7 Feature Extracting

Following pre-processing, the next step is Features Extracting. This step is corresponding to the Name Extracting task defined by MUC. As introduced in Section 5.1.2, there are in total three steps within feature extraction: Text Zone recognising, content extracting and result merging. The main role of Text Zone recognising is to constrain the extracting area. Content extracting performs the underlying NE task. With various techniques, required contents are extracted within defined text zones exactly. The final step in feature extracting is results merging. Duplicating or overlapping of results is in this step detected and recovered.

In CapturePlus, the system extracts features as following: first of all, then system loads an pre-defined empty template. Then, for each feature defined in a template, the system loads the Search Profile assigned for this feature. The next, the system performs extracting in three steps described above. After completing the extraction of one feature, the system goes to the next one and does the same steps for other features. Such operation is repeated until all features are extracted. Found information is stored in a result list with string value, begin position and length in document and hitting relevance.

Because contents extracting is carried out by variegated algorithms, results can be redundant or duplicated. Hence, such results must be merged into one result list and then fed to subsequent TE-Agent. Finally, all results are add in a hit list sorted by position. This list becomes then the input of the following TE-Agent.

The workflow of feature extracting is illustrated in following Figure 24.
7.1 Text Zone Recognising

The input of Text Zone recognising is pre-processed document. Text Zones are defined as part of a document, where useful contents can be extracted.

Due to the difference between normal and HTML documents, methods for detecting Text Zones are also different. For normal documents, Text Zones are mainly defined in two ways: by pre- and post-constraints or by hierarchical structures generated in the phase pre-processing, while for HTML document XSL-patterns are used, in order to get the text zones enclosed by tags. The condition for text zone detecting for HTML documents is that HTML files are already converted to well-formed XML documents in the pre-processing phase.
In normal documents, if structure information (for example, document is segmented in pre-processing) is found after pre-processing, Text Zones detection normally relies on such structure information.

Another type of text zone could be dependent on the syntactic property. For instance, in the practice, some contents have significant prefix or suffix strings, which do not belong to the underlying contents, but can help the system to locate the useful content. Such prefix and suffix strings can therefore be used to detect Text Zones.

### 7.1.1 Text Zone Recognising for normal documents

For normal documents, Text Zone recognising is implemented as extraction constraints. Extraction constraints are defined as normal extracting rules, which define begin and end tokens or significant keywords for text zones. CapturePlus uses two type of constraints for Text Zone detecting in normal document: Title keywords and pre- and post tokens. The first type of constraints can be used if document is segmented, while the later is designed for such contents that have significant prefix and/or suffix.

#### 7.1.1.1 Title Keywords

Industrial documents such as product description, industrial tendering etc. are normally semi-structured. Various topics are normally subdivided by sections and sub-sections. Normally, titles of sections describe their topics\(^3\). If to be extracted contents are topics dependent, words in title can be used to select wanted sections. For instance, given the document in Figure 25, the interesting contents are names and telephones of partners for Industrial Automation Systems. Assuming that document is segmented in the phase pre-processing, Text Zones can be defined as section, whose title contains keywords “automation” and “systems”.

Because a document has a hierarchical structure, determining Text Zones could be very complicated. For example, if keyword A is either in title level 1 or in subtitle level 2, the system has to decide which level is to be assigned as Text Zones. Furthermore, if one title may not cover all keywords, a decision must also be made whether this section is assigned as Text Zone. To make decision easily, relevance of each section in the whole hierarchical structure is calculated. Given a list of keywords \( K = \{k_1, k_2, ..., k_n\} \), the relevance \( R \) for each section \( i \) is defined as

\[
R_i = \frac{\sum k_i}{N} + \frac{\sum R_j}{J}
\]

\(^3\) This property is only significant for industrial documents. Many normally documents, such as stock price, newsgroup announcements, etc., however, do not have such definition of topics.
where \( K_i \) are keywords in title i, \( N \) is the total number of keywords, \( R_{ij} \) is relevance of subsections and \( J \) is total number of subsections. To ensure the relevance is smaller than 1, all calculated relevances that are greater than 1 are set to one. The decision is made very simply: section is assigned as Text Zone if its relevance greater than a defined threshold. If duplication exists (e.g., both section and subsection are selected), the biggest part is then assigned as Text Zone. In following an example is given, in order to explain such selecting algorithm.

<table>
<thead>
<tr>
<th><strong>Our Partners for Sales &amp; Marketing</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>- Motors &amp; Drives</strong></td>
</tr>
<tr>
<td><strong>Motors:</strong></td>
</tr>
<tr>
<td>B. Grimm Power Engineering Co., Ltd./Tel. +66 2 379-4433</td>
</tr>
<tr>
<td>Palawatr Co., Ltd./Tel. +66 2 811-9022</td>
</tr>
<tr>
<td>UHM Co., Ltd./Tel. +66 2 994-5464-6</td>
</tr>
<tr>
<td>Zuellig Industrial Co., Ltd./Tel. +66 2 656-8710-54 ext.4330</td>
</tr>
<tr>
<td><strong>Drives:</strong></td>
</tr>
<tr>
<td>Brainics Technology Co., Ltd./Tel. +66 2 729-4833</td>
</tr>
<tr>
<td>Watford Control Co., Ltd./Tel. +66 2 941-0800</td>
</tr>
<tr>
<td>- Electrical +Installation Technology</td>
</tr>
<tr>
<td>B. Grimm Power Engineering Co., Ltd./Tel. +66 2 379-4433</td>
</tr>
<tr>
<td>Pan Power Engineering Co., Ltd./Tel. +66 2 938-4043-4</td>
</tr>
<tr>
<td>S.P. Wires &amp; Cables Co., Ltd./Tel. +66 2 434-0099</td>
</tr>
<tr>
<td><strong>- Industrial Automation Systems</strong></td>
</tr>
<tr>
<td>Berli Jucker Public Co., Ltd./Tel. +66 2 367-1140-47</td>
</tr>
<tr>
<td>Industrial Electrical Co., Ltd./Tel. +66 2 642-6700</td>
</tr>
<tr>
<td>Rickermann Thailand Co., Ltd./Tel. +66 2 751-9111</td>
</tr>
<tr>
<td>Siam Rajathaneey Co., Ltd./Tel. +66 2 713-5010-25</td>
</tr>
</tbody>
</table>

Figure 25: Example for Text Zones detecting with title keywords
Figure 26: Sample Structure for Selecting Text Zones with title keywords

Figure 26 shows a sample document structure. Assumed the list of keywords contains three elements: k1, k2, k3. Sections, whose titles contain such keywords, are annotated in the figure. The default threshold is defined for example as 0.4.

We calculate relevance value for each section. The calculate table is showed as Table 5. Decision of Text Zones: “section 1.1.1” and “section 1.2”, whose relevance values are greater than defined threshold.

<table>
<thead>
<tr>
<th>Section</th>
<th>Relevance Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>0.667</td>
</tr>
<tr>
<td>1.1.2</td>
<td>0.333</td>
</tr>
<tr>
<td>1.1.3</td>
<td>0.000</td>
</tr>
<tr>
<td>1.1</td>
<td>0.333</td>
</tr>
<tr>
<td>1.2.1</td>
<td>0.333</td>
</tr>
<tr>
<td>1.2.2</td>
<td>0.333</td>
</tr>
<tr>
<td>1.2.3</td>
<td>0.000</td>
</tr>
<tr>
<td>1.2</td>
<td>0.444</td>
</tr>
<tr>
<td>1.3</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>0.259</td>
</tr>
</tbody>
</table>

Table 5 Calculating for selecting text zones

Note, that selecting of Text Zones is the first step for feature extracting. The main role of this step is to help following underlying feature extracting processing to focus on interesting areas in document. Hence, in this step high precision is not necessary, but high recall must be guaranteed. That is, detected text zones must cover all potential content instances. High precision can be reached in later steps (e.g. in the phase template filling). This selecting algorithm makes it possible that sections with higher relevance are assigned
as Text Zones. To enhance the selecting quality, a more complicated algorithm could be investigated, such as keywords with weighting, adaptive threshold, adaptive selecting strategy etc.

