Information Engineering Group

Information Extraction

A Survey

Katharina Kaiser and Silvia Miksch

Vienna University of Technology
Institute of Software Technology & Interactive Systems

Asgaard-TR-2005-6 May 2005

Authors: **Katharina Kaiser and Silvia Miksch**

{kaiser, silvia}@asgaard.tuwien.ac.at
http://ieg.ifs.tuwien.ac.at

Contact: **Vienna University of Technology**
Institute of Software Technology & Interactive Systems

Favoritenstraße 9-11/188
A-1040 Vienna
Austria, Europe

Telephone: +43 1 58801 18839
Telefax: +43 1 58801 18899
Web http://ieg.ifs.tuwien.ac.at

Contents

Abstract

Information Extraction is a technique used to detect relevant information in larger documents and present it in a structured format. Information Extraction is not Text Understanding. It is used to analyze the text and locate specific pieces of information in the text.

Information Extraction techniques can be applied to structured, semi-structured, and unstructured texts. For the latter one, Natural Language Processing is necessary which is implemented in traditional Information Extraction systems. To process structured and semi-structured texts often no NLP techniques are necessary as they do not offer such a rich grammatical structure. For this reason, so called wrappers are developed that incorporate the different structures of documents.

In this paper we will describe the requirements and components of Information Extraction systems as well as present various approaches for building such systems. We then will represent important methodologies and systems for both traditional Information Extraction systems and wrapper generation systems.

Chapter 1

Introduction

Unstructured data, most of it in the form of text files, typically accounts for over 80 % of an organization's knowledge stores, but it is not always easy to find, access, analyze, or use. Therefore, we need tools and methods to locate the information and synthesize it into knowledge in order to make it useful. A common approach to accomplish this task is Text Mining.

Text Mining (TM) is about looking for patterns in natural language text and has been defined as "the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources" [16]. It recognizes that complete understanding of natural language text is not attainable and focuses on extracting a small amount of information from text with high reliability. TM uses recall and precision scores to measure the effectiveness of different Information Extraction techniques to allow quantitative comparisons to be made.

Information Extraction (IE) is one of the most prominent techniques currently used in TM. It is a starting point to analyze unstructured text. In particular, by combining Natural Language Processing tools, lexical resources, and semantic constraints, it can provide effective modules for mining documents of various domains.

IE is an emerging NLP technology, whose function is to process unstructured, natural language text, to locate specific pieces of information, or facts, in the text, and to use these facts to fill a database. Peshkin and Pfeffer [31] define IE as the task of filling template information from previously unseen text which belongs to a pre-defined domain.

Its goal is to extract from documents salient facts about pre-specified types of events, entities, or relationships. These facts are then usually entered automatically into a database, which may then be used for further processing.

Today, IE systems are commonly based on pattern matching. Each pattern consists of a regular expression and an associated mapping from syntactic to logical form.

Chapter 2

Information Extraction Systems: Aspects and Characteristics

2.1 Design Approaches

Numerous IE systems have been designed and implemented. In principal, the used approaches can be categorized into two groups: (1) the learning approach, and (2) the Knowledge Engineering approach.

For systems or modules using *learning techniques* an annotated corpus of domain-relevant texts is necessary. Therefore, there is no need for system expertise. This approach calls only for someone who has enough knowledge about the domain and the tasks of the system to annotate the texts appropriately. The annotated texts are the input of the system or module, which runs a training algorithm on them. Thus, the system obtains knowledge from the annotated texts and can use it to gain desired information from new texts of the same domain.

The *Knowledge Engineering (KE)* approach asks for a system developer, who is familiar with both the requirements of the application domain and the function of the designed IE system. The developer is concerned with the definition of rules used to extract the relevant information. Therefore, a corpus of domain-relevant texts will be available for this task. Furthermore, she or he is free to apply any general knowledge or intuitions in the design of rules. Thus, the performance of the IE system depends on the skill of the knowledge engineer.

The KE approach uses an iterative process, whereas within each iteration the rules are modified as a result of the system's output on a training corpus. Thus, the KE approach demands a lot of effort.

2.1.1 Pattern Discovery

Pattern Discovery is a major part of IE. It is the task of identifying the extraction patterns that are relevant for the user's need, specified by the user's query. Patterns can be discovered automatically, semi-automatically, and manually.

At the beginning of developing IE systems the systems were customized manually to a given task [4, 17, 21]. Due to the very high costs for computational linguists approaches were demanded to automate this task. Thus, within the Proteus project [40] a semi-automatic creation of patterns was developed. Thereby, a discovery procedure of extraction patterns aims to help the user create the knowledge-bases, such as lexicons and

patterns. The methodology includes showing a pattern candidate and incorporating an existing ontology and pattern set.

The automatic approach takes a set of documents and outputs a set of extraction patterns by using Machine Learning techniques. Automatic learning systems can be categorized in three groups: (1) supervised learning systems [32, 37, 20, 19, 36], (2) semi-supervised learning systems [35, 1, 41], and (3) unsupervised learning systems [34, 11, 38].

2.1.2 Considerations for an Approach

Appelt [2] defined a number of considerations that influence the decision to utilize a particular approach for a particular module (see Section 2.2: Components of an Information Extraction System):

1. **The availability of training data.** If the required training data is available or cheaply and easily obtainable, the learning approach should be chosen. For complex domain-level tasks, where the annotation task is much slower, more difficult, more expensive, or requires extensive domain expertise from annotators, the KE approach may be favored.

2. **The availability of linguistic resources.** If linguistic resources like lexicons and dictionaries are available developing rules by a knowledge engineer may be possible. Otherwise, it may be necessary to rely on training from the annotated corpus.

3. **The availability of knowledge engineers.** If there is no skilled knowledge engineer, the learning approach should be chosen.

4. **The stability of the final specifications.** If specifications change, it is often easier to make minor changes to a set of rules than to reannotate and retrain the system. However, other changes in specifications may be easier to accomplish with a trainable system.

5. **The level of performance required.** Human skills count for a lot. The best performing systems for various IE tasks have been hand crafted. The performance of automatically trained systems depend on the quantity of available training data. Evaluations at MUCs show that with enough data the automatically training approach can achieve equivalent results to the Knowledge Engineering approach.

2.2 Components of an Information Extraction System

A typical Information Extraction system has phases for input tokenization, lexical and morphological processing, some basic syntactic analysis, and identifying the information being sought in the particular application (cf. Figure 2.1) [2]. Depending on what one is interested in some phases may not be necessary. In addition to the modules in the left-hand column, IE systems may include modules from the right-hand column, depending on the particular requirements of the application.

