Università della Calabria

Dipartimento di Matematica e Informatica

Dottorato di Ricerca in Matematica e Informatica

XXIX ciclo

Tesi di Dottorato

Ontology-driven Information Extraction

Settore Disciplinare INF/01 – INFORMATICA

Coordinatore: Ch.mo Prof. Nicola Leone

Supervisori: Ch.mo Prof. Nicola Leone

Prof. Marco Manna

Dottoranda: Dott. Weronika T. Adrian

Acknowledgements

It has been a beautiful journey and I owe my deep gratitude to a number of people who accompanied me on the way. First and foremost, I would like to thank my supervisor, Prof. Nicola Leone, who invited me to work and study in Calabria and provided the best environment for development I could dreamed of. He is an inspiring leader and a real problem-solver. Thank you, Professor, for all the particular solutions you offered along the way, and the overall experience of working and studying in the Department of Mathematics and Computer Science at University of Calabria.

I thank my co-supervisor Prof. Marco Manna for countless advice, corrections, lessons learned and hours spent doing the research. I have learned a lot from you. And I would like to thank you not only for the solid scientific formation I received, but also for your kindness and patience. Giving good advice is great, but being able to reach another person, taking into account their personality and character, is an impressive skill.

I thank Mina and Alessandra, who I worked with within the KnowRex research project, for showing me how effectiveness, solid programming work and a friendly atmosphere can be combined. Thank you Salvatore, Maria Elena, Gianni, Francesco and all the DLVSystem crew for receiving me as a "family" member. Your support in various situations was beyond what I could expect.

Thanks go to all my colleagues of the department, especially to fellow PhD students with whom I shared the room: Guillermo, Barbara and Bernardo, and to Alessia Cosentino who makes impossible possible on a daily basis. I have nowhere experienced such generosity of people in their willingness to help, offer their knowledge and advice as here. Working in this environment literally translates to a professional and personal growth.

Moreover, I would not be here, if I had not started my scientific career at AGH University of Science and Technology in Kraków. I would like to thank Prof. Antoni Ligęza, who first sparked my interest in logic programming and AI; Prof. Grzegorz J. Nalepa, who introduced me to the academia and was my first supervisor and a boss in the following years; and my dearest friends from GEIST research group. I learned a lot from you during the years spent at AGH and it remains an important time in my life.

The years of pursuing the PhD has also been a time of life changes. I would like to thank my parents and parents in law, my siblings Kacper and Ola and the best Szwagier Maciek, who supported us over these years, and my friends, especially Mirka, Olga, Kat, Krzyś and Krystian, for our invaluable friendship. Finally, I thank my dear family: Marek, Jaś and Maja, for being my greatest happiness and the best party for any adventure.

To my Grandpa

Abstract

Information Extraction consists in obtaining structured information from unstructured and semi-structured sources. Existing solutions use advanced methods from the field of Natural Language Processing and Artificial Intelligence, but they usually aim at solving sub-problems of IE, such as entity recognition, relation extraction or co-reference resolution. However, in practice, it is often necessary to build on the results of several tasks and arrange them in an intelligent way. Moreover, nowadays, Information Extraction faces new challenges related to the large-scale collections of documents in complex formats beyond plain text.

An apparent limitation of existing works is the lack of uniform representation of the document analysis from multiple perspectives, such as semantic annotation of text, structural analysis of the document layout and processing of the integrated knowledge. The recent proposals of *ontology-based* Information Extraction do not fully exploit the possibilities of ontologies, using them only as a reference model for a single extraction method, such as semantic annotation, or for defining the target schema for the extraction process.

In this thesis, we address the problem of Information Extraction from *homogeneous collections of documents* i.e., sets of files that share some common properties with respect to the content or layout. We observe that interleaving semantic and structural analysis can benefit the results of the IE process and propose an *ontology-driven* approach that integrates and extends existing solutions.

The contributions of this thesis are of theoretical and practical nature. With respect to the first, we propose a model and a process of *Semantic Information Extraction* that integrates techniques from semantic annotation of text, document layout analysis, object-oriented modeling and rule-based reasoning. We adapt existing solutions to enable their integration under a common ontological view and advance the state-of-the-art in the field of semantic annotation and document layout analysis. In particular, we propose a novel method for automatic lexicon generation for semantic annotators, and an original approach to layout analysis, based on common labels identification and structure recognition. We design and implement a framework named KnowRex that realize the proposed methodology and integrates the elaborated solutions.

Contents

List of Figures

Chapter 1

Introduction

Using machines to automatically extract relevant information from unstructured and semi-structured sources has practical significance in today's life and business. Structured data can be interpreted, analyzed and reasoned over easily and to some extend automatically. This in turn allows people to take reasonable decisions, such as recruiting a right person for a job (based on analysis of a pile of curricula) or buying a good house (by monitoring the real estate market postings).

The main shortcoming of the existing approaches to Information Extraction [25, 66, 11] is that they rely on the syntactic level of the information within the input files and lack understanding of extracted information by means of proper formal representation. Consequently, even small changes in the representation of content, organization of text etc. influence their performance. This motivates using higher level of abstraction to formally represent knowledge about the considered collection that would guide the process of IE.

Recently, some works have shown the promise of encoding formal knowledge in the form of ontologies [69, 68, 6, 45]. These approaches use ontologies either as a way to present the results of the extraction, or to allow matching different representations across sources. Still, the representation usually is limited to the *content* of the input files, while also the *structure* and *layout* of the documents could provide valuable information for the extraction process.

Combining different techniques to obtain comprehensive results is natural step forward within the IE domain [84, 26]. In this doctoral dissertation, we pursue this direction by decomposing the Information Extraction into sub-tasks, and addressing specific challenges within them. As a result, the topics we consider range from the meaning of single words and phrases to the layout and structure of a document, and from a unified representation of various relevant aspects to logic-based processing of the integrated knowledge. We analyze and solve separately specific problems identified within these areas. and then orchestrate the existing and novel solutions into a comprehensive Information Extraction framework.

1.1 Context

Information Extraction has been attracting research attention for a few decades. The problem originated in the Natural Language Processing (NLP) community in 1970s. With time, different sub-problems and areas of Information Extraction emerged, including entity, relation, and events extraction or co-reference resolution (see [81] for an overview of the sub-problems and respective methods). With respect to approaches, the initial efforts to build rule-based Information Extraction systems relied mostly on manually created regular expressions, cascaded finite-state transducers and lexicon-syntactic pattern [61]. Later on, a lot of attention was directed towards automatic learning of rules and patterns, training classifiers that would perform the extraction tasks etc. Supervised, semi-supervised and unsupervised methods have been studied, and different models, such as Hidden Markov Models (HMM) [104, 102, 107] or Conditional Random Fields (CRF) [75, 72, 100, 92] have been proposed. With the advent of the World Wide Web, a paradigm of *Open Information Extraction*, where no specific "target schema" is defined, has been attracting great attention as well [12, 38, 117, 37].

Currently, the landscape of solutions and tools is diversified, and depending on the characteristics of a task at hand different approaches give best results. For instance, if something is known about the structure or content of the considered input, one can apply the domain-specific knowledge. For building a general-purpose tool, one may want to take advantage of some typical properties of documents or statistical features of considered language. The "corporate research" is diversified too. On the one hand, there is a reborn of interest in rule-based techniques that follows the intuitive nature of rules [27]. On the other, there exist commercial tools that praise themselves for using modern AI solutions, such as *deep learning* and *word embeddings* [113]. Information Extraction is nowadays a mature field, in which two main directions of research and development are pursued: the first it to further enhance the specialized tools and techniques, and the second is to combine the existing solutions in a synergistic way.

As noted in [93], in the context of "emerging novel kinds of large-scale corpora," the established field of Information Extraction "assumes new dimensions and reinvents itself". In fact, dealing with the new types of document collections [17, 62] and the ever-increasing volumes of data poses new challenges for the IE systems, requires novel solutions and more intelligent techniques.

To address these new challenges, it is often necessary to abstract from the physical content of a file and analyze the input data by taking into consideration different aspects. When a human extract information from a set of somehow "similar" documents — be it with respect to their layout, structure or content — they intuitively use the common properties to locate and extract relevant information. For example, we can understand that some portions of text, serve only as spe-

cial "markers" that point to relevant information, and their implicit meaning can help extract more meaningful facts (think for example of specific section names in a collection of scientific papers that indicate the kind of content located within them). Another situation is when the structure of a document itself conveys some meaning and gives useful hints for locating desired information. For instance, a two-column layout may in fact "semantically" represent a table, if in the one column there are names of "attributes" and in the other – the values of them.

Such considerations require operating on a higher level of representing and processing knowledge. Unfortunately, most of the existing approaches to Information Extraction rely on the syntactic level of the document, even if they employ some semantic technologies. In this thesis, we address the problem of extracting structured information from *homogeneous collections of documents* i.e., sets of files that share some common properties. To automatically extract and generate instances for a target knowledge base, we propose a knowledge-based, *ontology-driven* approach to the problem that integrates and extends existing techniques.

1.2 Motivation

Motivation for research reported in this thesis stems from a practical need for a comprehensive IE system for collections of documents beyond the plain text format. While there exist numerous solutions for text, as well as for semi-structured documents tagged with some kind of markup language (such as HTML, XML etc.), extracting information from complex files such as PDF — where various textual and graphical elements can be placed and organized in different layouts — poses additional challenges [40, 42, 76]. PDF format in general does not guarantee a machine-readable representation of layout and structure. Instead, it must often be "recovered". Nevertheless, these complex files are widely used in many domains. This is why they should not only be considered, but also their analysis gives rise to interesting research challenges.

To focus our attention, we assume that the collections we analyze share some common properties, regarding layout or structure, or describing the same domain. To understand the intuition, one can imagine, for example, a collection of Curriculum Vitae documents: the documents share a topic (curricula of humans, typically even within a single profession), some structural information (sections about personal information, education etc.), and the layout which is often standardized.

For such characterized input, the general objective is to propose a knowledge-based approach to Information Extraction that would be domain-independent yet adaptable for different types of document collections. To this end, in this thesis we propose a framework of *Ontology-driven Information Extraction from homogeneous collections of documents*. To practically realize the approach, we in-

tegrate and extend solutions from the fields of *semantic annotation* [114], *table recognition* [119], *ontologies* [51] and *logic programming* [46, 18, 13].

In existing approaches, different sub-problems of Information Extraction are treated separately. Furthermore, some perspectives of document analysis are not even incorporated into the process. In our approach, we aim to integrate and extended where needed existing approaches and use an ontology as a reference model for all the techniques involved in the IE process. For instance, the output of annotating tools must be mapped to appropriate ontology concepts, the extraction rules must produce instances for the ontology etc.