Furthermore, Text Zones can be defined much finer to paragraph or sentence level. Title keywords can also be searched as general keywords in the whole text areas. In some cases, Text Zones in paragraph or sentence level play an important role. For instance, paragraph with keyword “conclusion” contains most evaluation or decision statement of a report. Some information retrieval methods are suitable for this approach. That is, automatic learning and generating keywords used in information retrieval can be performed by given examples here too. If Text Zones are only defined in paragraph or sentence levels, documents can be segmented automatically (see Section 6.1.2). A full automatic Information Extraction becomes therefore possible.

7.1.1.2 Begin and End Constraints

Text Zones can be detecting with title keywords, if documents already are pre-processed and segmented with hierarchical structures. Some documents, however, do not have very clear structure. Typically, documents with defined layout, which is filled with data extracted from database, belong to this gender. Examples of such documents considered here include; are product catalogue, CIA Factbook some configuration files etc.

Normally, such documents describe objects with properties. Such properties consist of two fields: name filed and value field. In the majority of cases, the interesting data for IE is in the value field. The name field, on the other hand, is just significant notation for subsequent value. For these documents, name field can be considered as constraints, which can restrict the extracting area for underlying values. If name fields are located before value fields, we defined them as begin constraints, otherwise, as end constraints. For instance, in the CIA Factbook (see also Section 4.1), each value filed has a begin constraint (such as “population:” etc.). Another type of constraint is prefix and suffix. For instance, to extract the currency, the suffix “$”, “DM” or “€” become very strong constraint. In generally, Text Zones are defined as text area between begin and end constraints.

As soon as text zones are restricted by Begin- and End-constraints, later extracting can be performed just within the text zones. In some special cases, contents in Text Zones defined by constraints are just the underlying information. For instance, in the CIA Factbook (see also Section 4.1), content value for “population:” is the total Text within the text zone, which have begin constraint name field “population:” and end constraint new line. In there cases, as soon as text zones are detected, following contents extracting is already finished as the same. In common cases, however, underlying content is only part of the text zone defined by constraints. Therefore, the following contents
extracting is still necessary. Anyway, extracting algorithms can be simplified considerably (e.g. with using regular expressions).

Here we will discuss about types of constraints. The simplest constrain is constant string such as pre- and postfix, name filed direct ahead the value etc. In a general case, constraints must not always be constant strings. Actually, complicated fields can also be a constraint types. To define and find such complicated constraints, we need to use various methods. In fact, methods for detecting of constraints are the same as underlying extracting algorithm. The only difference lies in the relationship to the extracted contents: for the latter are the underlying contents, for the former are the constraints of underlying contents. That is, constraints can be simply defined as search profiles too. In practice, however, constraints are normally constant strings or contents that can be defined by regular expressions simply, in order to enhance the total processing time performance. Hence, in CapturePlus, two types of constraints are used: constant strings and regular expressions.

There are two situations for construction of constraints: a constraint has complete begin and end part, a constraint has only begin or only end part. If both begin and end constraints are defined, Text Zones are enclosed by those two constraints. Otherwise, text zones are defined as the document parts between two begin or two end constraints. Algorithm for detecting of Text Zones is described in Figure 27. The output of this step is a list that contains text zones defined by begin position and length in document.

```
IF defined begin and end constraints
    WHILE there is constraints in documents
        Find position of begin constraint P1 in document
        Find position of end constraint P2 after P1
        Set area between P1 and P2 as Text Zone
        Move to next position after P2
    END WHILE
ELSE
    Find position of first constraint P1 in document
    WHILE there is constraint in documents
        Find position of constraint P2 after P1
        Set area between P1 and P2 as Text Zone
        Set P1 = P2
        Move to next position after P2
    END WHILE
END IF
```

Figure 27: Algorithm for detecting Text Zone using constraints
7.1.1.3 Related Work

Techniques for Text Zone recognition with keywords are widely used in many IE systems. Some MUC systems (CIRCUS Lehnert 1991, SRV Freitag 1998, AutoSlog Riloff 1993, Aseltine 1999 etc.) use trigger words to get the related sentences. The underlying extracting of values is taken within the recognised sentences. For instance, to detect a text zone for terrorist, the word “boom” is defined as trigger word. As soon as this trigger word “boom” occurs, its sentence is considered as interesting area and the following extracting is will be processed for this sentence. Such approach are similar to evaluation keywords in title. The difference is that MUC system’s triggers are evaluated only at the sentence level, while our keywords can be used in different levels (such as clause, paragraph or sentence).

Extracting within defined constraints is used in many recent IE systems. RoboTag has been developed which uses decision trees to learn the location of slot-fillers in a document (Bennett, Aone, & Lovell, 1997). In the extracting phase RoboTag evaluates the learned decision tree classifiers on the new text to produce a list of potential begin and end tags for each tag type. A matching algorithm has to decide the best possible pairing of the begin and end tags for each type. Each potential begin and end tag produced by the decision tree also has a confidence rating. A scoring function is used to evaluate the relative merits of different sets of pairs. There are three parts of scoring function for a pair. The first is the confidence with which the begin tag tree classifies the token as a good begin tag. The second component is the end tag tree confidence. The last part is a distance score, which is calculated from the tag length, mean distance, and match length preference. Each of the three length preferences uses an appropriate bias to the way in which these inputs are combined.

The FACILE system (Fast and Accurate Categorization of Information by Language Engineering) is based on this two-tiered text zone detecting approach (Ciravegna et al. 1999). Unlike CapturePlus, FACILE performs text zone recognising before pre-processing. The output of text zone recognising is a marked up document. The technique for text zone recognising is not described in the paper. It was mentioned in a personal discussion with the author in ECCAI 2000 (Berlin) that the text zone is defined by fixed length for defined document domain. This primitive technique is applied in some restricted application domains (such as a Spanish saving bank). Actually, such Text Zone recognition is only isolated emergency solution for their application domain.
7.1.2 Text Zone Recognising for Online (HTML) documents

Unlike to normal documents, information in online HTML is enclosed in marked up document. The title keyword approach used in normal document is therefore not suitable for HTML document, because HTML documents can not be segmented clearly as well as normally documents. On the other hand, the constraints approaches can still be applied for HTML documents. For instance, tags such as “<table>” or “<P>” can be defined as begin or end constraints. The disadvantage of the constraints approach used in online-documents is that the tag constraints can be always the same for many different values. For instance, values in a table could have the same begin constrains as “<TD>”. This fact can cause poor precision because many irrelevant values can be considered as useful information. Hence, for online-documents we have to use other approach to detect Text Zones. As mentioned already, HTML has implicit hierarchical structure and therefore can be converted to well-formed XML documents in the phase pre-processing (see Section 6.2). As soon as HTML document is converted to XML, text zone recognition can be processed as executing XSL-patterns.