2.2.1 Tokenization

The Tokenization module is responsible for splitting the input text into sentences and tokens. Tokenization is a trivial problem for European languages with clear word borders. However, in processing some languages like Chinese or Japanese, it is not evident from the

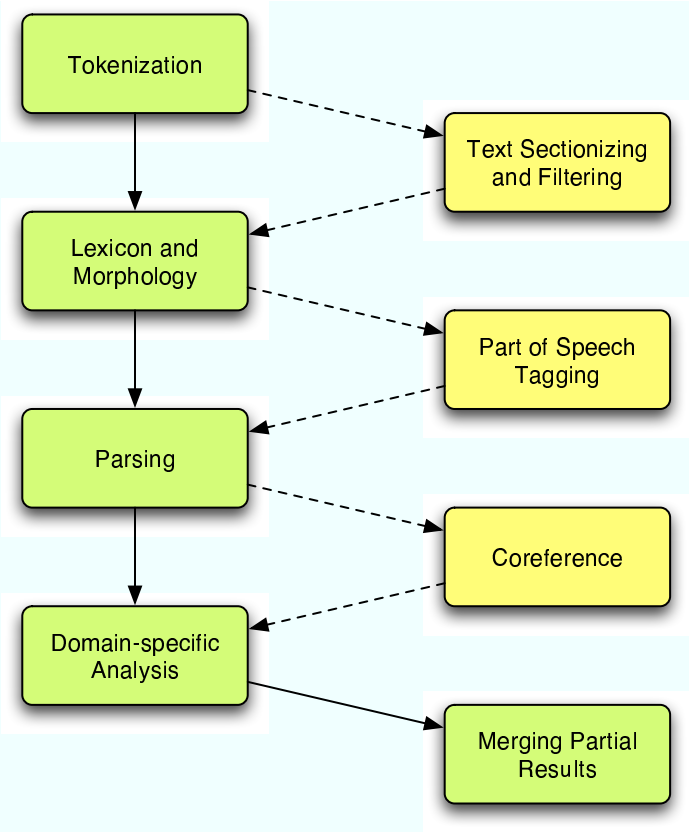

Figure 2.1: Modules of an Information Extraction System.

orthography where the word boundaries are. Therefore, systems for these languages must necessarily be expanded by a Word Segmentation module.

2.2.2 Lexical and Morphological Processing

Morphology is a sub discipline of linguistics and is interested in the structure of word forms. Many IE systems for languages with simple inflectional morphology, like English, do not have a morphological analysis component at all. In English, it is easy to simply list all inflectional variants of a word explicitly in the lexicon. For languages like French, with more complex inflectional morphology, a morphological analysis component makes more sense, but for a language like German, where compound nouns are agglutinated into a single word, morphological analysis is essential.

In the lexical analysis the tokens determined by the Tokenization module are looked up in the dictionary to determine their possible parts-of-speech and other lexical features that are required for subsequent processing. The most important job of this module is the handling of proper names. Recognizing names can thereby be done with either handcrafted rules under the Knowledge Engineering approach or automatically trained rules derived from an annotated corpus.

In addition to name recognition, this module must assign lexical features to words required for subsequent processing. This can be accomplished by either a lookup in a lexicon, or by automatic taggers, like a parts-of-speech tagger. A parts of speech tagger annotates each word of a sentence with its parts of speech tag, such as noun, verb, adjective, and so on. It can avoid incorrect analysis based on disambiguation caused by rare word senses in comprehensive lexicons.

2.2.3 Parsing

Syntactic Analysis has the aim to identify the syntactic structure of the analyzed document. Most IE systems only accomplish a shallow, fragment analysis of the texts. But there have been even IE systems which totally skip the phase of syntactic analysis.

IE systems are only interested in specific types of information in a text and ignore portions of text, which are not relevant for their task. Therefore, parsing these portions and finding irrelevant grammatical relationships will be unnecessary.

2.2.4 Coreference

The reason for this module is simply that application relevant entities will be referred to in many different ways throughout a given text and thus, success on the IE task was, to a least some extent, conditional on success at determining when one noun phrase referred to the very same entity as another noun phrase. Thus, a Coreference module should handle the following coreference problems:

- *Name-alias coreference.* Names and their common variants must be recognized as coreferring, e.g., 'Silvia Miksch' and 'Prof. Miksch'.
- *Pronoun-antecedent coreference.* Pronouns like 'she', 'he', 'they', and so on must be associated with their antecedents, resolving them to a domain relevant named entity if possible.
- *Definite description coreference.* Handling this problem requires arbitrary world knowledge to make the connection among descriptors. When building an IE system, it is reasonable to include ontological information for domain-relevant entities

that enable such resolution in restricted cases, but doing it in general is unrealistic. Examples are 'Apple Computer' and 'the company' or 'Apple' and 'the Cupertino computer manufacturer'.

As with other IE modules, it is possible to build Coreference modules by both knowledge engineering and automatic learning approach. The knowledge engineering approach is, for instance, applied by FASTUS [3]. For accomplishing the coreference task with the learning approach Decision Trees can be applied [27].

2.2.5 Domain-specific Analysis

The domain analysis is the core of most IE systems. The preceding modules prepare the text for the domain analysis by adding semantic and syntactic features to it.

This module fills the templates, which are in general constructed as attribute-value pairs. Templates consist of a collection of slots (i.e., attributes), each of which may be filled by one or more values. These values can consist of the original text, one or more of a finite set of predefined alternatives, or pointers to other template objects. Typically, slot fillers representing dates, times, job titles, and so on are standardized by normalization rules.

For extracting facts and events, the system needs domain specific extraction patterns (i.e., extraction rules or case frames). These patterns can be generated manually (by means of Knowledge Engineering) or automatically (by means of automatic learning). The portion of text that matches a defined linguistic pattern is memorized and the information is extracted by the guidance of the extraction rule from this portion of text to fill the template.

For designing the domain-relevant pattern rules there exist two approaches, which can be characterized as (1) the *atomic approach* and (2) the *molecular approach*.

The *molecular approach* is the most common one. It involves matching all or most of the arguments in an event in a single pattern. The development cycle of this approach starts with a small number of highly reliable rules that capture the common core cases of the domain, but ignoring a broad class of less frequently occurring relevant patterns. Further development is characterized by capturing ever-larger numbers of increasingly rare and problematic cases with increasingly general and possibly overgenerating rules. Thus, the system starts with high precision and low recall scores and evolves in increasing recall and progressively lower precision.

The *atomic approach* builds a domain module that recognizes the arguments to an event and combines them into template structures strictly on the basis of intelligent guesses rather than syntactic relationships. The development cycle can be characterized by assuming domain-relevant events for any recognized entities, leading to high recall, but much overgeneration, and thus low precision. Further development would result improving filters and heuristics for combining the atomic elements, improving precision

The atomic approach make sense at tasks characterized by the following features: (1) entities in the domain have easily determined types and (2) the templates are structured so that there is only one or a very small number of possible slots that an entity of a given type can fill and only entities of a given type can fill those slots.