The first sub-problem we consider is the *semantic annotation* of text. The objective of this process is to recognize pieces of text as instances of some abstract concepts. The methods from the field of Natural Language Processing (NLP) offer text analysis on a syntactic, morphological, grammatical and semantic levels. The annotation tools adopt different techniques for understanding the words and phrases, such as fixed lists of reference terms, patterns and templates or learning mechanisms. An interesting proposal in this respect are semi-supervised (or "distantly-supervised") methods for lexicon generation for semantic annotators [110, 23, 55]. These methods, given a set of example words, are able to dynamically extend the list to cover more words "of the same kind" so that the annotator can extract more domain-specific terms starting from a set of examples. This technique is worth studying and using in our framework, as it can serve in cases when for particular document collection existing annotators are insufficient.

If the documents share some common layout, it may be desirable to capture their common *labels*, typical keywords or structure elements. Although the text annotation allows to assign some semantics to text, this may be not enough when working with complex input data such as documents in PDF format. Understanding the "context" in which certain information appear in a document can significantly improve the results of the extraction. This is why we also consider methods of *document layout analysis* [116, 86]. In particular, the existing methods for two-dimensional structure analysis, including *table recognition* [119], would benefit from using some domain-specific knowledge in their otherwise "blind" processing. It is then important to study if and how these notions could be interleaved.

Basic objects recognized by appropriate semantic and structural tools, represented uniformly with use of an ontology, are suitable for further processing to extract more complex objects and relations. To this end, we study methods for formulating extraction rules that would allow for combining different levels of abstraction (e.g., semantics of a portion of text combined with a information about the placement of the text in a document). In particular, we draw inspirations from the HiLeX [79] system and its *semantic descriptors* that we extend such that they can operate on the two-dimensional document representation.

1.3 Contributions

In this thesis, we address the problems introduced in the previous sections and propose an ontology-driven approach to IE which allows for extracting semantically rich information from complex documents that share some common features. To this end, we integrate and extend recent technologies and results from the fields of classical information extraction, table recognition, ontologies, text annotation, and logic programming. In particular, we address the problems of semantic analysis of the document content, recognition of its structure, unified representation that covers both aspects, and methods of processing this knowledge for the IE purpose. We propose a framework for Information Extraction, and design and implement a system that allows to adapt the framework for different domains.

The proposed framework We provide a model and a process for what we call *Semantic Information Extraction*, and a software framework that realizes it. The conceptual architecture of the framework is illustrated in Figure 1.1. In the frame-

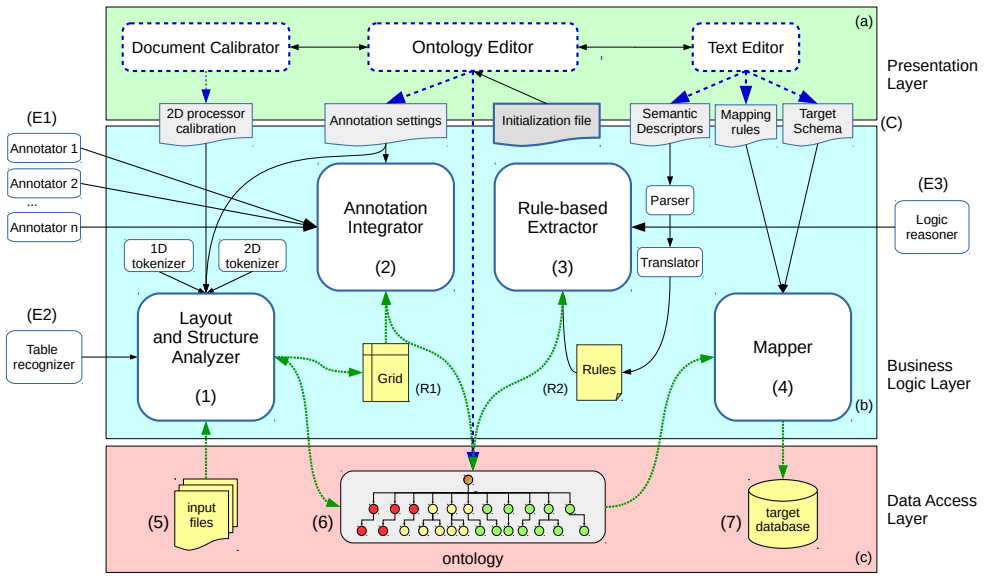


Figure 1.1: Architecture of the framework

work, we integrate several perspectives of document analysis: (1) layout and structure analysis, (2) semantic annotation of content and (3) rule-based information extraction. The framework allows to transform a collection of homogeneous data

i.e., files similar in some ways (5), by means of the ontological knowledge representation of the collection (6) into an instantiation of a target knowledge base (7). To this end, apart from the techniques of Semantic Information Extraction, the framework also supports (4) mapping to arbitrary target schema. Such an approach makes the framework flexible and modular.

Particular tasks are realized by individual components of the framework. Each of them (i) is guided by an appropriate configuration file (C); (ii) uses internal modules, such as tokenizers for text annotation, or parser and translator for rule-based extraction; and (iii) interacts with external tools: a pool of integrated annotators (E1), a two-dimensional document processor (E2) and a logic reasoner (E3). The input and output of the components is constituted by: the input PDF documents, the ontological representation (both the terminology and extracted instances), the intermediate knowledge representations (document "grid" (R1) and logic rules (R2)) and the set of instances for the target knowledge base.

The idea that guided our work was to make the system as intuitive as possible for a user. Thus, the interplay between different components is contained in the logic layer of the system (Fig. 1.1(b)), and the user interacts only with a graphical ontology editor with some additional text editor fields and a simple document calibrator for setting characteristics of the files such as margins etc. (Fig. 1.1(a)). The ontology designed by the user (or to be precise, extended from a generic one), guides the particular tasks, offering the input information and absorbing the extracted data that become the ontology instance.

In the proposed framework, the workflow of the operation is divided into phases of *design* and *runtime*. In particular, in the former, the user "explains" the specificity of the considered collection to the system (this is marked in Fig. 1.1 with dashed lines). Basically, this comes down to conceptualizing the collection, by extending the provided generic framework ontology with domain-specific concepts expected to be encountered. Then, the user must precise how to extract the ontological concepts. For "common" objects, they can select to use existing tools (annotators) available in the system. For the objects not recognizable by existing tools (e.g., collection-specific concepts or more advanced objects that have multiple attributes), they write extraction patterns (semantic descriptors) themselves. The user also specifies the target schema that should be populated as the result of the extraction and a mapping between the schema and the ontology.

All these activities serve to configure the system which then automatically performs the extraction over given input collection. The runtime phase workflow is depicted in Fig. 1.1 with dotted lines. In the runtime phase, the documents are first analyzed structurally, but with help of the semantic annotation of the collection-specific labels, and a two-dimensional "grid" representation of the documents is obtained. Then the content is analyzed semantically and recognized categories are assigned to words and phrases. Finally, using collection-specific extraction rules

designed by the user, more complex objects are extracted from the documents, completing the instantiation of the ontology. At the end, using logic rules, the ontology is transformed into the target knowledge base, for example a relational database.

Thesis outcomes and results The contribution of this thesis may be summarized as follows:

Semantic Annotation In the field of semantic annotation, we propose a unification framework that allows to combine the results of independent text annotation tools. Moreover, we propose a novel technique for automatic lexicon generation, if available tools are insufficient for particular domain.

Document Layout Analysis We propose a new approach for the analysis of document layout that combines table recognition technique with domain-specific knowledge about particular documents. The result of the proposed method is a two-dimensional "grid" representation of the input document.

Ontological Document Representation We propose a conceptualization of complex documents that takes into account both structure and content of the files. The proposed model can "guide" the Information Extraction by providing a reference ontology for all the steps in the process.

Rule-based Information Extraction We extend the formalism of semantic descriptors that are semantic rules for IE. In particular, we allow them to integrate various aspects of the input documents, such as the meaning of single words and phrases, information about the document structure etc.

Framework design and implementation We design and implement a framework, named KnowRex, which realizes our ontology-driven approach to IE. The system integrates the existing and novel solutions and remains intuitive, even for non-expert users.

Disclaimer The research for the thesis has been partialy carried out in collaboration with DLVSystem Srl., within the research project "KnowRex: Un sistema per il riconoscimento e l'estrazione di conoscenza", POR Calabria FESR 2007-2013. Part of the implementation was carried out by the corporate partners, DLVSystem Srl. and Exeura Srl.

1.4 Organization of the thesis

The structure of this thesis is as follows: In Chapter 2, we introduce the problem of semantic annotation of text and review existing relevant approaches and tools. Then in Chapter 3, we discuss the problem of lexicon generation for semantic annotators. We propose a novel method for automatic lexicon generation for semantic annotators by utilizing existing knowledge stored in semantic resources on the Web to build a set of instances for a desired semantic category starting from a set of "seed words". In Chapter 4, we consider the problem of document layout analysis and propose a method for building a two-dimensional grid representation of the structure of a document in PDF format. We combine the table recognition technique with semantic analysis of the document that focus on "common labels" i.e., phrases typical for a collection that mark certain sections etc. Then we propose a unified ontological representation of various aspects of document – both related to its content and structure – in Chapter 5. In Chapter 6, we describe an extended rule-based formalism of semantic descriptors, explain its capabilities and benefits for Information Extraction. In Chapter 7, we present the KnowRex framework that integrates the presented solutions and allows to practically realize the ontology-driven approach to IE. Finally, we evaluate the work in Chapter 8, and in Chapter 9 we conclude the thesis.

Chapter 2

Semantic Annotation

In this chapter, we introduce the problem of the semantic analysis of document content. We present the task of *text annotation*, and show how it is formulated, so that it can be solved automatically. We discuss existing approaches, methods and tools, and make a critical overview of them. The semantic annotation of documents is at the heart of any approach to recognize and extract information. In particular, any *annotation* process has the purpose of associating to input data some higher-level concepts (for example, classes of an ontology), so that the data gains relevant and shared meaning.

In order to propose a comprehensive method for representing, recognizing and extracting knowledge, it is essential to exploit as much as possible the existing semantic annotation techniques of documents which, suitably integrated and specialized, may allow to associate data to classes of an ontology and define ontological relationships between the various objects.

2.1 Different perspectives on text annotation

Text annotation problem can be approached from different perspectives, using a variety of methods and may be considered at different levels of abstraction. One can think of a strictly *syntactical* metadata, or one that describes a piece of text from the *grammatical* and *morphological* point of view; finally, portions of text can be associated to formal *ontological concepts*, defined elsewhere. In this section, we introduce selected tasks of semantic annotation. We discuss the goals they formulate and the representation and processing methods they use.