7.1.2.1 XSL Patterns

XSL-pattern is used to find element nodes in XML document. An XSL pattern describes a branch of an XML tree, which in turn consists of a set of hierarchical nodes. The syntax used for XSL pattern is somewhat similar to that used to specify paths on a disk drive. For example, given an XML document showed in Figure 28, the pattern addressbook/contact/phone selects phone elements that appear beneath a contact element, which itself appears beneath an addressbook element. Also XSL-pattern supports wildcard such as “*”. For instance, the results after executing XSL-pattern “addressbook/contact/*” could be all values in sub elements “name” and “phone”. Each content can have one or more XSL-pattern(s). With the help of patterns we can locate the text zones of the contents. For some contents, the node text corresponds exactly to the to-be-extracted Value. Such contents are usually elements in table or generated by robot automatically in web server. In this case, after executing XSL-patterns, the following task of contents extracting is already finished.
Manuel generating XSL Patterns is time consuming and error-prone. A XSL-pattern in a real application for a simple content can be very large. For instance, Figure 29 shows a sample XSL-pattern to detect the current temperature value in the CNN weather forecast web sites. A learning algorithm must be therefore developed, in order to perform an automatic generation of XSL-patterns from given examples. In this section we will introduce a supervised bottom-up learning algorithm to generate XSL-patterns automatically. The algorithm is at first presented. A simple worked examples is also given, so that the principle can be better understand.

To create XSL-patterns automatically, in the training phase, we have to generate the XML-patterns according the given examples by user. We use a standard XML parser to find out XML text nodes specified by examples. Because the examples are given by positions, we have to index all XML text nodes with position information. For each found XML node a XSL pattern can be generated. The generated XSL pattern is processed bottom-up: from the actual XML text node we go to the parent nodes until the root is reached. An XSL pattern is then generated as the path of node names with their attributes. Once we get all XSL patterns, we can use a supervised learning algorithm to induce the patterns. The algorithm is shown in Figure 30.
Figure 29: A real XSL-pattern to extract a temperature value from CNN weather web site

```html
<html/body[@bgcolor = "#FFFFFF" ][@link = "#000099" ][@vlink = "#666666" ]/form/table/tr/td[@rowspan = "2" ][@width = "163" ][@valign = "TOP" ]/table[@border = "0" ][@cellspacing = "0" ][@cellpadding = "0" ][@bgcolor = "#FFFF00" ][@border = "0" ][@width = "163" ]/tr/td[@width = "163" ][@bgcolor = "#FFFFFF" ]/table[@width = "163" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@bgcolor = "#FFFF00" ][@width = "163" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ]/table[@width = "163" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0" ][@border = "0" ][@width = "163" ][@cellpadding = "0" ][@bgcolor = "#FFFFFF" ][@align = "top" ][@valign = "TOP" ][@cellpadding = "0"
```

Figure 30: Algorithm for learning of XSL-Patterns.

```
WHILE Example Set is not empty
    Get an example
    FOR each trained XSL pattern in Training
        Get a trained pattern Pi
        New Pattern = GetOverlappingPattern (trained pattern, input pattern)
        Calculate the error rate E_i for New Pattern
    END FOR
    IF all E_i > Threshold
        Add the input pattern in Training
    ELSE
        Add new Pattern with the lowest E_i in Training
    END IF
    Reorganize the Training to content only disjunctive patterns
Delete example from Example Set
END WHILE
```

The system induces rules bottom-up. For each iteration, one XSL Pattern is input. an overlapping XSL pattern of the input and trained pattern is created. The clue of generating overlapping pattern is to try to create more generic XSL pattern for this feature. The overlapping pattern covers all contents extracted by input patterns. For example, the overlapping pattern of html/body/table/td/tr and html/body/table/td/font is assigned as html/body/table/td/*.
$n$ is the number of extractions made on the training set and $e$ is the number of error among those extraction.

$$\text{Error rate} = \frac{(e + 1)}{(n+1)}$$

If the error rates are lower than defined threshold, the overlapping pattern will be set as trained pattern with the lowest error rate. Otherwise the input pattern is set as a new trained pattern. After setting overlapping or new pattern, the trained pattern set is reorganised to make sure that there are only disjunctive patterns in the trained pattern set.

In Figure 31 we give an example to illustrate how XSL patterns are inductive generated. To note is that this simple example explains only a principle how inductive XSL pattern can be generated. In the real application, however, the XSL-pattern is much more complicated. The calculation of overlapping XSL pattern is performed also with respect to the attributes of nodes.

7.1.2.3 Related Work

Information Extraction from Web Pages is a topical task of the IE area. Techniques, which try to extract useful information from HTML, are classified as wrappers. Various systems developed wrappers such as (Doorenbos et al. 1997, Kushmeric et al. 1997, Muslea 1999). Particularly there are two basic types of wrappers: Tag-based and structure-based wrapper.

The Tag-based wrappers consider the HTML-documents as a flat text string with mark-up tags. Such wrappers try to identify tags-properties around underlying contents und use such properties to perform extracting. STALKER (Muslea 1999), for example, uses finite automata principle to contribute wrappers. For STALKER a document is a sequence of tokens $S$ (e.g., words, numbers, HTML tags, etc). It follows that the content of the root node in the embedded catalog (EC) tree is the whole sequence $S$, while the content of each of its children is a subsequence of $S$. More generally, the content of an arbitrary node $x$ represents a subsequence of the content of its parent $p$. A key idea underlying this work is that the extraction rules can be based on “landmarks” (i.e., groups of consecutive tokens) that enable a wrapper to locate the content of $x$ within the content of $p$.

For instance, a STALKER wrapper rule can be defined as following:

$$R_1 = \text{SkipTo}(\text{Name}) \text{SkipTo}(<b>)$$

or

$$R_2 = \text{SkipTo}(\text{Name Punctuation HtmlTag})$$
Given are 4 examples, the XSL patterns\(^4\) for each example are:

\(P1: \text{html/body/table/tr/td/b}\)
\(P2: \text{html/body/table/tr/td/font}\)
\(P3: \text{html/body/table/tr/font}\)
\(P4: \text{html/body/table/tr/p/font}\)

**Iteration 1:** Examples: \((P1, P2, P3, P4)\), Training: (Empty)
Input pattern: \(P1\)
Add \(P1\) as \(T1\) into Training, remove \(P1\) from Examples

**Iteration 2:** Examples: \((P2, P3, P4)\), Training: \((T1)\)
Input pattern: \(P2\)
Overlap \(O1\) \((P2, T1)\): \text{html/body/table/tr/td/*}
Error rate: calculated as 0.1, Pattern accepted
Add in \(O1\) as \(T2\) into Training, remove \(P2\) from Example
Reorganize Training, in Training \((T2)\)

**Iteration 3:** Examples: \((P3, P4)\), Training: \((T2)\)
Input pattern: \(P3\)
Overlap \(O1\) \((P3, T2)\): empty
Add in \(P3\) as \(T3\) into Training, remove \(P3\) from Example
Reorganize Training, in Training \((T2, T3)\)

**Iteration 4:** Examples: \((P4)\), Training: \((T2, T3)\)
Input pattern: \(P4\)
Overlap \(O2\) \((P4, T2)\): \text{html/body/table/tr/*/}
Error rate: calculated as 0.6, Pattern not accepted
Overlap \(O3\) \((P4, T3)\): empty
Add in \(P4\) as \(T4\) into Training, remove \(P4\) from Example
Reorganize Training, in Training \((T2, T3, T4)\)

**Training result:** \((T2, T3, T4)\)

**Figure 31:** An working example for learning XSL-Pattern from given examples

\(R1\) has the meaning “ignore everything until you find a Name landmark, and then, again, ignore everything until you find \(<b>\)”, while \(R2\) is interpreted as “ignore all tokens until you find a 3-token landmark that consists of the token Name, immediately followed by a punctuation symbol and an HTML tag.”

\(^4\) To simplify the example, all attributes in nodes are not showed in example patterns. In the real implementation the attributes are used also to calculate overlapping pattern. Sometimes the attributes play an important role to distinguish the XML nodes.
Such rules can be learned and generated automatically with an inductive learning algorithm. STALKER is a typical sequential covering algorithm: as long as there are some uncovered positive examples, it tries to learn a perfect disjunct (i.e., a linear LA that accepts only true positives). When all the positive examples are covered, STALKER returns the solution, which consists of an ordered list of all learned disjuncts.