2.2.6 Merging Partial Results

Usually, the sought-after information is spread among different sentences. In these cases information should be combined before creating the final templates. For this purpose some IE systems use a Merging module. This module uses an algorithm to decide which templates can be merged.

Again, there are two merging strategies: (1) the Knowledge Engineering approach, where one performs data analyses to identify good merging principles, defines rules, and tests the results; and (2) using some training mechanism to acquire the merging strategies automatically by the system.

2.3 Types of Data

We can differentiate the various IE systems by the type of data that are used as origin: structured data, semi-structured data, and unstructured data.

1. **Structured Data:**
   This is relational data (e.g., from databases). By means of the structure of the database a meaning of the particular data is assigned. Thus, detecting relevant information and the assignment of a meaning can be eased.

2. **Semi-structured Data:**
   No semantic is applied to these data, but for extracting the relevant information no Natural Language Understanding (NLU), like analysis of words or sentences, is required. Examples are advertisements in newspapers and job postings or highly structured HTML pages.

   According to [5, 26] both XML-encoded and HTML-encoded pages contain semi-structured data, but HTML is rather more 'human-oriented' or 'presentation-oriented'. It lacks the separation of data structure from layout, which XML provides.

3. **Unstructured Data:**
   This can be, for example, plain text. For extracting the relevant information we must understand the text. Therefore, methods of NLU are applied. Examples are newspaper articles.

2.4 Evaluating Information Extraction Systems

Over the course of several Message Understanding Conferences (MUCs) organizers and participants agreed how IE systems should be evaluated [25]. Thereby, the extracted output is presented as hierarchical attribute-value structures. Human annotators provide a set of key templates for the training data and the test data that is compared to the system's output. Values that correctly match the key values are counted as *correct*, whereas values that do not match are *incorrect*. Attributes with non-empty values not aligning with a key attribute are considered *overgeneration*. Thus, it is possible to define recall and precision scores for the output of an IE system given the total possible correct responses (P), number of correct values (C), number of incorrect values (I), and overgenerated values (O) as follows:

$$\textit{recall} = \frac{C}{P}$$

$$\textit{precision} = \frac{C}{C + I + O}$$

Thereby, recall measures the ratio of correct information extracted from the texts against all the available information present in the text. Precision measures the ratio of correct information that was extracted against all the information that was extracted. It is difficult to optimize both parameters at the same time. Is a system optimized for high precision the

feasibility of not detecting all relevant information improves. But if recall is optimized it is possible that the system classifies irrelevant information as relevant.

Besides, there can be defined the *F measure* to have a metric that can be used to compare various IE systems by only one value. This metric uses weighted recall and precision scores depending which value is given more importance.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

The *F* measure is a geometric mean between recall and precision. By means of the parameter β it can be determined whether the recall (R) or the precision (P) score is weighted more heavily.

Recall, precision, and *F* measure are the most frequently used metrics when referring to an IE system's performance.

2.5 Wrapper

A wrapper is a program carrying out retrieving information from different repositories, merging, and unifying them. But often wrappers are only applied to the latter two activities. The aim of a wrapper is to locate relevant information in semi-structured data and to put it into a self-described representation for further processing [24]. It seems, as if IE systems and wrappers do just the same, but the application areas are different. Besides, many information resources do not exhibit the rich grammatical structure NLP techniques applied in IE systems are designed to exploit.

The most widespread application area of wrappers is the World Wide Web with its unlimited amount of web sites that are mostly semi-structured. The differences between the structure of each document in the web and the fact that sites are changed periodically makes it obvious that building such programs by hand is not a feasible task. This facts lead to two main problems in this field: ”wrapper generation” and ”wrapper maintenance” [9].

Rule-based methods have been especially popular in recent years. Some techniques for generating rules in the realm of text extraction are called ”wrapper induction” methods. These techniques have proved to be rather successful for IE tasks in their intended domains, which are collections of documents such as web pages generated from a template script [29, 23, 26, 5]. However, wrapper induction methods do only extend well to documents specific to the induced rules.

Concerning the result pages in HTML format we have to consider the following aspects that are important for the wrapper generation [10]: format uniqueness and completeness as well as the HTML-quality level.

2.5.1 Format Uniqueness and Completeness

It provides the most obvious classification of html-pages returned by the different backends. Completeness is hereby defined to be homogeneity, that is, a search request always returns the same set of elements within the resulting html-page. The following classification can be given:

- **Rigorous structure**, i.e., unique format and complete information: for example, AltaVista always returns a ”name” and a ”body” element (even though with rather unstructured, raw information).

- **Semi-rigorous structure**, i.e., unique format and incomplete information: for example, Library of Congress always provides a list of attributes for each returned item but not all of the attributes are common for all the items.
- **Semi-relaxed structure**, i.e., no unique format and complete information: semi-relaxed structures result from data sets put together from different legacy collections.
- **Relaxed structure**, i.e., no unique format and incomplete information: most home pages on the Web have relaxed structures.

2.5.2 HTML-Quality Level

Another aspect that has to be mentioned is the HTML-quality level. The classification ranges from high level to low level. In existing information repositories, almost all variation between these two extreme classifications can be found.

- **High level:**
  Each item in the result page is surrounded by a couple of html-tags, such as –. Each of these ”tagged” items corresponds to exactly one attribute of the original data (e.g., each skill requirement is stated within a single tag).
- **Low level:**
  A string between two html-tags corresponds to more than one output attributes. In this case, some additional plain-text separators like ”.”, ”,”, ”;” are used for separating the different attributes of the original data (e.g., a set of requirements for a job position is stated within one tag). Here, the analysis of the html-structure of the result pages is not enough, and a plain text analysis must be done!

2.5.3 Wrapper Generation

For generating wrappers similar methods can be applied like for IE systems: (1) manual wrapper generation based on specialized languages to express extraction rules written by a human expert, (2) semi-automatic wrapper generation using Machine Learning approaches [24, 5, 30, 26], and (3) automatic wrapper generation applying unsupervised learning [15, 8, 13].

Chapter 3

Prior Work

This chapter introduces prior work on both manual and automatic learning of extraction patterns for IE systems and wrappers.

3.1 Manual Pattern Discovery in IE Systems

In earlier MUCs[1], many IE systems were customized manually to a given task.

FASTUS

FASTUS [17] is a (slightly permuted) acronym for Finite State Automaton Text Understanding System, although it does not deal with text understanding but IE. It is a cascaded, nondeterministic finite state automaton.