A number of criteria may be thought of when classifying text annotation. One can consider the level of formality of the used language, the placement of the annotation, the recipient of the annotation, the type of annotated resource etc. In particular, the level of formality may vary from strictly formal to informal. In

the case where a formal language is used, the annotation is expressed in a formal representation language such as RDF, RDFS or OWL, which are used for the definition of ontologies and metadata on the Web. The informal level refers, instead, to annotations expressed in natural language or in a controlled language, which allows to add information about documents or resources that can be used by a human user. With respect to placement, an annotation can be considered "embedded", inserted in the annotated document or "attached" i.e., stored separately and linked to the document by a link. The recipient of an annotation can be a human user or a machine. In the first case, the annotation is intended to provide additional information about a resource (in which case the annotation will typically be informal and probably expressed in natural language), in the latter case the annotation is intended to expose the meaning of a resource (in this case the annotation must be of a formal type for processing by a machine). Various types of resource can be annotated: documents, text fragments, HTML pages, images, etc., or even Web Services (for example, to express what a service does or to describe input and output parameters), annotations for data structures, or processes involved in the exchange of information between software applications that cooperate. Let us now review some of the annotation tasks and techniques.

Tokenization is the basic tool used for automatic text analysis. The task is to identify and report, within a text, the syntactically atomic linguistic units (or tokens) that compose it. The purpose is to subdivide the text into "pieces". It is therefore a preliminary step indispensable for any computational processing of the text, since it allows to divide character sequences into minimum units (words, punctuation marks, dates, numbers, abbreviations, and so on). Expected output is captured by a *token* concept. Such a token contains a string attribute that defines its value. For example, for an input a phrase like "aaa bbb cc ddd.", the tokenizer would return values such as "aaa", "bbb", "cc" and "ddd". A token could also be a structurally complex entity (e.g. a date). In that case it is assumed as a basic unit for subsequent processing levels (morphological, syntactic, etc.). In languages where word boundaries are not explicitly written in writing, the tokenization is called "word segmentation". There are, however, tokens that do not correspond to lexical units in the strict sense (for which a corresponding definition cannot be found in a lexicon). These include dates, acronyms, quantity and measurement expressions, proper names and punctuation marks.

The task of tokenization is not particularly complex with respect to the others, but there are some areas for ambiguity. For instance, if one considers the "dot" sign as the delimiter of the end of a sentence, one may be mistaken when the dot refers to an abbreviation or a date etc. Also, the task seems to be relatively simple for languages like Italian or English, that are based on spaces to delimit the

words; but for continuous spelling languages it is definitely more complex. In the first case, in fact, the token can be defined as any character sequence delimited by spaces. Even so, however, that definition leaves room for numerous exceptions.

Lemmatization is the process of reducing a word to its canonical form, called lemma. It is an operation that translates every word of a text into the basic form or dictionary entry. In particular, in the context of natural language processing, lemmatization is the algorithmic process that automatically determines the lemma of a certain word (and eventually annotate the word with the lemma).

The process may involve other language processing activities, such as morphological and grammatical analysis. It is undoubtedly complex, since it has to take into account the variety of language expressions that can be stated in a language. In many languages, the words can have different forms. For example, verbs may appear in different forms depending on a grammar tense, while the canonical form – the infinite mode – is what defines the word lemma and is the reference form for searching the word within a dictionary.

Grammar Analysis, Part Of Speech (POS) tagging is commonly referred to as a level of linguistic analysis of the text that assigns to each syntactically autonomous linguistic unit (or token), within a text, the set of relevant traits. These include typically the indication of the lemma, the grammatical category (or part of the speech), and the morphological traits associated with it. While the set of morphological traits may vary significantly from one annotation scheme to another (and from one language to another), all the morph-syntactic annotation schemes provide the indication of the grammatical category.

The feature provided by a POS tagger allows to assign to each token an annotation such as: Determiner (typically abbreviated as DT), Noun, singular or plural (NN), Preposition or subordinating conjunction (IN), Verb (VB)Proper Noun, singular (NNP) etc. In general, a typical POS task involves, first and foremost, the need to divide the text into sentences and tokens. At this point, it is possible for each token to locate in the dictionary the possible parts of the speech (nouns, verbs, adjectives, adverbs, pronouns, prepositions ...) to which it can be classified. After passing this step, the POS tagging phase allows to assign all possible POS tags to a word, using a dictionary and applying appropriate disambiguation rules.

Morphological and vocabulary analysis involves the consultation of specific lists of lemmas and their derivations (lists that can be optionally integrated with specific terms related to the domain being studied), the resolution of inflected forms (such as conjugation for verbs and declination for nouns) and the classification of words to certain categories (such as noun, pronoun, verb, adjective). A morphological analyzer attempts to locate the root and affixes (prefixes and suf-

fixes) of each word by obtaining morphomas, that is, the smaller linguistic units with semantic meaning, and generating decompositions. Vocabulary and morphological analyzer can be implemented separately, but they often work together on a single task. Even at this stage emerge the first ambiguity-related difficulties, in this case vocabulary-wise: for certain words the right morphology category to be attributed may not be well defined.

Syntactic analysis and generation of parse tree Syntactic analysis tries to identify the parts of a phrase and their function such as subject, predicate, object etc. Generally, the syntactic structure is represented as *parse tree*. Given a sentence on which lexical ambiguities have already been resolved with the right POS tagging, the syntactic analyzer re-elaborates it by creating the tree structure on the basis of appropriate rules.

Parsing has the goal of identifying the syntactic relationships between the elements of the text. It can be implemented at various levels of complexity. At this stage, the ambiguity of a syntactic type can appear. Consider, for example, the following phrase: "He saw a woman wearing glasses." It is not clear whether the man is wearing glasses and seeing a woman, or is the woman seen by the man wearing a pair of glasses. The context can certainly help resolve such ambiguities, but in the presence of articulated sentences one can encounter a proliferation of syntactic trees that gives rise to increased processing complexity.

Matching regular expressions Among the text manipulation mechanisms, it is interesting what is made available by regular expressions, typically used in text search techniques that are based on "pattern matching", that is, on the search for correspondence. Regular expressions are, in fact, used to perform a pattern matching operation on strings. A regular expression is a character pattern that follows a text-matching syntax that satisfies the conditions defined by the pattern. Formally, they are algebraic notations that describe strings patterns. They can be used to find individual characters, words, or even a more complex character scheme. In addition to their use for textual analysis, they are also used as mechanisms for validating forms, replacing text, user input processing, and so on. Regular expressions can easily formalize the rules for tokenize the text correctly with rules readable by tokenizers.

Semantic annotation based on ontologies Semantic annotation expresses semantic information associated with a resource (the meaning of a resource or portions of it). For such an annotation to be understood by a machine, it must be formal. In addition, in order to ensure the uniqueness of the interpretation of the

annotation, the terms that appear in the annotation should be selected from concepts and relations of a so-called *reference ontology*. An annotation with such features is an ontology-based annotation.

Noteworthy, an ontology, besides providing a common basis of knowledge on an established domain, by defining its main concepts and relationships and expressing meaning, allows to deduce new knowledge through the application of inference rules. This has interesting implications for using ontology-based annotations, as some implicit connections can be made explicit, by means of stating the relation of different text fragments to specific concepts of ontology.

It is possible to distinguish between simple annotations consisting of an existing concept in ontology and structured annotations, created by composition of concepts in the knowledge representation language of choice. Structured annotations allow the user to build an annotation best suited to express the meaning of the annotated resource.

Named Entity tagging includes the entity recognition methodologies that are typically used in the Information Extraction phases in which the elements of a text are categorized into predefined categories, such as names of people or organizations, quantities, monetary values, percentages, etc. The different approaches involve identifying, by means of machine learning rules or statistical methods, words or groups of words that specify entities belonging to appropriate categories of interest. Typically, a Named Entity Recognition task is divided into two steps: segmentation and classification. Segmentation identifies the "boundaries" of the entities within text, with the classification assigning them a semantic value (annotation). Entities can belong to some generally recognized categories, such as places or organizations, to common categories such as dates, measures, addresses or specific domains (pharmacological names, banks, and so on). Three approaches are generally used to recognize entities:

- *Lookup List or Gazetteers*: lists of objects grouped by categories;
- *Rule-based (pattern-matching)*: entities are identified using regular expressions that analyze the context and / or some features of the same entities as spelling, POS category, or other related characteristics;
- *Machine-trainable*: in this mode, the technology that has gained most recognition for the achieved results is the one of the Hidden Markov Model (HMM), probabilistic models that handle word sequences.

Semantic analysis of phrases This kind of the semantic analysis aims to extract the meaning of an entire phrase, starting from the meaning of all the term that

compose it and from the relationships between them. In fact, the meaning of a sentence is not only given by words, but also by the knowledge of the rules that decide their meaning on the basis of the combination, the order in which they appear, the ties that bind them to other terms, within or outside the sentence. An example of a semantic rule may be a one that presupposes the presence of the complement object when the verb is used in the sentence or attributes a semantic annotation that inserts the term in a precise context. This operation resolves some cases of global ambiguity (that refers to the whole sentence) that can appear even when POS tagging returns the correct morphological category of a word and the parser constructs the correct structure of the sentence.

2.2 Free and commercial tools

Currently, there is a multitude of tools that support semantic annotation. Part of them is available, others belong to organizations or companies and are kept private. In this section, we present and describe the selected tools, characterizing them from different perspectives.

In the literature, several ontology-based annotation systems as well as informal annotators have been presented (see [95, 114, 89] for more detailed surveys). A widely debated issue concerns the accuracy and reliability of the analysis offered by the annotation systems that can be subdivided into three different annotation mechanisms used: manual, semi-automatic and automatic. With the automatic annotation, a computer interprets the linguistic data without the manual intervention of the analyst, who at the programming stage of the system has established the rules and algorithms necessary to interpret them. Semi-automatic annotation systems provide a communication interface between the processor and the analyst that allows the latter to intervene to solve dubious cases by improving the accuracy and reliability of the results compared to the analyzes conducted through fully automated procedures. Manual annotation is a costly process. There are also several pattern-based annotation models or machine learning approaches. Annotation systems based on the patterns can use automatic patter discovery methods or manually defined patterns. Typically, one starts from an initial set of concepts from a document corpus to identify patterns in which these concepts are found. When new entities are detected, they are inserted into the initial set and the pattern detection continues until it can be extended further. Annotation systems based on machine learning algorithms use probabilistic or induction methods (statistical models for predicting entity location within documents, or inductive rules that lead to the identification of certain entities within the documents).