According to the author, the function for learning perfect disjuncts is a greedy algorithm: it generates an initial set of candidates and repeatedly selects and refines the best refining candidate until it either finds a perfect disjunct, or runs out of candidates.

Compared to CapturePlus, STALKER considers HTML as a non-hierarchical structure but only as flat set of strings with some useful tokens (tags). This feature of STALKER allows a very quick extracting speed, because the SkipTo operation makes a straightforward pattern matcher. In CapturePlus, conversely, document has to be at first parsed by an XML parser. The XSL pattern matching in also more complicated and therefore more computational intensive than the algorithm used in STALKER. However, STALKER faces unsolved problems right away, if the input document is complicated. Today, the professional web pages contain much layouts and formatting information. Simple flat SkipTo operation could extract too many false instances. On the other hand, CapturePlus uses not only hierarchical tags but also attributes information, which can help to locate the underlying content exactly. As same as CapturePlus, STALKER uses also supervised leaning algorithm to generate such extracting rules automatically, that can cover all given examples.

Similar systems are e.g. (Doorenbos et al. 1997, Kushmeric et al. 1997). Such systems can extract information with high reliability based on the HTML tags used to delimit table entries.

Structure-based wrappers, on the other hand, consider the HTML-documents as hierarchical structured. For the structure-based wrappers, HTML-documents have implies tree-structure that defined by tags. The root of such structure is tag “<HTML>”, such root has at least two sub elements that denoted by tags “<HEAD>” and “<BODY>”. Using this method a tree-style structure can be generated top-down from the root tag “<HTML>”. As soon as the structure is generated, extracting rules can be developed like a file directory path. An example of such wrapper-systems is XWRAPPER.

XWRAPPER (Liu L, et al. 2000) is an interactive system for semi-automatic construction of wrappers for Web information sources. The goal of this work can be informally stated as the transformation of HTML input into XML output. A main technical challenge is to discover boundaries of meaningful objects
(such as regions and semantic tokens) in a Web document, to distinguish the information content from their meta-data description, and to recognize and encode the metadata explicitly in the XML output.

For an HTML document, the information extraction phase takes as input a parse tree generated by the syntactical normalizer. It first interacts with the user to identify the semantic tokens (a group of syntactic tokens that logically belong together) and the important hierarchical structure. Then it annotates the tree nodes with semantic tokens in comma-delimited format and nesting hierarchy in context-free grammar.

Extraction process involves the following steps:

In the pre-processing step, HTML documents are at first cleaned. That is, the bad tokens are repaired. The clean HTML document is fed to a source-language-compliant tree parser. A tree structure of source document is generated.

The second step is region extraction. Region extraction is performed via an interactive interface, which lets the XWRAP user guide the identification of important regions in the source document, including table regions, paragraph regions, bullet-list regions, etc. The output of this step is the set of region extraction rules that identify regions of interest from the parse tree. With the interaction, user can define interesting regions. The system generates then extraction rules for such regions. For example, a region extraction rule can be defined as $HTML.BODY.TABLE[0].TR[0].TD[4].TABLE[2]$.

Finally, Semantic Token extraction is processed. The semantic-token extractor is an interactive program, which guides a wrapper developer to walk through the tree structure generated by the syntactic normalizer, and highlight the semantic tokens of interest in the source document page. The output of this step includes a set of semantic token extraction rules that can be used to locate and extract the semantic tokens of interest of the Web documents from the same web site, and a comma-delimited file, containing all the element type and element value pairs of interest.

XWRAPPER is an interactive system for Information Extraction from HTML. However, compared to CapturePlus, XWRAPPER does not provide automatically generating rules. A GUI-software helps users to define rules. Furthermore, XWRAPPER uses its self-defined extracting rules and has non-standard techniques. CapturePlus, on the other hand, uses standard XSL-patterns to perform Text Zone recognition. Such standard solution has a significant advantage, that this approach can be realised and integrated into existing system quickly and easily. The XSL-pattern approach can also be applied for XML-documents immediately. With the growing requirement on
information extracting from XML documents, this approach with learning and extracting XSL-patterns can be used in many practical domains widely.

7.2 Contents Extracting

After detecting text zones, contents extraction can be applied. Within the defined text zones, useful contents have to be exactly located. Contexts

In the practical domains, totally different types of contest can occur. That is, all possible information such as number, date, name, description, property etc. can be defined as underlying contents. Because of the variety of contents, it is impossible to have a general technique in IE system, which can be used for all types of contents. Suitable IE techniques are therefore to be used to solve corresponding part problems. Hence, in our system, contents extracting uses various techniques; from simple full text extracting, extracting with similarity, extracting with regular expressions to complicated trainable token based rules. To define such rules, Search Profile is used. XML provides an uniform description format used to define all different rules (see Section 5.2). In this section different extracting techniques are introduced. Their advantages and disadvantages are discussed.

7.2.1 Simple Extracting Techniques

In this section we will begin to introduce some primitive extracting techniques, namely “Full Text Extracting”, “Extracting with Similarity” and “Regular Expressions”.

The simplest extracting form is matching to full text patterns. Actually, such extracting mechanism does simply full text searching with some searching options (such as case sensitive, whole word etc.) within predefined text zones. Typical contents used by this techniques are domain specific names, constants etc.

An extended technique for full text extracting is to use a thesaurus. A thesaurus describes words with relations. Table 6 lists possible relations between words and gives simple examples (Xiao 1998). By full text extracting with thesaurus, system searches all related words stored in the thesaurus to the current full text string.
<table>
<thead>
<tr>
<th>Short Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonym</td>
<td>Equivalence relation</td>
</tr>
<tr>
<td></td>
<td>lift, elevator</td>
</tr>
<tr>
<td>Hypernym</td>
<td>Content of a word contains content of another word</td>
</tr>
<tr>
<td></td>
<td>computer, machine</td>
</tr>
<tr>
<td>Hyponym</td>
<td>Contrary to Hypernym</td>
</tr>
<tr>
<td></td>
<td>personal computer, computer</td>
</tr>
<tr>
<td>Meronym</td>
<td>A term if a part of another</td>
</tr>
<tr>
<td></td>
<td>branch, tree</td>
</tr>
<tr>
<td>Antonym</td>
<td>Terms has contra meaning</td>
</tr>
<tr>
<td></td>
<td>black, white</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linguistic variants</td>
</tr>
<tr>
<td></td>
<td>go, went</td>
</tr>
<tr>
<td>Trigger</td>
<td>Pair of word co-occurrence with high frequency</td>
</tr>
<tr>
<td></td>
<td>patient, drug</td>
</tr>
<tr>
<td>Causality-Relation</td>
<td>A verb is causative, when another word is the result</td>
</tr>
<tr>
<td></td>
<td>give, have</td>
</tr>
<tr>
<td>„sounds like“</td>
<td>Similarly spoken</td>
</tr>
<tr>
<td></td>
<td>computer, commuter</td>
</tr>
<tr>
<td>Acronym</td>
<td>Short form of word terms</td>
</tr>
<tr>
<td></td>
<td>PC, personal computer</td>
</tr>
</tbody>
</table>

**Table 6 Relations in a thesaurus**

Under all relations defined in a thesaurus, synonym is typically the most important. For example, all country names can be defined as synonym “country name”. Such synonyms are stored in a thesaurus and loaded in the extracting phase. Exists e.g. synonyms for “country name”, full text extracting can be performed as searching all relevant country names defined in thesaurus. Other examples are product names in one product family, hotel names in hotel chain etc. Actually, such synonyms are named “background knowledge” in other systems (Sonderland 1999). Background knowledge is very a powerful aid for high quality information extraction. Because only words defined in a thesaurus are searched for, precision and recall could be enhanced. Synonyms can be created manually, or by given examples. However, in the later case, training examples must be complete, in order to cover all possible synonym words. A disadvantage of using thesaurus is lack of flexibility. Constructing of background knowledge is time consuming.