FASTUS, as used for MUC-4, is based on a 4-step processing: (1) triggering, (2) recognizing phrases, (3) recognizing patterns, and (4) merging of incidents. In the first pass, trigger words are searched for in every sentence. Additionally, person's names identified in previous sentences are also treated as trigger words for the remainder of the text. In the second step, noun groups, verb groups, and several critical word classes (e.g., prepositions, conjunctions, relative pronouns) are recognized by a nondeterministic finite state automaton. In this phase no parts-of-speech tagger is necessary as it was shown to not improve the results, but decreases the system's performance. In the third step, patterns are recognized. Pattern generation was done completely hand-coded. For MUC-4 95 patterns were implemented to extract incidents detected in the documents. In the last step incidents are merged. The application area of MUC-4 have been news articles about terrorist activities. Merging is accomplished for incidents of the same sentence and for incidents remaining at the end of the processing which are merged with incidents found in previous sentences. In case of incompatible incident's types, dates, or locations, merging is blocked.

The basic system is relatively small, although the dictionary used is very large. The manually developed rules were very effective and performed very well. Due to the fast run time and the simple concepts the development time was also very fast.

GE NLTOOLSET

The GE NLTOOLSET [21] is an IE system using a knowledge-based, domain-independent core of text processing tool. The processing of the toolset is divided into three stages: (1) pre-processing, (2) linguistic analysis, and (3) post-processing.

[1] Message Understanding Conferences

In the pre-processing phase the text is segmented and irrelevant parts are filtered out, phrases that are template activators are identified, and portions of text are marked that could describe discrete events. In the analysis phase parsing and semantic interpretation is performed. The post-processing module selects the templates and maps semantic categories and roles into those templates.

The system's knowledge base consists of a core sense-based lexicon and a feature and function grammar. The core lexicon contains over 10,000 entries, of which a small set is restricted to the application domain. The core grammar consists of 170 rules, again with a small set of MUC-specific rules.

PLUM

PLUM (Probabilistic Language Understanding Model) [4] as used in MUC-4 applied a manually generated rules. The system architecture contains a preprocessing module, a morphological analysis module, a parsing module, a semantic interpreter, a discourse processing module, and a template generation module.

Within the *preprocessing module* message boundaries were determined, the header is identified, and paragraph and sentence boundaries are determined. The *morphologic analysis module* assigns parts-of-speech (POS) information, whereby the POS tagging is augmented by (automatically trained) probabilistic models for recognizing words of Spanish and English origin. The *parsing module* generates one or more non-overlapping parse fragments spanning the input sentence. These fragments are then processed by the *semantic interpreter*. This module uses semantic components, such as lexical semantics and semantic rules. Lexical semantics are constructed by an automatic case frame induction procedure. They indicate the word's semantic type and predicates pertaining to it. Semantic rules are general syntactic patterns. Their basic elements are 'semantic forms', which can be either entities of the domain, events, or states of affairs. Entities correspond to people, places, things, and time intervals of the domain and arise from noun phrases. Events (who did what to whom) and states of affairs (properties of entities) may be described in clauses. The *discourse module* constructs event objects corresponding to relevant events in the message. Thereby, it must infer indirect relations not explicitly found by the interpreter and resolve any references in the text. The *template generator* then uses the structures created by the discourse module to generate the final templates.

PROTEUS

PROTEUS [40] is a core extraction engine consisting of seven modules (see Figure 3.1): (1) lexical analysis, (2) name recognition, (3) partial syntactical analysis, (4) scenario pattern analysis, (5) reference resolution, (6) discourse analysis, and (7) output generation.

The *lexical analysis* module splits the document into sentences and tokens. Each token is assigned a *reading* using dictionaries. Optionally, a parts-of-speech tagger can be invoked to eliminate unlikely readings from tokens. The *name recognition*, *partial syntax*, and *scenario patterns* modules use deterministic, bottom-up, partial parsing, or pattern matching. Patterns are regular expressions. Patterns for *name recognition* identify proper names. The *partial syntax* module finds noun phrases and verb phrases. The *scenario patterns* module finds higher-level syntactic constructions. The *reference resolution* module links anaphoric pronouns o their antecedents and merges other co-referring expressions. The *discourse analysis* module builds more complex event structures using higher-level inference rules. Thereby, several clauses contain information about a single complex fact. The

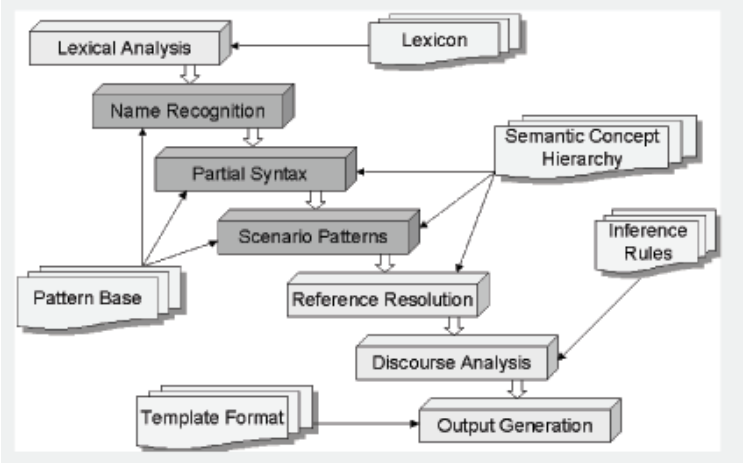

Figure 3.1: Proteus system architecture.

template generation module performs a transformation of the gathered information into the final template structure.

The pattern acquisition consists of several steps. First, the user enters a sentence containing an event and selects an event template from a list of events. Then, the system applies applies current patterns to the example to obtain an initial analysis. Thereby, it identifies noun/verb groups and their semantic types and applies a minimal generalization. The system presents the result to the user, who can modify each pattern element (e.g., choose the appropriate level of generalization, make the element optional, remove it). The user has then to specify how pattern elements are used to fill slots in the event template. Now the system builds a new pattern to match the example and compiles the associated action, which will fire when the pattern matches and will fill the slots of the event template. The new pattern is added to the pattern base.

3.2 Automatic Pattern Discovery in IE systems

Due to the cumbersome and time-consuming manual generation of extraction rules accomplished by knowledge engineers, research has been directed towards automating this task. Therefore, two approaches can be applied: (1) supervised learning, where a large set of training data is required to learn the rules using Machine Learning techniques, and (2) unsupervised learning, where rules are learned by a small set of seed rules and an annotated corpus using bootstrapping methods.

3.2.1 Supervised

Supervised learning uses training data to induce extraction rules. Thereby, almost no knowledge about the domain is necessary, but a large set of training data has to be annotated according to the underlying structure of information to be extracted.