A further distinction between annotation systems can be made with respect to whether or not they are using ontological external knowledge. This feature

of supporting ontological models is particularly important in the context of this thesis. In fact, it becomes crucial to be able to manage ontological knowledge bases capable of providing a common meaning to the relevant entities for a given domain. Interest in these aspects is demonstrated by the so-called ontology-based annotators, annotation systems that exploit ontologies.

Every automatic annotation generated by these tools represents an instance of an ontology concept. A semantic annotation system based on ontologies can in fact operate through instances or concepts. In annotation through instances, the annotation consists of associating an instance with the annotated element in order to enhance the properties that describe the instance. In annotation through concepts, the annotation consists of an association of an ontology to the annotated element is an association of a composite of ontology concepts (through appropriate operators) to the annotated element. Below is a selection of annotation systems with a brief description of each.

OntoMat-Annotizer [59] is a tool that allows to annotate web pages based on ontologies expressed in RDF, DAML, DAML+OIL and OWL, using a user-friendly interface. It is a stand-alone application developed in Java that comes with a plug-in interface. The graphical interface is divided into two areas: in the first, one can view and explore an ontology (the taxonomic structure is represented by a tagged tree, with different icons for concepts, relationships and instances); in the other, there is an HTML browser. To construct an annotation, one selects a portion of text within the HTML page and associate it (through "drag'n'drop") to an existing or specially created ontology instance by enhancing the attributes and properties of a concept of an ontology. An annotation produced with OntoMat contains the definition of the instance expressed in a formal ontology language (RDF, DAML, DAML + OIL, OWL), and the URL of the file to which it refers. The annotation can be saved in the header of the annotated document, "embedded", or stored in an external file, and then "attached" to the original document. OntoMat annotations may be associated with any occurrence of a certain string within the file, but they do not relate to a specific position in the document, so when the user reopens an annotated document, they cannot locate the previously annotated text pieces. The system was developed by the University of Karlsruhe.

MnM [115] system provides both automatic and semi-automatic support for HTML page annotation, using the Amilcare Extract Tool. This stand-alone application developed in Java, integrates a web browser with an ontology editor, providing access to a local ontology or one available from a server. Knowledge representation (KR) languages supported by the system are RDF, DAML and DAML+OIL. The system allows to record both instances and concepts. Af-

ter selecting the reference ontology (RO) and uploading the HTML page to be annotated, the user must do the following: (1) select a concept from the ontology; (2) associate a portion of text highlighted with a property of the previously chosen concept; (3) create an instance or choose an existing one. For each selected concept, one must save the annotations, since the system does not allow to store annotations referring to different concepts in a single file. The annotation is stored using two files: the first, in the proprietary format, contains the ontology name and URL, the notation chosen for the annotation, and the properties used to mark HTML portions of the HTML page; the second, in XML format, includes a copy of the HTML page with the changes made by the markup operation. In case of annotation through instances, the set of instances created to annotate is saved in a file with reference to the concepts they derive from. This creates a knowledge base, making it possible to reuse the same instances for future annotations. The system is implemented by the Knowledge Media Institute, the Open University and the Department of Computer Science at the University of Sheffield.

Smore [67] is a stand-alone Java application that allows to combine document creation and annotation, including both an HTML page editor and an ontology editor. The reference ontology can be loaded from a local file, searched on the Web or built by the user. When creating the ontology, the user can choose whether to retrieve a concept from an existing ontology, or define it manually based on the RDF subject-predicate-object model. In order to build an RDFS triple, the user has to have a good understanding of the syntax and the language terms. Smore annotates HTML pages, images, and emails using RDF, DAML and DAML+OIL format ontologies. The RO along with the instances used to annotate can be saved by the user, such as a dataset file. The generated annotation can be stored either embedded in the header or as a separate file. The system was developed by the University of Maryland.

Cohse [14] annotation tool is a plug-in for both Internet Explorer and Mozilla, and presents itself as an additional taskbar within the browser. The taskbar offers several features: annotate highlighted text with the selected concept, explore the different ontology concepts, save annotations about an HTML page, and retrieve previously saved annotations. The RO exploration uses the OilEd [15] interface, the DAML + OIL ontology editor, developed in Java, which has been appropriately integrated with the annotation system. To annotate a portion of highlighted text, it builds an annotation expression and associates through appropriate connectors. Cohse allows to annotate both with a simple expression (single concept of RO) and a structured one (composition of multiple RO concepts). The annotation is in the same format as the ontology (DAML+OIL) and is "attached". The

reference to the file and, in particular, the position of the text that is annotated in the document is maintained using an XPointer expression. The annotation system interacts with the browser and a set of ontologies, providing the ability to store annotations made in an "Annotation Service" or in an "RDF Repository". Cohse can build annotation expressions and check their consistency using the features provided by the ontology editor and an inference engine (reasoner). The system was developed in the Department of Computer Science, University of Manchester, UK.

Melita [29, 28] is an ontology-based annotation viewer of textual documents. Its goal is not to provide an additional annotation interface, but to show how it is possible to interact actively with an Information Extractionsystem. Melita aims to reduce the time between the manual insertion of annotations and the learning phase in which annotations are processed by the IE system; such time frame is termed "timeliness", and usually the two phases are sequential. Melita implements a smart scheduling to try to reduce timeliness to the minimum. In fact, the IE system (Melita uses Amilcare) can begin computing the annotations automatically without the user having to annotate the initial document corpus, Melita can highlight different user annotations (with different colors), common automatic annotations (deduced by rules that have an average accuracy rate), and "certain" annotations (generated by rules that have a high accuracy rate) allowing the user to choose which of the various annotation proposals are to be maintained. The RO can be expressed in the format processed by Amilcare (file extension: .sce), or in a logical format file (file extension: .ont). Melita annotates text documents in embedded way by inserting into documents XML tags. Additionally, each user can store their annotations in an external XML file (called "Gazetteer") by listing for each concept all instances that have been annotated. Melita is a Client-Server application developed by the Department of Computer Science at the University of Sheffield.

Gate [32] processes a set of documents (XML, HTML, SGML, RTF, txt etc.) to automatically generate a set of annotated texts in XML format. Starting from a set of web page URLs and domain ontologies, expressed in RDF or DAML + OIL, it generates a set of instances of concepts belonging to the reference ontologies. Gate is a rule-based IE system, which means that extraction rules are defined *a priori*. The rules are expressed in the JAPE (Java Annotation Pattern Language) language and, in addition to recognizing the correspondence between a string in the document and an instance of an ontology concept, also identify possible new instances, utilizing techniques such as context analysis, recognition of part of speech, tags, and other indicators. The graphical interface displays in-

stances automatically generated by highlighting the text portions with the colors associated with the selected concepts in the ontology. The user can confirm or modify the instances proposed by the system. Annotations are represented in a direct acyclic graph where nodes are particular locations of the document and arches are annotations. The annotation format is based on the TIPSTER format [49]. Each annotation has an identifier, a type, a pair of nodes to indicate the start and end positions of the text within the document and an attribute value pair to express linguistic information and part of speech. The first version of Gate was built in 1996 in C ++ and provides context analysis for different languages: English, Greek, Spanish, Swedish, German, Italian and French. The latest version is implemented in Java and makes automatic annotations using RO. All were produced by the Department of Computer Science at the University of Sheffield.

KIM [94] is a platform that manages automatic semantic annotation for indexing and retrieving documents. The automatic learning capabilities of the system are based on Gate; KIM allows to annotate textual documents using the terms chosen for a default ontology called KIMO (KIM Ontology). KIMO is a "Lightweight Upper Level Ontology" consisting of generic concepts appropriate for any application domain. Ontology is expressed by applying the RDF (S) syntax with an expressive power limited to OWL Lite [16] (the RO cannot contain, for example, the definition of meta-classes). KIM is a plug-in for Internet Explorer and presents itself as an additional taskbar within the browser. The taskbar offers several features. The 'Classes' tab displays the class hierarchy, each element in the hierarchy is associated with a different color. To search for the associated instances, one has to select the corresponding check boxes and invoke the annotation of the current document. Calculated instances are stored in a knowledge base and sent to a dedicated attachment server. The 'Entity' tab contains a list of all recognized instances in the current document that is sorted by the number of occurrences on the page. Also, by selecting an instance within a page, one can retrieve all information about that instance, type, properties, and attributes. One can search for all documents with a particular instance, see if the query they want to make is already present among the most frequently asked requests stored by the system, and make complex queries by doing search by concept and imposing restrictions on the values assumed by certain properties. KIM was developed by Ontotext Lab, Sirma AI, Bulgaria.

SemTag [34] is an annotation system developed within the WebFountain research project. The IBM Almaden (San Francisco) researchers have used SemTag to annotate about 264 million web pages and generate 434 million disambiguated semantic annotations, which are made available as metadata. To define the annota-

tion classes, SemTag uses the TAP ontology [54], which is similar to the ontology used by KIM. To overcome the problem of disambiguation, SemTag uses a vector space model to associate a concept with the correct class or to say that a concept does not match a certain class of TAP. To do this, the context of the word (10 words to the left and 10 to the right) is compared with the contexts of individuals in TAP having aliases that are compatible with that word. TAP is built so that there are not too many entities that share the same alias, and this makes disambiguation simpler. SemTag has been developed as a parallel architecture, each node annotates about 200 documents per second. From the results of its execution, the authors found that 80% of the annotations were semantically correct, that is, the context of the annotated word was correctly identified by the system.

AlchemyAPI (`http://alchemyapi.com`) is a cloud-based text mining platform that provides tools for performing semantic type tagging. AlchemyAPI offers a set of natural language processing capabilities and can be used on text mining platforms. The framework uses linguistic analysis techniques, statistical language processing and auto-learning tools to analyze content and allow semantic metadata extraction. In particular, named entity extraction procedures help identify people, businesses, organizations, cities, geographic features, product descriptions, prices, and other types within documents (portions of text or HTML pages). AlchemyAPI allows to extract text labels and their placement based on visual and structural features. The demo interface provided by the Alchemy API tool allows to submit an URL or a text file. The results obtained are shown on the same page. Annotated terms are highlighted within the text and listed for different properties such as: relevance, sentiment (positive, negative, neutral, mixed), type (typology of the annotation), sub-types, and the linked data.