Two types of thesauri are used in our system: an automatically created domain specific thesaurus and a standard thesaurus. The first one is created by given domain specific examples. The second one is a standard NLP component (WordNet). A thesaurus editor provides an interactive GUI to allow users to add, delete, and change words and relations.

Full text extracting can be used to extract constants. Another important usage is synonym extracting. For instance, if country names are stored as background knowledge or given as list of keywords explicitly, extracting country names from documents becomes trivial. Such synonym can also be learned from examples. Such component to gather all synonym words from given examples is called synonym collector in CpaturePlus.

Extracting with similarity can find some contents with symbolic morphologic relation. For example, extracting the word “product” with similarity will find
contents such as “product”, “products”, “productivity” etc. Extracting with similarity is suitable for such documents with noise. For example, if documents are generated by scanning and by OCR (Optical Character Recognising), errors can occur during the converting. Characters such as “0” and “o”, or “1 (one)” and “I” are often confused. Exact pattern matching can ignore the features with error characters. Extracting with similarity can overcome such problems. Similarity of words can be calculated with algorithm. Another advantage is that such technique can find words with morphological relation without help of a thesaurus. However, extracting with similarity is a fuzzy technique. It can bring many words that have merely symbolic similarity but not the semantic relationship and cause therefore poor precision.

As introduced in Section 3.2.1, regular expressions are one of the basic IE techniques. Combined with Begin- and End-constraints (Text Zones recognising), regular expressions can be used to extract some features such as date, number, section number etc.

Table 7 summarises above three simple extracting mechanisms.

<table>
<thead>
<tr>
<th></th>
<th>Suitable for</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Text Extracting</td>
<td>domain specific names, constants etc.</td>
<td>Performs high precision and recall</td>
<td>Expensive for generating background knowledge, lack of flexibility</td>
</tr>
<tr>
<td>Extracting with Similarity</td>
<td>Contents with noise, morphological variants</td>
<td>Domain independence, without background knowledge, high recall</td>
<td>Sometimes bad precision</td>
</tr>
<tr>
<td>Regular Expressions</td>
<td>Contents with significant syntactic properties</td>
<td>Good precision, can be generated from examples automatically</td>
<td>Sometimes need context constraints to improve extracting quality</td>
</tr>
</tbody>
</table>

Table 7 Summarisation of simple extracting techniques

7.2.2 Extracting with Token Based Rules

Token-Based Rule (TBR) was initially developed by DFKI (The German Research Centre for Artificial Intelligence GmbH).

The token-based rules consist of three parts: precore, core and postcore. While the core describes the underlying content, the pre- and postcore define information before and after the content. The rule distinguishes number, string and tokens. Tokens are white space and special characters (such as “,”, “!”, “&” etc.). Number and string are separated by tokens. The formal definition is shown as following:
Token-based Rule := Relevance Precore “{c” Core “c}” Postcore
Precore := (Token | SkipOperator)*
Core := (Token | SkipOperator |)+
Postcore := (Token | SkipOperator)*
Relevance := float number between 0.0 and 1.0
Token := Digit | String | Special Character
Skipoperator := “[“ Number of skip tokens “]”
Digit := [0-9]+
String := [a-z,A-Z]+
Special Character := ASCII Characters \ (Digit, String)

For example, the rule sf [1] {c <int> c} bed extracts a number (between “{c” and “c}” is the underlying core), which has two strings before it and one string after it. One of the two strings before the core is “sf”, another is any desired string (specified by [1]) or can be empty. The string after the core is given as “bed”. Figure 32 shows an example usage of TBR.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Possible Instances</th>
</tr>
</thead>
</table>
| Abfahrt [2] Zuges | ... Abfahrt eines Zuges ...
| | ... Abfahrt des Zuges ...
| | ... Abfahrt des eigenen Zuges ...
| <int> [1] C | ... 40 °C ...
| | ... 30 °C ...
| <int> [3] Haltestelle [2] H | ... L2H1 und endet mit Haltestelle L1H1 bis L1H21 ...

Figure 32: Token-based rules (TBR)

Extracting with Token-Based rules is processed as following: at first, documents are tokenised. System decomposes total text in three types of string:

- Number, which consists of digital characters
- Alpha data, which consists of alpha characters such as “a, b, c, ...z, A...Z”
- Token characters: all other characters that do not belong to above two groups. Typical tokens are white space and special characters such as “!”, “,” etc.

After tokenising, the system interprets token-based rules as state automata and matches pattern within a defined text zone. As soon as the matching is successful, the Extractor outputs result with position information and relevance value.
Token-based rules can be learned and generated from examples automatically. In the training phase, the Rule Trainer indexes document before training. Characters in content are typed as number, string and tokens specified by special characters. The Trainer uses an inductive supervised training algorithm. The trainer generates at first a rule only with primitive core pattern. Then the trainer tries to extend rule by adding pre- or postcore and skipping operators. Extended rules are then evaluated overall all examples. New rules are generated if the extended rules are evaluated with satisfied result. The output of training is a set of token-based rules with relevance. Figure 33 shows a simple example of learning token-based rules. While the float at the begin of rule gives relevance of the rule, the following string defines then precore, core and postcore.

```
Given training examples as input (fat characters are the contents)
Speed: 20 km/h,
Speed: 14.5 km/h,
Speed: 2000 m/h
....

The training result could be:
0.8 Speed {c <int> c} km/h
0.5 Speed {c <int> . <int> c} km/h
0.6 Speed {c <int> c} m/h
...
```

Figure 33: Example of learning token-based rules

Token-based rules are typical syntactic rules. They work similarly to as regular expression but with respect to contexts (precore and postcore). Such rules can be widely used to extract contents from industrial documents. Temperature, industrial Norms (e.g. DIS213, ISO 3456 etc.), speed, physical distance (e.g. 100 km, 55 cm etc.), time point, time ranges etc. are examples of concepts that can be extracted with token-based rules. Based on the basic token-based rules, more complicated extracting technique can be developed. Hierarchical concept is an example for extracting some difficult contents with help of hierarchical replacing of token strings (see Section 8).

7.2.3 Extracting with Keywords

The difference between Information Extraction (IE) and Information Retrieval (IR) is already discussed in Section 2.1. However, some IR techniques can be used for some IE tasks. Such tasks are for example extracting a short description of a product from product catalogue, extracting a plain text description about travel destinations etc. For this case, documents have to be
pre-processed. Segments and boundaries (such as sentences, paragraphs, sections etc.) have to be found before IR methods can be used.

In Information Retrieval (IR) there is no lack of models: the Boolean (Salton 1968), the vector space, the probabilistic, and the fuzzy model are all well known (Crestani 1994). Our system uses (Xiao 2001b) weighted keywords to extract short descriptions from documents. In the training phase, the Keywords Trainer first eliminates stop words from the examples and then analyses statistical information about the words. Keywords are generated if the occurrence frequencies are greater than a threshold. The output of training is a set of keywords with relevance. For example, a keyword-based extracting rule can be defined as:

\[
\{0.40, \text{“travel”}, 0.37, \text{“accommodation”}, \ldots\}.
\]

The rule contains keyword “travel” with relevance 0.4, keyword “accommodation” with relevance 0.37 and other keywords with their weighted relevance.