AutoSlog

AutoSlog [32] was the first system to learn text extraction rules from training examples. It extracts a domain-specific dictionary of concept nodes for extracting information from

Table 3.1: Sample of AutoSlog heuristics.

Linguistic Pattern
<subject>passive-verb
<subject>active-verb
<subject>verb infinitive
<subject>auxiliary noun
passive-verb <direct-object>
active-verb <direct-object>
infinitive <direct-object>
...

text. A concept node is a rules which includes a ”trigger” word or words and a semantic constraint. If the system finds the trigger in the text and the concept node's conditions are satisfied, the concept node is activated and the concept node definition is extracted from the context.

The system identifies a sentence annotated with a slot filler and semantic tag. Then, it looks up its list of heuristics (for samples see Table 3.1) and sees if any of the heuristics match the part of the sentence containing the slot filler.

Each heuristic handles only a single-slot extraction. AutoSlog also uses a semantic tagger and semantic constraints in the extraction rule; it does not merge similar concept nodes and handles only free text.

PALKA

The PALKA system [20] uses an induction method similar to Mitchell's candidate elimination algorithm [28]. It produces the extraction rule as a pair of a meaning frame and a phrasal pattern, called *Frame-Phrasal pattern structure (FP-structure)*. If existing rules cannot be applied, PALKA creates a new rule and tries to generalize it with existing ones to include a new positive instance. It specializes existing rules to avoid a negative instance if it generates the wrong interpretation when applying existing rules.

CRYSTAL

The CRYSTAL system [37] takes texts, which are processed by a syntactic parser. It needs training documents annotated by a domain expert, as well as a semantic hierarchy. CRYSTAL starts learning by creating the extraction rules for each instance of the target event in the training data. Using inductive learning it finds the most similar pair or rules and merges them together by finding the most restrictive constraints that cover both rules.

LIEP

The LIEP system [19] uses heuristics in a manner similar to AutoSlog, but learns multi-slot rules and cannot handle single slot extraction. It allows a user to interactively identify

events in text, based on the assumption that a large annotated training corpus is hard to obtain. For each potential training sentence, entities of interest (e.g., people, companies, etc.) are identified.

LIEP tries to choose extraction patterns that will maximize the number of extractions of positive examples and minimize spurious extractions. If a new example cannot be matched by a known pattern, LIEP attempts to generalize a known pattern to cover the example. If the generalization is not possible or the resulting pattern decreases the quality, a new pattern is constructed.

WHISK

The WHISK system [36] uses Machine Learning algorithms known as covering algorithms.

WHISK uses a covering algorithm to induce regular expressions in a top-down induction. It begins with the most general rule and continues by progressively specializing the available rules. The process stops when a set of rules is generated covering all positive training examples. Afterwards, post-pruning is achieved to remove rules that generate overfitting.

RAPIER

The RAPIER[2] system [7] uses pairs of sample documents and filled templates to induce pattern-match rules that directly extract fillers for the slots of the template. It employes a bottom-up learning algorithm in order to limit search without imposing artificial limits on the constants to be considered, and in order to prefer high precision by preferring more specific rules. Pairs of rules are randomly chosen and a beam search is achieved to find the best generalization of the two rules, taking a least general generalization (LGG), then adding constraints until the proposed rule operates correctly on the training data. RAPIER can only handle single-slot extraction on semi-structured text.

GATE

GATE [14] is a framework and graphical development environment which enables users to develop and deploy language engineering components and resources. A set of reusable processing resources is packed together to ANNIE, A Nearly-New IE system. These processing resources can be used individually or coupled together with new modules.

ANNIE consists of a tokeniser, a sentence splitter, a POS tagger, a gazetteer, a finite state transducer, an orthomatcher, and a coreferencer. The *tokeniser* splits text into tokens (i.e., numbers, punctuation, symbols, and words of different types). The *sentence splitter* segments the text into sentences, which are the input of the *tagger*. It produces a parts-of-speech tag as an annotation on each word or symbol. The *gazetteer* consists of lists of cities, organizations, days of the week, and so on. The *semantic tagger* consists of handcrafted rules, which describe patterns to match and annotations to be created as a result. The *orthomatcher* performs co-reference, or entity tracking, by recognizing relations between entities. The *coreferencer* finds identity relations between entities in the text.

GATE provides easy-to-use and extendable facilities for text annotation to annotate required training data for NLP algorithms. The annotation can be done manually by the user or semi-automatically by running some processing resources over the corpus and then

[2]Robust Automated Production of Information Extraction Rules

correcting and adding new annotations manually. Depending on the information to be annotated, some ANNIE modules can be used or adapted to bootstrap the corpus annotation task.

GATE addresses the complete range of issues in NLP application development in a flexible and extensible way. It promotes robustness, re-usability, and scalability.

Discussion

The main bottleneck of supervised IE systems is the preparation of the training data. Most systems need a large amount of annotated documents for a particular extraction task, which also leads to the lack of portability of an IE system.

3.2.2 Unsupervised

Unsupervised Learning systems reduce the burden of the user to require only a statement of the required information. No extraction patterns are given in advance by the user. The main challenge is to realize the user's need into a set of the extraction patterns. The systems are based on bootstrapping methods, expanding an initial small set of extraction patterns.

AutoSlog-TS

AutoSlog-TS [33] is an extension of AutoSlog [32]. AutoSlog-TS (cp. Figure 3.2) needs only a corpus pre-classified with respect to each document's relevance to the task of interest. It generates extraction patterns for every noun phrase in the training corpus by means of heuristics. A major extension to AutoSlog is that it allows multiple rules to fire if more than one matches the context and thus multiple extraction patterns may be generated. Statistics will reveal which pattern is needed to be reliable for the domain. In a second stage these patterns are evaluated. Thereby, it processes the corpus a second time and generates relevance statistics for each pattern by which the extraction patterns are ranked.

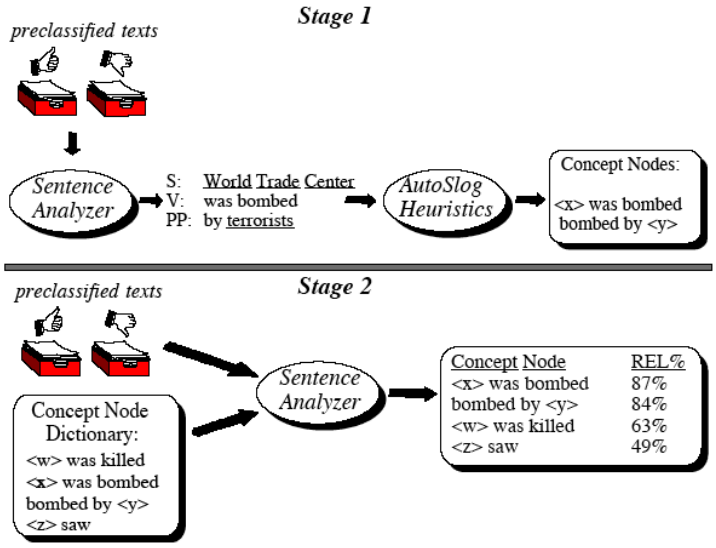

Figure 3.2: AutoSlog-TS flowchart.