Stanford Named Entity Tagger [41] is a Java implementation of a Named Entity Recognizer. It is able to tag word sequences within a text that matches the names of people, companies, proteins, and so on. Stanford NER is also known as CRF Classifier. The CRF-based approach utilizes the statistical modeling methods offered by Conditional Random Fields for the implementation of pattern recognition and machine learning mechanisms. The software provides an implementation of a linear chain CRF (arbitrary) model. Therefore, after the model training phase one can use the code to construct sequence patterns for any extraction task. Annotation can be made by choosing a three, four, and seven reference classes. The interface is represented by an editor where one enters the text to annotate. Running the application shows the results that are highlighted with different colors to distinguish the identified entities.

Lupedia - Ontotext (http://lupedia.ontotext.com/) developed under the NoTube Lupedia project, is configured as a "text enrichment" service that use the Ontotext dictionary, called LKB Gazetteer, to locate the words and terms in DBpedia and LinkedMDB databases. It supports multiple languages, including English, Italian and French, provides several ways to filter output, and allows to set weights and heuristic criteria to get more accurate matching. The interface appears on a single page where one can enter the text to annotate, choose the lookup options, the language, and the datasets.

DBpedia Spotlight (http://dbpedia-spotlight.github.io/demo/) is an automated annotation tool that uses the knowledge base provided by DBpedia resources, providing a solution for linking unstructured information sources to the Linked Open Data Cloud through DBpedia. It uses named entity extraction mechanisms, including entity detection and name resolution (or disambiguation). It can be used to build an ad hoc solution for Named Entity Recognition processes. DBpedia Spotlight is able to recognize the names of concepts or entities (e.g. 'Michael Jordan') and establish a match between these names with unique identifiers (such as `dbpedia:Michael_I._Jordan` or `dbpedia:Michael_Jordan`). By linking text documents with resources from DBpedia, the system allows a number of interesting use cases. For example, ontology can be used as a knowledge base to display complementary information on web pages or to improve information retrieval activities. The Web application is a user interface that allows you to enter text and generates an annotated HTML version of the text. Web service endpoints provide demo access, which can also retrieve data in either XML or JSON format.

OpenCalais Annotator (http://opencalais.com/opencalais-demo/) is a component of Unstructured Information Management (UIM) applications for managing and manipulating unstructured information. They are systems that analyze large volumes of unstructured information, in order to find new knowledge that is relevant to the user. Open Calais is able to receive plain text, and identify and recognize a large number of entities such as people, places, organizations or relationships of the kind 'working for' or 'is at'.

2.3 A critical overview

The problem of text annotation has been considered for years, but the area remains open for further improvements. Although a number of methods and tools have been proposed, each has their strengths and limitations. The analysis of the state

of the art of annotation systems has highlighted the wealth and variety of technologies and tools currently available on the market as well as non-commercial tools. This, on the one hand, emphasizes the fact that the research sector addresses these issues and the ever-increasing interest from software houses, on the other hand, it poses the problem of how to use these technologies in the most profitable manner, namely, how to choose the most suitable and appropriate for the purposes of the application and the usage scenario.

Integrating existing solutions Guided by the selection criteria (related to the general goal: to integrate the tools into a comprehensive multi-perspective framework), we have carried out a qualitative analysis of the features and functionality of annotation tools currently available on the Web. The research has led to discovering dozens of annotation systems, each with features and goals that can be assimilated, but at the same time targeted at specific applications. In the end, many of those "discarded' are in a form of browsers, equipped with an editor for manual annotation of existing web pages (using techniques for pattern learning) and often configured as wrappers for HTML pages, while others focus mainly on multimedia content (images, videos). Some of the tools are used for creating and sharing ontologies on the Web, enabling the addition of metadata to documents, or as simple HTML editors for adding semantic annotations.

In the end, we have identified four annotation systems that were most suitable for our integration purpose. In particular, apart from the features that distinguish each of them, they are automatic annotation tools, capable of annotating a generic text, with respect to different configurable semantic categories, capable of returning the results of the complete annotation of information relating to the recognized semantic categories and their placement within the text. Moreover, for the identified annotators, it has been possible to have relevant resources (APIs, documentation, web services, online demos, etc.), an important element for making subsequent steps (integrating them into a common framework). These systems are, in particular, StanfordNER, Lupedia-Ontotext, OpenCalais and AlchemyAPI.

Advancing the state of the art As for the methods of annotation, each has their strengths and limitations. The use of gazetteers is certainly the simplest but not always applicable because it may require too long lists. Rule-based entity recognition has the benefit of simplicity in development, but it takes a long time to test the results on the corpora and refine the rules more than once. It is fast at run time and applied to standard texts ensures good performance. The machine-learning approach has the advantage of being more general and applicable without varying to different domains, but requires a lot of data for training.

An interesting solution is demonstrated by semantic annotators based on dic-

tionaries with *automatically generated lexicons*. Automatic lexicon generation overcomes the problem of manual creation of dictionaries (gazetteers, thesauri etc.) which is laborious and error-prone. The dictionary-based annotators are fast and flexible, especially if they provide additional mechanisms for tolerating minor deviations in wording. This is why, except for reusing existing solutions for ontology-based annotation, we decided to advance the state of the art in the direction of semantic lexicon generation. The problem and the proposed solution is further discussed in Chapter 3.

Chapter 3

Automatic Lexicon Generation

In this chapter, we continue our considerations about the semantics of the document. Out of numerous types of semantic annotators, introduced in Chapter 2, we focus on those that are based on *dictionaries* (or *lexicons*). In fact, these annotators are effective, accurate and fast. However, their "bottleneck" remains the construction of the *lexicon*. Manual creation of the dictionaries is laborious, so there is a need for automatic methods. In this chapter, we present the problem of automatic lexicon generation. We focus on methods that start from a few examples given by a human, and expand the dictionary with more words "like them" [101]. This task is often referred to as *entity set expansion* problem, and the methods that iteratively extend the initial seed set with new candidate instances are called *bootstrapping* algorithms. We propose a new approach to the entity set expansion problem, by using and integrating knowledge stored in the so-called *semantic resources* available online. We introduce a notion of *entity networks*, and propose a logic-based design and implementation of them. We also discuss the problem of *word sense disambiguation*, propose a method that solves it and implement it with use of answer set programming. This chapter is based the work described in the paper [5], accepted to the International Conference on Logic Programming.

3.1 Motivation

In this section, we introduce the problem and explain its intrinsic challenges. We define sub-problems that we treat separately, in order to combine their solutions later. The problem we study goes under the name of *entity set expansion*. Informally, given a set of words called *seeds*, the goal is to extend the original set with new words of the same "sort". For example, starting from *Rome* and *Budapest*, one could expand these seeds with *Amsterdam*, *Athens*, *Berlin*, ..., *Warsaw*, and *Zagreb*, which are also capital cities of European Union member states. But is

this the most appropriate way? In fact, an alternative expansion could be made by *Amsterdam*, *Berlin*, *Dublin*, ..., *Paris*, and *Prague*, which are also Europe's capitals situated on rivers. Moreover, *Rome* is not only a 'capital', but also a 'drama television series', a 'female deity', and many other things, while *Budapest* is also a 'film series' and a 'rock band', apart from being a 'capital'. Which is then the "best" common category of the original words? Are they 'capitals' or 'films'?

Our target application of the entity set expansion is to create lexicons for dictionary-based semantic annotators [97]. When manual creation of a dictionary is too costly, it is a practice to start with only a few examples and a text corpus, and then use various techniques to learn the target category and extend the list with more words. Sometimes the category is not even explicit, but given implicitly by the context in which the words appear, that serves as an indicator for what should be looked for.

However, the problem has also other applications such as knowledge management and search. Probably, the most famous application in this context is the *Knowledge Graph* by Google, based on which the search engine may suggest a list of somehow similar objects. First, one of the results of the Knowledge Graph is that for a desired category (one can try for instance 'museums in NYC' or 'top movies of 2016'), the search engine provides a "carousel" at the top of the results page that displays the possible instances of that category. Second, a functionality similar to set expansion is present in the 'People also searched for' section on the Knowledge Graph panel. Here, the goal is to suggest to the user something that will be of interest too and will enrich their research process. However, as this method is based on statistics, and more than on semantic analysis, it relies on other people's previous search sessions, and the results are sensitive to what other users perceive as related. For instance, starting from *Leonardo da Vinci*, the expanded set includes *Leonardo di Caprio*, *Pablo Picasso* and *Albert Einstein*, whose common sort remains unclear.

3.2 Related work

The task of automatic lexicon generation has been widely studied in the NLP community. Several approaches to tackle the problem of entity set expansion have been proposed. In particular, the idea of *bootstrapping algorithms* [98, 111, 64] consists in starting from a set of seeds, discovering *patterns* in which they appear in a given corpus, then using those patterns find more examples and repeating the process until an end condition is met. The patterns are usually lexico-syntactic, however, more complex ways of characterizing the words in a category to be expanded have also been proposed. In particular, in recent years the *word embeddings* are the most studied approach [21]. As far as the corpus is concerned,

the great potential of the Web has been recognized and used to extend the set of seeds [36, 101, 91]. As for the process itself, improvements have been proposed for each step: representing words [57], discovering patterns [22, 71], evaluating them [56] and minimizing so-called *semantic drift* [33].

Nevertheless, there are several problems with existing approaches. First, inherent limitation of statistical methods when analyzing the words, is that they do not take into consideration possible different senses of the same word, domain-specific exceptions etc. As pointed out in [64], methods that work well for generating "general" lexicons may fail for domains-specific dictionaries, when the meaning of words do not always agree with statistics. This problem has been addressed in [65] where athours propose a word representation that takes into account different word meanings. Instead, we propose to first select the meaning of words in the seed set that best fits the task context.

Moreover, the categories in existing approaches are usually as simple as a 'person', an 'animal', or a 'city'. We would like to go a step further and be able to discover more "descriptive" categories, by including the properties of the objects represented with the seed words (e.g., a "person born in Italy" or a "city locate by a river"). To this end, we propose to use knowledge available on the Web, specifically, stored in selected *semantic resources* that represent semantics of objects, their categorization and relations with other objects. We want to use these resources to disambiguate word meanings and discover commonalities among objects represented with them. Once the common category is singled out, we want to utilize the Web-harvested knowledge, specifically stored in the hypernym database built automatically using Hearst-like patterns [61]. This way, our approach combines structural knowledge from the semantic resources for analyzing and understanding objects, and Web-harvested knowledge to extend the set.

3.3 Semantic resources and entity networks

In order to understand a common category of objects, we will use the online *semantic resources*. We aim to integrate information from in them to combine the strengths and minimize weaknesses of the resources. To reason over the integrated knowledge, we will represent it with a novel model of an *entity network*.