In the extracting phase, the Keywords Extractor evaluates defined keywords within text zones and calculates then relevance value as a result of the evaluation. Relevance value \( R \) is given by following formula, where \( r_i \) is relevance of keywords found in content, \( N \) is total number of keywords.

\[
R = \frac{\sum r_i}{N}
\]

If the relevance is greater than a defined threshold, the total text zone is output as a result with position and relevance information.

To use the weighted keywords in information extraction, input documents must support the following two conditions:

1. Because keywords are evaluated within defined area, documents must be segmented at first. The segmentation can be processed in the pre-processing phase by Segmentor interactively (see Section 6.1.2). If documents are only segmented in paragraph or sentence level, segmentation can be processed automatically without interaction.

2. Contents must be distinguished by keywords easily. Normally, such contents are short description or a short conclusion. Traditional extraction technique such as regular expressions or token based rules are not suitable for such contents. Weighted keywords, however, can be used to extracted desired contents very efficiently.
7.2.4 Related Work

Content extracting is the core technique in total information extraction. The basic techniques are syntactic rules and Natural Language Processing (NLP) (see also Section 3). Advanced techniques are normally based on basic techniques and use various well-known artificial intelligence or machine learning methods.

In the basic technologies, the syntactic rules are widely used for extracting information, which has significant syntactic properties. This technique becomes a standard IE techniques because of the clear definition and good time performance. No dictionaries or lexicons are needed. Syntactic rules are suitable for such contents as time, speed, currency, name, number etc. Another advantage of syntactic rules is that rules can be language independence. They could be used in any language to extract contents, which have same syntactic properties. However, a simple syntactic rule is not able to extract contents exactly even if the contents have significant syntactic properties. Using syntactic rules alone can cause the result with either high precision but poor recall (too fine rules) or poor precision but high recall (too crude rules). Various constraints can enhance extracting performance considerably. Such constraints could be pre-patterns and post-pattern (Freitag 1998, Xiao 2001, Califf 1997), or semantic constraints (Embley 1999).

Natural language processing has been an active area of connectionist research for over a decade. The NLP algorithms are used to extract relevant data from grammatical, free text. Many advanced IE technologies rely on NLP techniques. NLP techniques are used not only for NE tasks, but also for TE and TR construction. The MUC researchers developed IE systems primarily based on NLP techniques (SAIC 1998). Actually, NLP technique ism one of the most important basic techniques used in IE. NLP techniques are especially suitable for free text or semi-structured text, which contains contents written with natural gram. However, NLP are hard tasks themselves and there are still some unsolved problems in NLP. One of them is so called Word Sense Disambiguation (WSD), whose aim is to identify the correct sense of a word in a particular context, among all of its senses defined in a dictionary or a thesaurus (ji & Huang 1997). If NLP (e.g. POS-tagging) is used as pre-processing of IE, the error caused by WSD problem can be propagated to later processing. Such earlier phase errors can be amplified in the later processing phase and are not recovered because other advanced techniques (such as many Machine Learning approaches) are based on NLP. To work around this problem almost all advanced algorithms use various syntactic and semantic constraints to identify the contents of interest.

Riloff (1994) introduced a practical system CIRCUS using Case-based reasoning (CBR) for information extraction. To create a set of cases for a document, for each sentence researchers collect all of the concept nodes
produced by CIRCUS and merge them into a case. A case is represented as a structure with five slots: signatures, perpetrators, victims, targets, and instruments. During training, each document in the training corpus is converted into a set of cases. Each document is represented as a set of cases, one for each sentence that produced at least one concept node. The resulting case base contains thousands of natural language contexts from hundreds of texts. The heart of the algorithm is its ability to accurately judge the relevancy of new cases. If any of the cases are deemed to be relevant then the document is classified as relevant, otherwise as irrelevant.

Actually, CBR techniques can be used in several IE tasks. In the lowest level CBR can be used to extract strings with similarity. This technique can avoid the problem of syntactic rules that the rules are either too fine or too crude. At the template filling level, filled templates can be stored in case base and the selection of elements in new template can be solved with CBR matching. Other tasks such as scenario template production (ST) and template relation (TR) can also be modeled in case base. However, in practice some problems in using CBR have to be faced: first, construction a case base is normally a individual solution. That is, for each situation a case base must be developed. Second, conversion documents to cases and calculation relevance between cases are also domain specific. This is a restriction for portability. Third, to get a good result, a case base must contain enough past cases. This causes in general that calculation relevance is computational intensive. The processing speed in a CBR system is normally not satisfactory for large text volumes.

Another machine learning method to be described is a generalisation of regression trees, that is also called model trees, one of the family of divide-and-conquer algorithms that includes categorical decision trees (such as C4.5, (Qunlian 1993)). C4.5 induces a decision tree in a greedy divide and conquer fashion. As the tree is being constructed, the choice of which feature to split on at a given node is made.

Recently, a system called RoboTag has been developed which uses decision trees to learn the location of slot-fillers in a document (Bennet 1997). RoboTag uses C4.5 to learn decision trees which predict whether a given token is the first token of a slot-filler or the final token of a slot-filler. The features available to the decision trees are the result of pre-processing the text: segmenting the text, performing morphological analysis and lexical lookup. The trees consider the token for which the decision is being made along with a fixed number of tokens around it. Once RoboTag has learned trees to identify start and end tokens for slot-fillers, it uses the trees to identify possible start and end tokens and then uses a matching algorithm to pair up start and end tokens to identify actual slot-fillers.

In the training phase some parameters have to be set in order to improve tagging performance. In the extracting phase RoboTag evaluates the learned
decision tree classifiers on the new text to produce a list of potential begin and end tags for each tag type. A matching algorithm has to decide the best possible pairing of the begin and end tags for each type. Each potential begins and end tag produced by the decision tree also has a confidence rating. A scoring function is used to evaluate the relative merits of different sets of pairings.

The key technique in C4.5 is to select training features. The extracting of C4.5 alone is very quick. However, the total processing speed is depended on the selection of features and the pre-processing of features. Robotag has e.g. four features: Token Type, POS, Location and Semantic Type. Such features are based on other basic IE technologies (such as token rules, POS tagging etc.). However, such features are selected only for defined problems and defined document class. So the selection of features is an individual solution. That means, that the portability of this approach is not optimal.

Another approach to information extraction is to use AI neural networks, as described in a paper by Dieter Merkl (1997a). Merkl uses hierarchical self-organising maps to extract key information from documents. A self-organising map (SOM) is an unsupervised artificial neural network, consisting of layers of neural starting with an input layer, proceeding through to an output layer. As SOMs are unsupervised, they possess self-learning qualities that use a genetic algorithm style approach to propagate layers that perform better than others do. In the summary of Merkl's paper, he concludes that this approach is very effective, in particular it is well suited to text classification problems. Being a neural network based system, however, it does need to be trained with a sample corpus to become proficient in that domain prior to being useful. Although neural networks are used for NLP and also for document classification, there are few applications using Neural Networks to extract fine information. There has been relatively little recent language research using symbolic learning (Freitag 1998). One reason is that IE tasks focus normally on extracting fine piece of information (such as name, price etc.) from documents. Some other techniques are more efficient and effective for these tasks. However, Neural Network is suitable for classification task for such documents with much noise.

7.3 Results Merging

Feature extraction outputs lists of extracted instances. As mentioned above, each extracting technique inserts the extracted instances in its own list. Because a basic profile contains not only underlying extracting techniques, but also definition of logical combination of such techniques, results processed by various techniques have to be combined with defined logical operation. Currently there are two logical operation available: “AND” and
“OR”. For “AND” operation, intersection of instances are retrieve, while for “OR” operation, union of instances are calculated.