Mutual Bootstrapping

Riloff and Jones [35] propose a co-training algorithm using mutual bootstrapping for lexical discovery. Lexicons and extraction patterns are thereby used as separate features. Mutual bootstrapping is used due to the assumption that a good pattern can find a good lexicon and a good lexicon can find a good pattern.

Given a handful of lexical entries as initial data, patterns are discovered that extract the initial lexicon. The extracted patterns are ranked and the most reliable are used to extract more lexical items.

A strong limitation of mutual bootstrapping is that a minor error can cause a large amount of errors during the following iteration. A touching up was introduced by *meta-bootstrapping*. Thereby, each iteration takes only the five best noun phrases for adding to the extracted lexicons.

EXDISCO

EXDISCO [41] also applies a mutual bootstrapping strategy. It is based on the assumption that the presence of relevant documents indicates good patterns and good patterns can find relevant documents.

Given an unannotated corpus and a handful of seed patterns, the document set is divided into a *relevant document set* containing at least one instance of patterns and a *non-relevant document set* not containing any seed patterns. Now candidate patterns are generated from the clauses in the documents and ranked in correlation with the relevant documents. The highest pattern is added to the pattern set and each document is re-ranked using the newly obtained pattern set. Again, the entire document set is split into *relevant* and *non-relevant* and the system keeps iterating.

Snowball

The Snowball [1] system is based on Dual Iterative Pattern Expansion (DIPRE) algorithm [6]. DIPRE is similar to co-training and works well on data with two distinct features, each of which can independently distinguish the class of instances from the other.

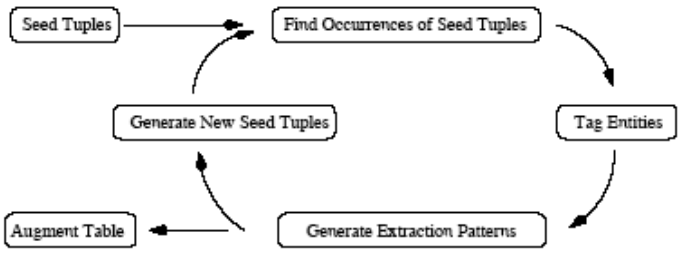

Figure 3.3: Main components of Snowball.

Figure 3.3 shows the main components of the Snowball system. Given a handful of initial relation instances and a general regular expression that the entities must match, Snowball generates patterns from text documents. A key improvement from DIPRE is that Snowball's patterns include named-entity tags (e.g., <LOCATION>-based <ORGANIZATION> instead of <STRING1>-based <STRING2>). Snowball patterns are generated by clustering similar tuples using a simple single-pass clustering algorithm.

After generating patterns, the system discovers new tuples that match the patterns in a certain degree. Each candidate tuple will then have a number of patterns that helped generate it associated with a degree of match. This information helps Snowball to decide what candidate tuples to add to the final template.

QDIE

The Query-Driven Information Extraction (QDIE) framework [38] tries to minimize human intervention by using a set of keywords as input. It parses the documents by a dependency parser and a Named Entity tagger and retrieves relevant documents specified by the user's query. Dependency trees of the sentences are used for pattern extraction. Each dependency subtree of a sentence that confirms to the pattern model becomes a *pattern candidate*. QDIE calculates the relevance score for each pattern candidate using tf/idf scoring in IR literature. A pattern is more relevant the more it appears in the relevant document set and less across the entire document set.

QDIE uses the Subtree model, a generalization of the Predicate-Argument model [41] and the Chain model [39], such that any subtree of a dependency tree in a source sentence can be regarded as an extraction pattern candidate.

Discussion

The development of unsupervised IE systems arose from the high costs for annotating training documents for supervised learning. AutoSlog-TS [33] only takes a corpus pre-classified for relevancy as initial input, as the classification of documents is a far easier task than annotating a large amount of training data.

A drawback of bootstrapping-based systems is that a minor error can cause a large amount of errors during iteration, often caused by polysemous words and phrases. To reduce the degree of polysemous patterns an named-entity constraint can be implemented, which will increase the amount of pattern.

3.3 Semi-automatic Wrapper Generation

In semi-automatic wrapper generation Machine Learning approaches are applied. Tools may support the design of the wrapper. Some approaches offer a declarative interface where the user shows the system what information to extract.

WIEN

WIEN (Wrapper Induction ENvironment) [24, 22] is designed for automatically learning of Web pages and is strongly influenced by ShopBot. It works on structured text containing tabular information.

WIEN looks for uniform delimiters that identify the beginning and end of each slot and for delimiters separating the tabular information from the surrounding text.

In order to automate wrapper construction as much as possible and thereby avoiding to manually labeling training data, a set of techniques for automatic labeling has been developed. The labeling algorithm takes a set of heuristics for recognizing instances of attributes to be extracted as input, whereas it is not concerned in the way they are obtained.

Wrapper induction uses a bottom-up induction algorithm, which takes a set of labeled pages as input. WIEN uses only delimiters immediately preceding and following the data

to be extracted. It cannot wrap sources in which some items are missing or sources where items may appear in a varying order.

SoftMealy

SoftMealy [18] is based on non-deterministic finite state automata (NDFA). It uses a bottom-up inductive learning algorithm to produce extraction rules using labeled training pages. It can deal with missing values and is able to handle permutations of values if the training set includes all possible permutations.

STALKER

STALKER [30] is a hierarchical wrapper induction algorithm. It transforms complex documents into a series of simpler extraction tasks handling both missing data and permutations of values.

Web documents are described by *Embedded Catalog (EC)* trees. Given the EC and a set of extraction rules STALKER is able to extract the data. To induce the extraction rules a user marks up relevant data on sample pages and STALKER applies a sequential covering algorithm for rule induction. Thereby, STALKER starts generating a rule that covers as many positive examples as possible. Then it tries to create another rule for the remaining examples, and so on. This procedure is accomplished until all examples are covered. After having a list of rules it refines the rules and creates a *disjunctive rule* to cover all examples.

STALKER handles only single-slot extraction, but requires a fewer set of training examples than other algorithms.

