other hand, annotates HTML source documents with predefined feature tags when pre-processing (Freitag 1998). That is, the pre-processing in SRV has annotated all useful information manually. Actually, such artificial processing in SRV does not have practical meaning. WHISK (Soderland 1999) can also be used to extract information from HTML. WHISK uses manually created segmentation for multi-slots documents. Each instance of template is separated by special token strings manually. For instance, instances are separated with tag “<HR>”. Because such separation can not processed automatically, WHISK is not suitable for full automatic IE in the practical application.

Compared to the related systems, Pre-processing for HTML in CapturePlus is full automatic and uses W3C recommend standard tool. Furthermore, other systems represent HTML in their own structures and provide isolated solutions. CapturePlus, conversely, converts HTML to well-formed standard XML-style documents for later processing. Hence, CapturePlus provides a way to use standard technologies to solve various problem rather than some isolated single solutions. This way to standard technology has practical advantages such as reducing developing time, preventing errors in implementing, better portability to other domains etc.

Once documents are pre-processed, the results are fed to the underlying IE components. The next step after pre-processing is feature extracting, that is, all relevant useful contents are to be extracted. Note, feature extracting finds out only the useful information but does not fill them in template yet. In next section we will introduce feature extracting in detail.