After logical operations in a basic profile, updated results are stored in a hit list. Furthermore, a complex search profile consists of again many basic profiles. And, many complex profiles can be assigned to a template element. Hence, the results of extracting for one template element are produced by many different extracting methods. Because extracting algorithms work separately and sometimes autonomously, results can contain redundant information. Possible redundant results can be:

1. Duplication: Two results have the same position and length. In this case, one of instance has to be dropped.

2. Overlapping: Extracted areas are overlapped. In this case, the overlapped area is merged. The new result is added into the final list and the original two are then deleted.

3. Containment: One extracted area contains the other. Here only the bigger one is kept and the other is dropped (special cause of case 1)

4. Mixing redundant: complicated situation, e.g. both overlapping and containment. In this case, a new result is created that covers all overlapping results and also results with containment. Original results, which are identical to or only a part of the new results, are removed from the result list.

Figure 34 shows cases for merging results. The merging process runs as following: at first, an empty list is assigned as final list. Each list outputted by corresponding basic profile is to be added into the final list. For each list, instance is fetched as inserting instance. Such inserting instance is compared with the instance in final list. Algorithm tries to find the conflicts in two steps: at first, indexes for begin and end position of the inserting instance in the final list are retrieved. If the begin and end index is equal, it means there is no conflict and the inserting instance can be added into the final list simply. Otherwise, system tries to merge the extracting areas between the begin and end index. New instances are therefore created and added into the final list. Original instances, which are between the begin- and end-index, can now be safely removed from the final list.
7.4 Evaluation and discussion

In this section we evaluate some sample data with above introduced feature extracting techniques.

7.4.1 Introduction of evaluation method

The evaluation is processed as following:

1. For different document types, suitable sample data are gathered. Parts of the sample data are downloaded from well known bench-mark sample database from the internet. Such sample date is used, in order to make comparison with existing IE systems. Other sample data are collected from real life applications. Such sample in general are much more complicated than the first one. The sample is then disposed in two groups randomly: the one is the training set and the other is the testing set.
2. For each sample set, empty template is created by knowledge engineer. For bench-mark sample, same template is created as which in the existing IE systems.

3. The tester assigns then examples in training set to the empty template. If there are more than one example, tester has to duplicate the template instance and assign the examples to each instance.

4. Tester drives training. The system runs the training process and assign each trained extracting rules to the corresponding template element automatically. After training, the tester can save the trained template as Search Profile.

5. In the test phase, the tester loads at first a trained template together with its Search Profile

6. The tester drives IE-system to perform extracting from documents in testing set

7. After extracting, the tester evaluates the results and then calculates the precision and recall

8. The step 2 to 7 can be processed repeated with different number of examples.

9. After finishing evaluation, we can get also the training curves in behaviour with different number of training examples.

7.4.2 Evaluation

For feature extracting we tested our system with two document types: structured and semi-structured documents (see also Section 2.2.1). Structured documents are formatted so uniformly that a few examples are enough to learn perfect text extraction rules. Structured document is often found on Web sites that have been created automatically, for example from an underlying relational database or as a response to a search query. This class of document will be presented by CNN weather forecast web pages. The CNN Weather Forecast domain consists of Web sites for various domestic and international cities with four-day weather forecast. (See also Section 4.1).

The second type is semi-structured document. These have fairly stereotyped information but are not rigidly formatted. Experiments are reported from two domains of semi-structured document: the Rental Ads and Last Minute Holidays. The Rental Ads domain was collected from the seattle Times online-classified Ads. We downloaded the test set from Muslea's RISE Repository for
Information Extraction\(^5\). To make comparison we used the same template defined by WHISK (Soderland 1999). The template consists of three contents: information about neighbourhoods, about price and about bedrooms. The Last Minute Holidays Services is collected from the Web sites in lastminute.com (http://www.lastminute.com). To be extracted contents are names of offers, a range of price, a short description about the offer, information about award minutes per item, information about tickets, additional information to children and infants. Figure 35 and Figure 36 are the sample site for each domain.

<table>
<thead>
<tr>
<th>BALLARD AT LOCKS</th>
<th>Charming, security bldg, on bus-line, pool. Studio/1 BR $535-$625. 206 784-5797. 3025 NW Market St.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballard 2 BR/2 ba, w/d. 1500 sf, Penthouse, d/w. 7533 24 th Av NW. $1195 Wu Prop 206-522-8172 #371</td>
<td></td>
</tr>
<tr>
<td>(This ad is from 07/23/97 to 08/04/97.)</td>
<td></td>
</tr>
<tr>
<td>BALLARD - 1 Bedroom</td>
<td>new cpts, drapes &amp; paint, nr hospital &amp; bus. $535. 1519 NW 65 th, Apt #4 206-542-4600 (This ad last ran on 08/03/97.)</td>
</tr>
<tr>
<td>BALLARD - 2 br, frpl, deck, cov’d prkg, conv loc, no p/sm. $595. Call 206-783-4302.</td>
<td></td>
</tr>
<tr>
<td>(This ad last ran on 08/03/97.)</td>
<td></td>
</tr>
<tr>
<td>Ballard - 3 BR, 1150 sf., deck, fpl, cov pkg, WD, elevator. NS/NP $840. 300 NW 80 th, 206/782-1475</td>
<td></td>
</tr>
<tr>
<td>(This ad last ran on 08/03/97.)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 35: Sample Data of Rental Ads

\(^5\) http://www.isi.edu/~muslea/RISE/repositary.html
For CNN Weather Forecast, we used 10 examples for training and 100 examples for testing. As for WHISK (Soderland 1999) we used a training set of 400 examples and a test set of a further 400 examples. For Last Minute Holidays the training set has 40 examples, while the testing set has 100 samples. All these examples were made manually in a GUI tool.

Table 8 shows the results of the experiments. With two tests the system gets 100% recall and 100% precision for CNN Weather Forecast domain. In Rental Ads the recall and precision are improved with more examples. Recall begins at 88% from 50 examples and climbs to 95% when 400 examples. Precision starts at 87% from 50 examples and reaches to 98% when 400 examples. Compared with WHISK, For Last Minute Holidays the recall is 91% and precision is 98% with 20 examples. With 40 examples the recall is improved to 97%, but the precision goes down to 96%.

<table>
<thead>
<tr>
<th>Example Name</th>
<th>Examples</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN Weather</td>
<td>2</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>87%</td>
<td>88%</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>98%</td>
<td>95%</td>
</tr>
<tr>
<td>Rental Ads</td>
<td>20</td>
<td>98%</td>
<td>91%</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>96%</td>
<td>97%</td>
</tr>
</tbody>
</table>

Table 8 Experiments result
From the evaluation we will introduce some interesting points about three different test domains. Because the results are final results after template filling, we can also analyse some properties of NE processing. We found that in different situations some single NE-agents are especially important.

In CNN Weather Forecast, training with one example can perform perfect result already. The web pages are generated by robot automatically and the values of contents are merely some number or one word description (such as “high” etc.). Because the values of contents are exactly and unambiguously enclosed by tags (that is, the whole texts of XML nodes), Rule Extractor and Keyword Extractor become then unnecessary. The contents, such as current, maximal and minimal temperature etc., can be directly extracted by XSL-pattern, actually, after text zone recognising. Some elements, such as temperature for the coming four days, have the same XSL pattern. But they are ordered in a strict sequence. In this case, the TE-Agent plays a very important role, in disambiguate based on spatial constraints (see Section 9).