Lixto

Lixto [5] (Figure 3.4) is a declarative wrapper program for supervised wrapper generation and automated web information extraction. The system assists the user to semi-automatically create wrappers by providing a visual and interactive user interface. It provides a flexible hierarchical extraction pattern definition and deals with various kinds of conditions, such as contextual, internal, and range conditions, predefined concepts, references to other user-defined patterns, or comparison conditions.

The generated wrappers are relatively stable and easy to update. The definition of extraction patterns works on both the tree structure of the document and flat strings (i.e., regular expressions). The rules are presented in an internal rule language named ELOG, whereas the wrapper designer does not deal with ELOG. The wrapper designer can label instances directly on the web page and no working inside the HTML source or the tree representation is necessary.

XWrap

XWrap [26] is a semi-automatic wrapper generation framework. It consists of four components (Figure 3.5) for data wrapping: Syntactical Structure Normalization, Information Extraction, Code Generation, and Testing and Packing.

The Information Extraction component derives extraction rules using declarative specification to describe how to extract information content of interest from its HTML formatting. It performs this task in three steps: (1) identifying interesting regions in retrieved documents, (2) identifying important semantic tokens and their logical paths and node positions in the parse tree, and (3) identifying useful hierarchical structures of the retrieved

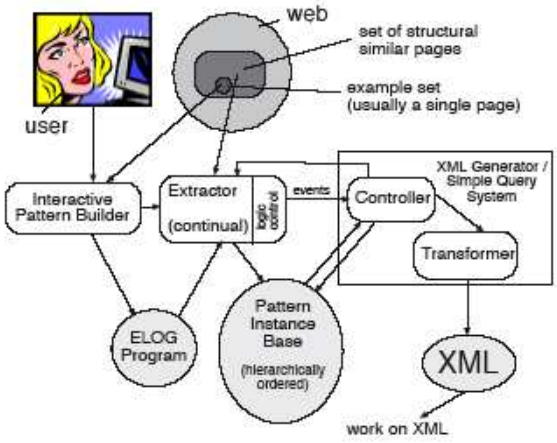

Figure 3.4: Overview of the Lixto system.

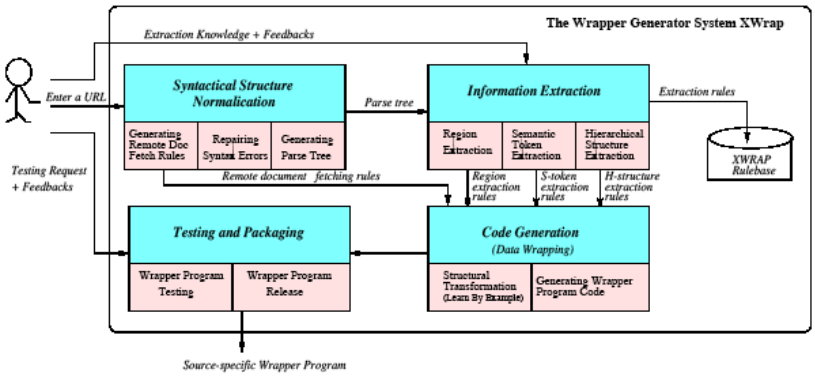

Figure 3.5: The XWrap system architecture.

document. Each step results in a set of extraction rules specified in declarative languages which are used by the Code Generation component to generate the wrapper program code.

3.4 Automatic Wrapper Generation

Automatic wrapper generation tools use unsupervised learning techniques. Therefore, no training sets are necessary, but a post-generation tuning.

ShopBot

ShopBot [15] is a domain-independent comparison-shopping agent. ShopBot autonomously learns how to shop at vendors given the home pages of several on-line stores. It relies on a combination of heuristic search, pattern matching, and inductive learning techniques and does not need to apply sophisticated natural language processing.

ShopBot operates in two phases: the learning phase, which is performed offfline, and the online comparison-shopping phase. During the learning phase the learner module (see Figure 3.6) automatically generates symbolic vendor descriptions of each site. Together with the domain description this is all the knowledge required by the comparison-shopping phase for finding products at this vendor.

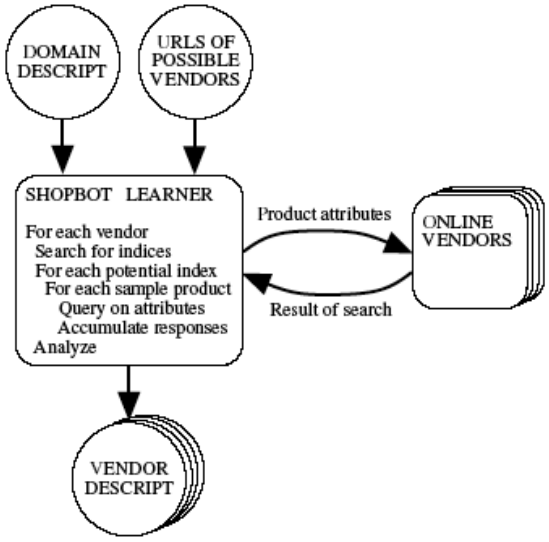

Figure 3.6: ShopBot learner module.

To learn the vendor description three components, which strongly interdepend, have to be considered: (1) identifying an appropriate search form, (2) determining how to fill in the form, and (3) distinguish the format of product descriptions from the resulting page.

First, the module searches for a set of candidate forms and computes an estimate for each form of how successful the comparison-shopping phase would be if the particular form were chosen by the learner. The estimation is done by filling the form and making several test queries using the form to search for several popular products. The test queries' results provide training examples from which the learner induces the format of product descriptions in the result pages from the used form as well as to compute the estimate measure for the form. After obtaining estimates for all forms, the learner picks up the form with the best

estimate and records the vendor description, how to fill the form, and the result pages of the form.

RoadRunner

RoadRunner [13, 12] is based on an unsupervised learning algorithm. Its goal is to automatically extract data from Web sources by exploiting similarities in page structure across multiple pages. RoadRunner works by inducing the grammar of Web pages by comparing several pages containing long lists of data. Its grammar is expressed at the HTML tag level. RoadRunner works well on data-intensive sites.

Given a set of sample HTML pages belonging to the same class, the nested type of the source dataset is found. The system compares the HTML codes of the pages and infers a common structure and a wrapper. These are used to extract the source dataset.

IEPAD

IEPAD (Information Extraction based on PAttern Discovery) [8] processes semi-structured texts by means of unsupervised inductive learning. It is more expressive than WIEN (cp. Section 3.3) and discovers extraction patterns from Web pages without user-labeled examples. IEPAD applies several pattern discovery techniques such as PAT trees, multiple string alignments, and pattern matching algorithms. Its extraction rules are pattern-based instead of delimiter-based and it can handle exceptions such as missing attributes, multiple attribute values, and variant attribute permutations.