Knowledge dispersed over the Web is nowadays in great part stored in various *semantic resources* — databases, created manually, semi-automatically or fully automatically — that store information about the world with some degree of formalization, organizing knowledge into ontologies, thesauri etc. People can use and combine information from different sources, because they understand its meaning and can make connections. It is thus desirable to design and implement automatic systems that acquire, process and use knowledge available on the Web

to solve particular problems given to it.

In this section, we propose a new method of integrating knowledge from existing *semantic resources* — from lexical databases, to general-purpose encyclopedias, that store information related to different aspects of human knowledge. We describe selected resources and introduce a notion of an *entity network* which is a unified model for representing classes, objects and relations among them. The integrated knowledge will serve to understand better the relations between objects represented by seed words given by a user in a process of automatic lexicon generation. We present an ASP-based tool that, given a set of words, can automatically create a representative entity network for them.

3.3.1 Semantic resources

Currently, more and more machine-readable knowledge is available on the Web in a form of *semantic resources*. These knowledge bases formalize and organize human knowledge about the world in different scope and manners, focus on various dimensions and areas of knowledge. There exist general-purpose knowledge bases, such as Wikipedia or OpenCyc, domain specific ontologies, computational lexicons such as WordNet and related projects, hybrid solutions that integrate knowledge from several sources and more. For the problem we address, we decided to use a combination of selected resources. Although main assumptions and the model remain the same regardless of the source selection, for reasons that we will become apparent in the next sections, we have selected the following: WordNet, Wikidata, BabelNet and WebIsADatabase. Let us now introduce each of them in more details.

WordNet [83] is a computational lexicon of English[1] that organizes concepts into sets of synonyms, called *synsets*. The synsets are interlinked via lexical and semantic relations (a different set of relations is defined for different parts of speech). As WordNet is manually curated, the resulting network is reliable and so this knowledge base became a widely acknowledged reference source in NLP community and beyond. An important feature of WordNet (from the viewpoint of our approach) is that if we select all synsets that represent nouns and the hypernym relations among these synsets, we can build a Directed Acyclic Graph (DAG) out of them. Moreover, the most general concept called 'entity' is reachable from all the synsets.[2] These two facts ensure that if we start from any noun synset and follow the hypernym relations, we will always reach the most general concept and never get into a cycle.

[1] There exist also satelite projects for other languages, not integrated with the core system.

[2] See https://wordnet.princeton.edu/#relations.

Wikidata (http://wikidata.org) is a free, open and collaboratively edited knowledge base, operated by the Wikimedia Foundation, that can be read and edited by humans and machines. It acts as central storage for the structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wikisource, and others. Wikidata is a document-oriented database, focused on *items*. Each item represents a topic and is identified by a unique number. The items are described with a set of *statements* (or claims). Statements take the form of key-value pairs, each consisting of a *property* (the key) and an *object* (the value linked to the property), both equipped with identifiers. Noteworthy, the semantic relations also have their pages with their properties etc.

BabelNet [88] is a multilingual terminological resource that integrates information from Wikipedia, Wordnet and other Web resources. It provides both encyclopedic knowledge about multiple instances (thanks to the integration with Wikipedia) and a dense network of relations among the entries (by integrating relations from Wordnet, Wikidata, DBPedia, Freebase and others). BabelNet organizes information into *Babel synsets* – multilingually lexicalized concepts. Within the description of a synset, there are synonyms, definitions, examples, sometimes also pictures, and semantic relations with other synsets, including taxonomic relations. Each relation is encoded as an "edge" that is a complex object with different properties, such as language, name, relation group etc. What is important, BabelNet provides links to other resources, by means of which it indicates which entries in other knowledge bases correspond to the given entry (synset) in BabelNet. Therefore, BabelNet can serve as a "knowledge hub" i.e., it lets one explore different resources and integrate information from them. The taxonomy in BabelNet is not guaranteed to be acyclic.

WebIsADatabase [103] is a publicly available database containing more than 400 million hypernymy relations extracted from the CommonCrawl web corpus (http://commoncrawl.org/). The tuples of the database are created by harvesting the corpus and applying lexico-syntactic patterns, such as: NP_t *is a* NP_h, NP_h *including* NP_t, NP_h *such as* NP_t etc., where NP_t indicates the hyponym and NP_h the hypernym. The dataset can be queried both for classes of a given instance, and for instances of a given class. [3] A distinguishing feature of the WebIsADb is that it tackled the problem of the so-called "long tail" i.e., it harvested the Web corpus to identify not only the most popular named entities, but also the less known ones. Thus, using the WebIsADb can enrich the existing knowledge bases with new facts. In our approach, we resort to this semantic resource in the phase of discovering new instances of the target category.

[3] See WebIsADb demo at: http://webisadb.webdatacommons.org/webisadb/.

3.3.2 Entity networks

To integrate knowledge from several semantic resources, we propose a model that can uniformly represent information acquired from them. The basic notions we will use are *(semantic) entities* and an *(entity) network*. An entity is a pair $\varepsilon = \langle id(\varepsilon), names(\varepsilon) \rangle$, where $id(\varepsilon)$ is the identifier of ε, and $names(\varepsilon)$ is a set of (human readable) terms describing ε.

From a syntactic viewpoint, $id(\varepsilon)$ is a set of strings of the form *src* : *code* where *src* identifies the semantic resource where ε is classified, and *code* is the local identifier within source *src*. In turn, $names(\varepsilon)$ is nothing else but a set of strings. For example,

$$\varepsilon = \langle \{\texttt{wn:08864547}, \texttt{wd:Q40}, \texttt{bn:00007266n}\}, \{\texttt{Austria}, \texttt{Oesterreich}\} \rangle$$

is an entity representing the object, the Republic of Austria, referred to in WordNet (abbreviation `wn` with identifier `08864547`), Wikidata (abbreviated `wd` with item identifier `Q40`), and in BabelNet (with synset identifier `bn:00007266n`).

From a semantic point of view, entities may refer to three different kinds of objects. Namely, they can either point to (i) individuals, called hereafter *instances*, such as in the previous example, where the entity denotes a particular country, or (ii) concepts that generalize a *class* of objects e.g., $\varepsilon = \langle \{$ `wn:08562388`, `wd:Q6256`, `bn:00023235n` $\}$, $\{$`country`$\} \rangle$ or (iii) *(semantic) relations* that hold between two objects e.g., $\varepsilon = \langle \{\texttt{wd:P31}\}, \{\texttt{instance of, is a, ...}\} \rangle$ or $\varepsilon = \langle \{$ `wd:P131`$\}, \{\texttt{located in, ...}\} \rangle$ etc. For convenience, we will group the entities representing instances and classes into one group, so-called *(knowledge) units* and the relations will be a separate group.

An *(entity) network* is a four-tuple $\mathcal{N} = \langle Uni, Rel, Con, type \rangle$ where:

- (i) *Uni* is a set of knowledge units, both classes and instances;
- (ii) *Rel* is a set of semantic relations;
- (iii) $Con \subseteq Uni \times Uni$ is a set of ordered pairs denoting that two units are connected via some (one or more) semantic relations;
- (iv) $type : Con \rightarrow (2^{Rel} \setminus \emptyset)$ is a function that assigns to each connection a set of semantic relations; and
- (v) the pair $graph(\mathcal{N}) = (Uni, Con)$ is the directed graph underlying $\mathcal{N}$.

An entity network $\mathcal{N}$ is consistent if for each pair $\varepsilon_1, \varepsilon_2$ of different entities of $\mathcal{N}$, $id(\varepsilon_1) \cap id(\varepsilon_2) = \emptyset$.

Example 3.3.1
Consider the network $\mathcal{N} = \langle Uni, Rel, Con, type \rangle$ *shown in Figure 3.1. For clarity, identifiers are left implicit, and each entity is described by one representative name. In particular,* $Uni = \{\varepsilon_1, \ldots, \varepsilon_5\}$*, where only* ε_5 *is a class,* $Rel = \{\varepsilon_6, \ldots, \varepsilon_{10}\}$*, Con are the arcs connecting units and, for each pair* (u, v) *of connected units,* $type(u, v)$ *contains the relations graphically associated to arc* (u, v)*.* $\triangleleft$

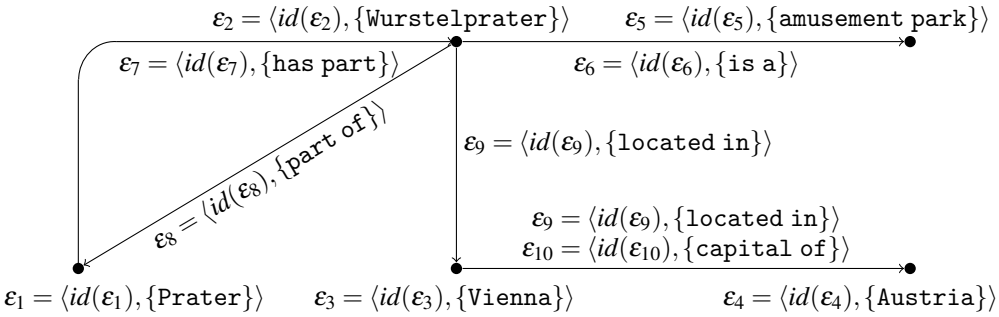


Figure 3.1: An entity network example

From a knowledge representation viewpoint, two semantic relations are commonly referred to as 'is-a': the membership relation 'instance-of' and the subtype relation 'subclass-of'. In particular, BabelNet follows this standard by displaying only 'is-a' when referring both to membership and to subtype relations. On the contrary, Wikidata keeps separate pages (and identifiers) for them. Finally, WordNet does not give any explicit identifier to these relations, but it considers the 'instance' relation as a specific form of hyponym.[4] In our approach, both relations will be treated uniformly as in BabelNet. To this end, we define the following useful set of relations:

$$Isa = \{\langle \{\texttt{wd:P279}\}, \{\texttt{subclass}, \texttt{is a}, \ldots\} \rangle, \langle \{\texttt{wd:P31}\}, \{\texttt{instance}, \texttt{is a}, \ldots\} \rangle\}$$

grouping together the above two entities. Given a network $\mathcal{N}$, it is convenient to refer to the set $Con|_{Isa} = \{(u, v) \in Con \mid Isa \cap type(u, v) \neq \emptyset\}$ to select the 'is-a' connections only.

3.3.3 ASP-based network construction

To construct an entity network, we propose an encoding in Answer Set Programming [46, 18, 13], enriched with *external predicates* [20], by means of which

[4] See http://wordnet.princeton.edu/wordnet/man/wngloss.7WN.html

semantic resources can be queried. The external predicates refer to functions, implemented separately, that encapsulate requests to semantic resources and interpret their responses. This makes the solution modular and easily extensible: addition of a new resource requires only an addition of a new rule and a new (typically very simple) function, compatible with the resource's API.