In Rental Ads the XSL patterns do not extract the underlying values exactly because all information is in one XML node., the XSL-patterns here are used really only for “text zone recognising”. Later processing can therefore not be ignored as in CNN weather report domain. Actually the XAL-pattern Text Zone detector, the Keywords Extractor and Rule Extractor in NE-Agent and also TE-Agent take on total extracting task here together. In this case the extracting is somewhat similar as Information Extraction from normal (not HTML-coded) documents. The XSL-pattern detector is used merely to distinguish multi-slots. After text zone detecting, for content “neighboured”, keywords extractor collects simply all neighboured names from examples. For contents “price” and “bedroom”, the rule extractor is used. The rule for content “price” is created almost perfectly after 5 examples as “$<\text{int}>$”. For content “bedroom”, however, is learned at first very crudely. Hence, the recall reaches already 99% when 50 examples, while the precision is only 33%. With 300 examples, the precision climbs to 88% and then stays in this level. From the evaluation we observed that the rule extractor improves rule quality by adding examples.

More interesting is the Last Minute Holidays domain. Some contents are extracted exactly after XSL pattern extracting already (such as range title and range price), because they have unique XSL patterns. Some contents can be distinguished after NE-Agent (such as short description and includes information) with different keywords. A few contents can be extracted exactly until the TE-Agent processing (such as additional information about children and infants), because they have same XSL patterns, same keywords but different spatial location (for example, additional information about children is always before infants). The extracting performance changes when more examples are used. The recall is improved considerably, while the precision loses slightly when more examples. After deep analysing of each single
element in template, we observed that the recall of elements about options of accommodations (such as TV, shower, mini-bar etc.) climbs while more examples, because more examples covers more options. On the other hand, the precision of elements “departure date” downgrades slightly with more examples. The reason is that the format of date varies: sometimes with time specification, sometimes not.

7.4.3 Discussion

From the evaluation we know that various NE-Agents are to be configured for different document types and also content types. Generally, this task can be made by knowledge engineer in the design phase. A knowledge engineer uses GUI tool to create different extracting rules, patterns, keywords etc. and stores them in Search Profile. At the second step, the knowledge engineer can develop empty template, which represents desired knowledge. For each element in a defined template, a definition of extracting rules, which is created and stored in search profile, is assigned. In the extracting phase, NE-scheduler passes out the task to underlying NE-agents according to the profile definition.

Another possibility to assign Search Profile to template is in the training phase. In this way, knowledge engineer does not need to create search profile manually. After developing a template, the knowledge engineer fills some examples into the template as training samples. Before training, the knowledge engineer has to specify the training algorithm (actually, the extracting methods) for each element. Then, the system drives various training algorithms, which are specified by knowledge engineer. After training, rules, patterns or keywords etc. are automatically created and stored in search profile.

An advanced assignment of Search Profile to template element in the training phase could be applied in the future. The NE-Scheduler could at first analyse the feature of examples and then decide to choose which training algorithm. A simple scenario could be that NE-Scheduler counts at first the number of words in the given example set. Is the number of words more than a defined threshold (e.g. 4), algorithm for weighted keywords is used, otherwise, NE-Scheduler selects the token-based rule extractor to do training. However, this is only a very simple sample. The selection could be much more complicated and therefore a complete selecting approach must be investigated.

As a step in this difference, below is given a discussion learning algorithms used in NE-Agents: XSL-Pattern, token based rule ,synonym collectors and weighted keyword extractor. The suitable situation for single training methods, the training quality, the number of required examples and the limitation of such learning algorithms are discussed here briefly.
Training algorithm for XSL-pattern uses a supervised bottom-up machine learning approach (see Section 7.1.2.2). The number of required examples depends on the quality of given examples. If given examples have strong evidence by evaluation, a few examples can bring already very good result (such as CNN Weather example). Normally, semi-structured or structured HTML documents have quasi-defined tree structure. Hence, the tree path defined by XSL-patterns can access the right place exactly. The time performance for extracting is strongly dependent on the complexity of input documents. Executing XSL-patterns by an XML parser could take from seconds to several minutes. If the HTML document is very large, for example more than 10M bytes, the extracting could slow down considerably in the practice.

In the practice, we observed that the attributes in nodes play sometimes for identifying right instance an very important role. For instance, an attribute for describing a cell width in table can be used to distinguish to other cells in the table. On the other hand, however, using attributes can cause the XSL-patterns too fine to use. Hence, overlapping XSL-patterns can be generated based on the path but on the attributes. That is, in the training phase, the overlapping patterns could have the same path but less attributes to the input patterns.

The synonym collector is simply implemented for gathering all different input examples. Extracting is processed as normal full text extracting (see Section 7.2.1). The processing can reach maximal speed and get very high precision and recall. However, one main barrier is that the given examples must be completely. That is, all possible value of the content must be given in examples, what is a big defiance in some real domain. However, an incremental learning strategy can enhance the usability of the synonym collector. That is, if user assigns in the future additional values for defined content, the synonym collector can add the new values in the thesaurus directly without the need for complete retraining.

The weighted keywords are designed for contents as parts of text. Learning algorithm is here a statistical word occurrence analyse. Hence, for statistical approach, more examples are required. Empirical study shows that in the most case the number of examples should be greater than 30. Compared to other techniques, the processing time is optimal. The time cost for learning is lineal and the time cost for extracting is similar as full text extracting but with additional weighting. However, the extracting performance is normally a little worse than full text extracting. The second difference to full text or synonym extracting is the precision and recall is not so good as the later two algorithms, because statistical approaches can not predict exactly. Hence, we can use the keyword approach to filter some significant wrong instances, which have the same Text Zones by applying XSL-patterns, without losing recall. Later
processing can enhance the precision. Another limitation is, that documents have to be pre-processed and segmented before processing.

Finally, algorithm for learning token based rule is also a supervised machine learning algorithm with EM (Expectation Maximization) approach (Dempster, Laird, & Rubin, 1977). Generally, better rules can be created when more examples. In the practice, for each single rule, at least 5 examples are needed. The leaning quality is dependent on the feature of contents. If the content has significant syntactic nature or very strong pre- and/or postcore strings, token-based rule can be created perfectly. Otherwise, token-based rule-learning generates rules with strong constraints and ignore weak features. That is, normally, token based rule guarantees high precision, but sometimes with poor recall. The processing time is average and comparable with the time using regular expressions. However, if the document is large, processing time can be extended considerably. Token based rules are used as widely as regular expressions in the system. In the next section we will introduce a hierarchical concept description and extraction approach, which is based on learning and extracting token based rule.

Table 9 concludes these four learning algorithms.

<table>
<thead>
<tr>
<th>XSL-Pattern</th>
<th>Collected Synonyms</th>
<th>Weighted Keywords</th>
<th>Token based rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used area</td>
<td>Structured or semi-structured HTML, for text zones recognition</td>
<td>Information, which can be defined by background knowledge</td>
<td>contents with strong syntactic feature and/or constraint</td>
</tr>
<tr>
<td>Machine Learning technique</td>
<td>Supervised, bottom-up learning</td>
<td>Simple collection</td>
<td>Supervised, EM approach</td>
</tr>
<tr>
<td>Required examples</td>
<td>Normally 10 to 20 each element</td>
<td>Required complete examples for all cases</td>
<td>At least 5 examples</td>
</tr>
<tr>
<td>Performance</td>
<td>Good precision and recall</td>
<td>Very good precision and recall when examples are complete</td>
<td>Dependent on the contents. Normally worse than exact full text extraction</td>
</tr>
<tr>
<td>Extracting speed</td>
<td>Slow down when document is large</td>
<td>Similar as full text extracting, very fast</td>
<td>High precision, with strong rule sometimes poor recall</td>
</tr>
<tr>
<td>Limitation</td>
<td>Pre-processing, Converting from HTML to XML required</td>
<td>Difficult to provide complete examples</td>
<td>No strong limitation</td>
</tr>
</tbody>
</table>

Table 9 Conclusion of learning algorithms in NE-agents