Chapter 4

Conclusions

Research and development concerning Information Extraction started emerging in the late 1980ies. Research has been focused through the MUC conferences, which have also focused on the evaluation of IE systems and the definition of evaluation metrics.

Documents on which IE is applied can be structured, semi-structured, or unstructured. Unstructured documents (i.e., free text) require Natural Language Processing techniques. Documents containing semi-structured and structured data, which hardly contain full grammatical sentences, require delimiter-based techniques that can better analyze the document's structure.

Constructing IE systems or wrappers, which are tailored to extract information from web pages, is often cumbersome and labor-intensive. Both IE systems and wrappers can be generated manually, semi-automatically, or automatically. Manual pattern discovery and rule generation demand for a skilled knowledge engineer who is familiar with both the requirements of the application domain and the function of the IE system. Semi-automatic rule generation requires a large set of training data. This is used to derive the extraction rules by Machine Learning algorithms. As the costs for obtaining the training data sets are very high research has been directed towards automatic rule generation by means of unsupervised learning methods. These are based on bootstrapping methods that need only an initial small set of extraction patterns to learn the required rules.

Application areas for IE and wrapper generation systems are manifold and diversified. In the area of commercial systems comparison-shopping is vitally important. There, product information pages from different online vendors are fetched, relevant information is extracted and presented in a unified list to the user.

In the future, research will strongly focus on automatic learning to handle a large amount of dynamic documents, whereby the systems need to be flexible and scalable.

Acknowledgements

This work is supported by ”Fonds zur Förderung der wissenschaftlichen Forschung FWF” (Austrian Science Fund), grant P15467-INF.

Bibliography

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plaintext collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, 2000.

[2] D. E. Appelt. Introduction to information extraction. *AI Communications*, 12:161–172, 1999.

[3] D. E. Appelt, J. Hobbs, J. Bear, D. Israel, and M. Tyson. FASTUS: A finite-state processor for Information Extraction from real world text. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1172–1178, 1993.

[4] D. Ayuso, S. Boisen, H. Fox, H. Gish, R. Ingria, and R. Weischedel. BBN: Description of the PLUM system as used for MUC-4. In *Proceedings of the Fourth Message Understanding Conference (MUC-4)*, pages 169–176, 1992.

[5] R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of the Conference on Very Large Databases (VLDB)*, 2001.

[6] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extended Database Technology, EDBT'98*, 1998.

[7] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for Information Extraction. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 328–334, Orlando, FL, July 1999.

[8] C.-H. Chang, C.-N. Hsu, and S.-C. Lui. Automatic information extraction from semi-structured web pages by pattern discovery. *Decision Support Systems, Special Issue on Web Retrieval and Mining*, 35(1):129–147, April 2003.

[9] B. Chidlovskii. Automatic repairing of web wrappers. In *Proceeding of the Third International Workshop on Web Information and Data Management*, pages 24–30, Atlanta, Georgia, USA, 2001.

[10] B. Chidlovskii, U. Borghoff, and P. Chevalier. Towards sophisticated wrapping of web-based information repositories. In *Proceedings of the Conference of Computer-Assisted Information Retrieval*, pages 123–135, 1997.

[11] M. Collins and Y. Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing*, pages 100–110, 1999.

[12] V. Crescenzi, G. Mecca, and P. Merialdo. Automatic web information extraction in the RoadRunner system. In *International Workshop on Data Semantics in Web Information Systems (DASWIS-2001) in conjunction with 20th International Conference on Conceptual Modeling (ER 2001)*, 2001.

[13] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the Conference on Very Large Databases (VLDB'01)*, pages 109–118, 2001.

[14] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia, July 2002.

[15] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 39–48, Marina del Rey, CA, USA, 1997. ACM Press.

[16] M. Hearst. What is text mining. http://www.sims.berkeley.edu/~hearst/text-mining.html, 2004.

[17] J. R. Hobbs, D. Appelt, M. Tyson, J. Bear, and D. Islael. SRI International: Description of the FASTUS system used for MUC-4. In *Proceedings fo the 4th Message Understanding Conference (MUC-4)*, pages 268–275, 1992.

[18] C. N. Hsu and M. T. Dung. Wrapping semistructured web pages with finite-state transducers. In *Proceedings of the Conference on Automatic Learning and Discovery*, 1998.

[19] S. B. Huffman. Learning information extraction patterns from examples. In *Lecture Notes in Computer Science. Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040, pages 246–260, London, UK, 1996. Springer Verlag.

[20] J.-T. Kim and D. I. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):713–724, October 1995.

[21] G. Krupka, P. Jacobs, L. Rau, L. Childs, and I. Sider. GE NLTOOLSET: Description of the system as used for MUC-4. In *Proceedings of the 4th Message Understanding Conference (MUC-4)*, pages 177–185, 1992.

[22] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.

[23] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[24] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, 1997.

[25] W. Lehnert, C. Cardie, D. Fisher, J. McCarthy, E. Riloff, and S. Soderland. Evaluating an Information Extraction system. *Journal of Integrated Computer-Aided Engineering*, 1(6), 1994.

[26] L. Liu, C. Pu, and W. Han. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In *Intern. Conference on Data Engineering (ICDE)*, pages 611–621, 2000.

[27] J. McCarthy and W. Lehnert. Using decision trees for coreference resolution. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1050–1055, 1995.

[28] T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 305–310, Cambridge, MA, August 1977.

[29] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.

[30] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[31] L. Peshkin and A. Pfeffer. Bayesian information extraction network. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[32] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proc. of the 11th National conference on Artificial Intelligence*, pages 811–816, 1993.

[33] E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligende (AAAI-96)*, pages 1044–1049, 1996.

[34] E. Riloff. An empirical study of automated dictionary construction for information extraction in three domains. *Artificial Intelligence*, 85(1-2):101–134, 1996.

[35] E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 474–479. AAAI Press/MIT Press, 1999.

[36] S. Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1-3):233–272, 1999.

[37] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL: inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1314–1319, 1995.

[38] K. Sudo. *Unsupervised Discovery of Extraction Patterns for Information Extraction*. PhD thesis, New York University, New York, September 2004.

[39] K. Sudo, S. Sekine, and R. Grishman. Automatic pattern acquisition for Japanese Information Extraction. In *Proceedings of Human Language Technology Conference (HLT2001)*, San Diego, CA, 2001.

[40] R. Yangarber and R. Grishman. NYU: Description of the Proteus/PET system as used for MUC-7 ST. In *Proceedings of the 7th Message Understanding Conference: MUC-7*, Washington, DC, 1998.

[41] R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. Automatic acquisition of domain knowledge for information extraction. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany, August 2000.