The "network builder" program consists of logical rules of the form

$$h \leftarrow l_1, \ldots, l_n$$

where h is an atom in the head of rule, and $l_1, \ldots, l_n$ are positive literals in the body of rule. The literals may be either atoms of the form:

- `p` $(t_0, ..., t_n)$ where `p` is a predicate name and $t_0, ..., t_n$ are terms, or
- external predicates encoded as `&p` $(t_0, ..., t_n$ `;` $u_0, ..., u_m)$ where:
  - `&p` is an external predicate (the name must start with a & symbol),
  - $t_0, ..., t_n$ are input terms,
  - $u_0, ..., u_m$ are output terms,
  - and a semicolon symbol (";") separates input from output terms.

In fact, all the rules that call external predicates establish new connections and are of the general form:

$$\begin{aligned} &newConnection(unit_1, unit_2[, optional\ arguments]) \leftarrow unitID(unit_1), \\ &\qquad \&externalPredicate(unit_1; unit_2)[, (optional\ restrictions)]. \end{aligned}$$

The program is divided into *sub-modules* that are sets of rules responsible for distinct tasks. The overall strategy and workflow of "calling" appropriate modules is described in Section 3.5.

The first task is to understand the meaning of input words and create entities for them. To this end, as a primary resource for word senses, we have selected BabelNet due to its wide coverage of instances and integration of knowledge from multiple sources. Let us consider a set W of seed words. The construction of a network starts with a mapping μ which associates a fact to each word $w \in W$ as follows: $\mu(W) = \{\texttt{seed}(w) \mid w \in W\}$. The following rule, starting form a seed word, queries BabelNet for possible meanings (with the predicate *&babelnetSense*) and infer a set of relations *senseOf* that connect the words with unit identifiers:

```
senseOf(SeedWord, SenseID) :- seed(SeedWord),
                      &babelnetSense(SeedWord; SenseID).
```

There may be more meanings for a single word. This problem is addressed separately as described in Section 3.4.

Note that we do not want to include all the knowledge that is somehow reachable from the seeds in the semantic resources, but only a small representative subset of it. Thus, we introduce a mechanism of "levels" that we assign to units in the network. In particular, when acquiring the meanings of words, we assert the following fact in our knowledge base:

```
babelnetID(SenseID,1) :- senseOf(Word, SenseID).
```

The second argument of the predicate *babelnetID* denotes the level. This parameter lets us control the application of some rules, as we will show later.

The second task for the entity network builder is to "better understand" the objects of concern. In particular, we address this problem with the following two sub-tasks: the first is to understand the taxonomy (hierarchy of types to which an entity belongs), and the second is to expand the network for the semantic relations beyond the taxonomy. To realize these tasks, we utilize the BabelNet functionality of providing links from its synsets to other resources, especially WordNet and Wikidata. With the following rules, we can establish "bridges" that let us jump between resources and integrate information from them:

```
eqBnWn(BID, WNID) :- babelnetID(BID,Lv), &bnWnEq(BID; WNID).
wordnetID(WNID) :- eqBnWn(_, WNID).

eqBnWd(BID, WDID) :- babelnetID(BID,Lv), &bnWdEq(BID; WDID).
wikidataID(WDID) :- eqBnWd(_, WDID).
```

The external predicates *&bnEnEq* and *&bnWdEq* simply ask BabelNet for links to other resources for the particular BabelNet entry identified with BabelNet ID `BID`. The results of establishing these "bridges" are two-fold. On the one hand, we will query selected resources best suitable for particular tasks, and on the other, we will keep the network consistent by knowing which entries in different resources refer to the same object.

Let us now explain the expansion of a network in the direction of the hypernyms. We approach the problem by combining information from BabelNet and WordNet. On one hand, BabelNet is the best source for membership relations for individual objects, with better coverage than any other considered resource. On the other, if we want to analyze the hierarchy of classes, the WordNet hierarchy is more reliable, and – what is important – it is guaranteed to be acyclic. Thus, we query both BabelNet and WordNet for hypernyms of concepts, but we restrict BabelNet queries only to some level, using a *babelnetDepth* predicate, e.g.

`babelnetDepth(3).` Conversely, we query WordNet for all the hypernyms, up to the most general concept.

The following rule, starting from a unit identified with a BabelNet ID, queries BabelNet for hypernyms and establish a new connection, *bnISA* between the unit and its hypernym:

```
bnISA(ID, PID, PLv) :- babelnetID(ID,Lv), &bnetISA(ID; PID),
    babelnetDepth(BabelNetMax), Lv<BabelNetMax, PLv = Lv +1.
```

Here, the level of the starting unit is checked (parameter *Lv* is compared to the value *BabelNetMax*), and the level of its hypernym is set to the level one step higher. This limits the number of applications of the rule. Conversely, the rule that queries WordNet contains no restrictions:

```
wnISA(ID, PID) :- wordnetID(ID), &wnetISA(ID;PID).
```

As for the other semantic relations, we select Wikidata as a primary source, because it assigns to relations unique identifiers. As we stated before, we know which objec (and representing it item in Wikidata) we are interested in, so we can start with a Wikidata ID and ask for the item's properties. For each property, we get information about the name and identifier of the relation (*ID* and *Name* attributes in the rule below) and the target item *Y*. We establish new connections representing the statements with the following rule:

```
wdRel(X, Y, ID, Name) :- wdataID(X), &widRel(X; Y, ID, Name).
```

The size of the graphs underlying the entity networks depends mostly on two factors: the number of possible meanings of the input words and the depth in a WordNet hierarchy of the initial units (as most of the edges represent hypernym relations from WordNet). For instance, for two seed words, the network may have as few as 20 connections, or as many as a few hundreds.

The network building module consists of a single file with the ASP program and a set of files with implementations of the functions that query external resources. Each resource provides an API, in case of BabelNet and Wikidata, an individual *key* is also required. For executing the program, we use the idlv [20] grounder that supports external predicates. The functions querying the resources have been written in Python.

3.4 Word sense disambiguation

In this section, we discuss the second sub-problem signaled before, namely, the *word sense disambiguation* (WSD) which allows to recognize the "correct" meaning of a word, if more senses are possible. Polysemy of words is a challenge, that

in the context of thesaurus construction, is faced, when the seed words have more than one meaning. Analysis of commonalities among the seeds, taking into account all possible senses of all the seed words, gives rise to costly and useless computation. Thus, to make our method more effective, we address this problem, when we face polysemous words. In this section, we introduce the theoretical foundations of our approach, as well as a tool, implemented in logic, that for a set of words, designates their "best" senses.

Due to the words' polysemy, an entity network constructed for a set of words W may contain multiple units for each single word $w \in W$ that represent the word's different meanings. As we are interested in determining a common category of objects represented by W, but want to avoid useless network expansion for all possible word senses, we first address the problem of *word sense disambiguation*.

Following the definition stated in [87], given a text T that can be viewed as a sequence of words $(w_1, w_2, \ldots, w_n)$, WSD is a task of assigning the appropriate sense(s) to all or some of the words in T, that is to identify a mapping A from words to senses, such that $A(i) \subseteq Senses_D(w_i)$, where $Senses_D(w_i)$ is the set of senses encoded in a dictionary D for word w_i and $A(i)$ is that subset of the senses of w_i which are appropriate in the context T.

Note that in our setting, which is a little bit different from the classical one, we do not consider words in a sequence, but rather in a set (of seed words), and instead of one dictionary, we assume a combination of semantic resources. Nevertheless, the goal remains the same: out of possible meanings, we want to select the one that is "the best" in the given context.

In the following section, we introduce a notion of an *optimal common ancestor (OCA)* in a directed acyclic graph. This notion captures an intuition that if we analyze the taxonomy of the objects that represent different word senses, then finding the "closest" common supertype of all the words will point the correct senses of words. We describe a declarative encoding in answer set programming that uses guess-check-optimize paradigm to determine the set of optimal common ancestors given a set of words and their entity network.

3.4.1 Optimal common ancestors

Our approach follows the intuition that the "correct" word meaning can be selected out of its possible senses with the help of other seed words. More precisely, if we pick a sense per word, find a common supertype for all the picked senses, assign a score to this ancestor, that will reflect its "closeness" to the words, then we will be able to determine the best combination of senses, and thus the best sense for each word in W.

Let us consider the word: *Python* that has 16 senses in BabelNet. It may denote a 'reptile', a 'mythological dragon', a 'missile', a 'programming language',

a 'rollercoaster' in Efteling amusement park or even the 'Monty Python group'. Similarily, *Java* (18 senses in BabelNet) is a name of an island, a coffee, a programming language, a spirit, a town, a breed of chicken etc. If we put *Java* and *Python* as seeds for entity set expansion, we assume that these words represent the same category. As it appears, out of possible meanings of both *Java* and *Python*, there are three senses that are shared by these words, namely: a 'programming language', a 'name', and a 'band'. Thus, intuitively, we should select one of the compatible pair of senses from this set, and not for example, an 'island' for *Java* and a 'reptile' for *Python* (for which some common supertype may be a 'physical entity'). Note that, if we put more seeds into the set e.g., add *C++*, the context would change, and the supertype closest to all three would be only the 'programming language'. Let us now formalize the notions.

Definition 3.4.1 (Common ancestor)
Consider a directed acyclic graph $G = (N,A)$, and a nonempty set $S \subseteq N$ of nodes called seeds. *A node $a \in N$ is a* common ancestor *of S with respect to G if, for each seed $s \in S$, either $s = a$, or there is a path in G from s to a. If so, a is often called a common ancestor of (S,G), for short.*

Example 3.4.1
Consider the directed graph $G = (\{1,2,3,4\},\{(1,2),(1,3),(2,4),(3,4)\})$, which is clearly acyclic. One can verify that node 4 is a common ancestor of $(\{1,2\},G)$ since there is a path from node 1 to node 4, and a path from node 2 to node 4. Conversely, 3 is not a common ancestor of $(\{1,2\},G)$ since node 3 is not reachable from node 2. But 3 is a common ancestor of $(\{1,3\},G)$ since 3 is also a seed and there is an edge from node 1 to node 3. ◁

Consider now a common ancestor a of some pair (S,G), with $G = (N,A)$. The *distance* of a from S in G, denoted by $dist(a,S,G)$, is the nonnegative integer k such that the following conditions are both satisfied:

1. there is $V \subseteq A$ such that both $|V| = k$ and a is a common ancestor of $(S,(N,V))$; and

2. there is no $V' \subseteq A$ such that both $|V'| < |V|$ and a is a common ancestor of $(S,(N,V'))$.

Note that $dist(a,\{a\},G) = 0$ trivially holds. Basically, the distance counts the the minimum number of arcs sufficient for connecting each seed to the given ancestor. According to the previous example, we have that $dist(4,\{1,2\},G) = 2$ since $V = \{(1,2),(2,4)\}$ is the smallest set of arcs sufficient to connect both seeds to node 4.

Definition 3.4.2 (Optimal common ancestor)
A common ancestor a of some pair (S,G) is optimal *if, for each common ancestor a' of (S,G), it holds that $dist(a,S,G) \leqslant dist(a',S,G)$. Hereinafter, $optca(S,G)$ denotes the set of all optimal common ancestors of (S,G).* $\triangleleft$

Given an optimal common ancestor a, we call a $witness(a)$ the set $\{V \subseteq A \mid dist(a,S,G) = |V|\}$. Since G is acyclic, each $V \in witness(a)$ represents a tree connecting each node of S to a. By referring again to the previous example, $optca(\{1,2\},G) = \{2\}$ since $dist(2,\{1,2\},G) = 1$ due to the unique witness $V = \{(1,2)\}$.

3.4.2 ASP-based sense detector

To detect appropriate senses for a set W of words we proceed as follows. First, we build from W its associated entity network $\mathcal{N} = (Uni, Rel, Con, type)$, as described in Section 3.3.3. Second, we build from W and $\mathcal{N}$ the directed acyclic graph $G(W,\mathcal{N}) = (N,A)$ where $N = W \cup Uni$ and $A = Con|_{Isa} \cup \{(w,u) \mid w \in W \ \wedge \ u \in Uni \ \wedge \ w \in names(u)\}$. Third, according to Definition 3.4.2, we compute all optimal common ancestors of the pair (W,G) as well as their associated witnesses. Fourth, from each optimal common ancestor a and each $V \in witness(a)$, we extract the set $\{(w,u) \in V \mid w \in W\}$ where each (w,u) says that unit u is an appropriate sense for w with respect to a (since V is a tree, each w cannot have more senses). The last two steps are carried out via Answer Set Programming. More precisely, we define a mapping μ that encodes the input pair (W,G) as set of facts, and design a program P and a weak constraint ω such that, a_{opt} is an optimal common ancestor of (W,G) if, and only if, a_{opt} is encoded as appropriate atom in some answer set of $\mu(W,G) \cup P \cup \omega$.

As for the input, we have $\mu(W,G) = \{\texttt{edge(}u\texttt{,}v\texttt{)} \mid (u,v) \in A\} \cup \{\texttt{seed(}w\texttt{)} \mid w \in W\}$. Regarding P, we gradually introduce and explain its rules. To reduce the search space, we identify a suitable set $C \subseteq Uni$ of "candidate" optimal common ancestors as follow:

```
unit(U) :- edge(_,U).
hasAncestor(W,U) :- seed(W), edge(W,U).
hasAncestor(W,V) :- hasAncestor(W,U), edge(U,V).
partialAncestor(U) :- seed(W), unit(U), not hasAncestor(W,U).
ancestor(U) :- unit(U), not partialAncestor(U).
superAncestor(V) :- ancestor(U), edge(U,V).
candidateOptAncestor(U) :- ancestor(U), not superAncestor(U).
```

The first rule determines the set $\{\texttt{unit(}u\texttt{)} \mid u \in Uni\}$. The subsequent two rules determine, for each $w \in W$, which are the ancestors of w. The forth rule defines

units that are not common ancestors. Rule five identifies the common ancestors. Rule six detects parents of common ancestors, which of course cannot be optimal. Rule seven defines the candidate optimal common ancestors.

All the atoms derived so far are obtained deterministically, and they are part of every answer set of $\mu(W,G) \cup P \cup \omega$. Conversely, to identify the optimal common ancestors of (W,G), we need to consider separately each candidate optimal common ancestor:

```
keepAncestor(X) | discdAncestor(X) :- candidateOptAncestor(X).
:- not #count{X:keepAncestor(X)} = 1.
```

The first disjunctive rule guesses some candidate ancestors. The second rule (a strong constraint) imposes that each answer set may contain only one candidate ancestor.

Once a candidate ancestor is kept, we need to guess a suitable witness. To this end, to reduce again the search space, we consider only arcs forming paths to the guessed ancestor:

```
activeEdge(U,V) :- edge(U,V), keepAncestor(V).
activeEdge(U,V) :- edge(U,V), activeEdge(V,T).
keep(U,V) | discard(U,V) :- activeEdge(U,V).
senseOf(W,U) :- seed(W), keep(W,U).
reach(W,U) :- senseOf(W,U).
reach(W,V) :- reach(W,U), keep(U,V).
:- seed(W), keepAncestor(U), not reach(W,U).
```

The first two rules mark as *active* the arcs reaching the kept ancestor. Disjunctive rule three guesses a witness. Rule four determines the sense associated to each word. Rules five and six determine which units are reachable from the seeds, according to the guessed witness. Rule seven (a strong constraint) guarantees that the guessed ancestor is reachable from each seed.

For each guessed ancestor a, we now compute the distance $dist(a,W,G)$. For safety reasons, we add to P the auxiliary set of atoms `distanceRange(1..`κ`)`, where $\kappa = |A|$:

```
distance(N) :- distanceRange(N), N = #count{X,Y:keep(X,Y)}.
```

Finally, the last rule defines the weak constraint ω, which guarantees that the witness (and thus the ancestor a) is not ignored, only if it has minimal size:

```
:~ distance(N). [1:N]
```

As previously stated, the answer sets of the program $\mu(G) \cup P$ enhanced with ω are representative of all the optimal common ancestors for (W,G). Moreover, the correct word senses are also encoded in the answer set of the program and can be easily retrieved.

3.5 Lexicon generation via entity set expansion

The solutions presented in the previous sections, that address the sub-problems of knowledge compilation and word sense disambiguation, are used in the process of automatic lexicon generation for a semantic annotator. In this section, we explain, how we use these notions to realize the entity set expansion task.

Note that, within the task, there are several sub-problems to solve, namely, to disambiguate word senses, to determine and formulate the target category, to expand the set by discovering new candidate instances, and to evaluate the candidates according to some measures. The first two are in fact variants of the problems described in previous section. As for the expansion, we resort to a new resource (in the sense that it was not used for category definition), WebIsADatabase which is a hypernymy database covering more and less "popular" instances.

Let us now explain the process in more details. First, let us assume a set of seed words W that we want to expand. We create an entity network for W that contain possible senses of the words (cf. Sect. 3.3.3, the first and second modules). If there are more possibilities of assigning senses to the words, we call the *sense detector* (cf. Sect. 3.4.2). If, as a result, we obtain a single set of word meaning appropriate in the context, we proceed to the next step. Otherwise, we communicate the ambiguity. The user can now select the intended meaning or add more seeds to clarify the intentions. In the latter case, for the enlarged set of seeds, we repeat the steps described so far (see a simplified example in Figure 3.2).

Once we know the single optimal combination of word senses, we proceed to the phase of *category recognition*. As we already said, we determine the target category by expanding a network in two directions: to determine the (set of) common ancestor(s) and the set of common relations.

For the first task, we expand the entity network for hypernyms (see Sect. 3.3.3, second module). Interestingly, while for sense disambiguation, we tend to go to WordNet hierarchy as soon as possible, due to the reliability and structure of WordNet taxonomy, when we look for actual common ancestors for already known senses, we explore BabelNet deeper — the reason for it is that the classes of BabelNet are more descriptive and human-readable.

Once we get the common ancestors, we expand the other relations (Sect. 3.3.3, third module). We collect the relations that are shared by all the seed units. For each shared relation, say r, we obtain a set U_r of units that are the *image* of the relation w.r.t. the seed units. If the set U_r is a singleton, say $U_r = \{u^*\}$, it means that the seed units are connected via the relation r to the same unit u^* and we can stop exploring the relation here. However, if it is not the case, then we may want either to discard this relation, or continue the analysis and determine the common category of units in U_r. In the latter case, we treat the U_r set of units as the new seed set, for which we repeat the process of finding a common ancestor

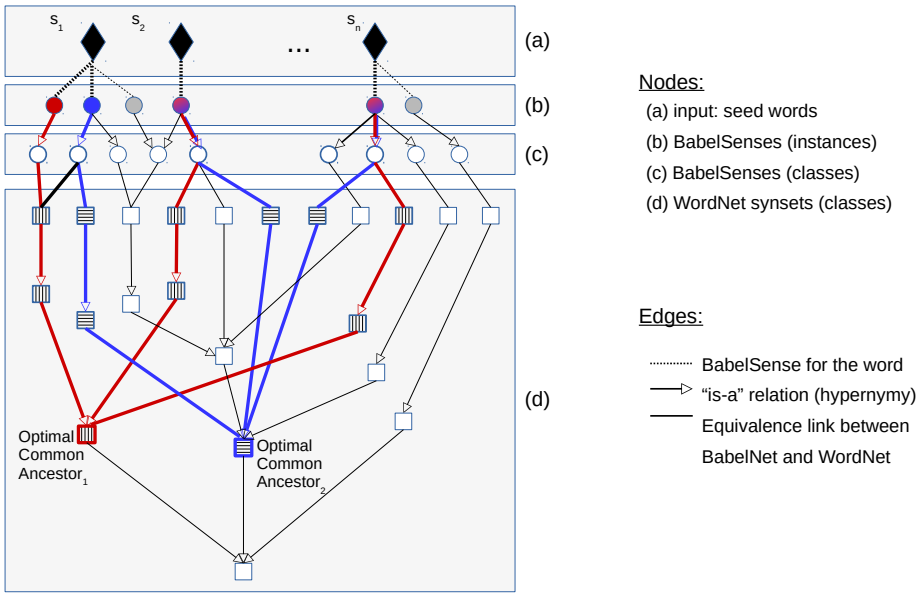


Figure 3.2: Creating an entity network from seeds and selecting the word senses

and analyzing common relations. To ensure termination, we fix the "depth" of the analysis by setting appropriate parameter. The result of the analysis is a set of properties describing the desired units.

To discover new object of the target category, we query WebIsADatabase for instances of the common ancestors of the seeds. We set a threshold to filter out noisely results (those that have too few witness pages). The obtained set of new candidate instances is then *evaluated* against the properties discovered earlier. In particular, we check if they are hyponyms of one of the desired common ancestors, and if they share the relations discovered for the seed set.

The results of the evaluation may be three-fold: (i) the instance can be found in reference semantic resources, in particular BabelNet, and in the entity network constructed for it all the properties agree (so it belongs to the target category), (ii) the network can be created, but not all the properties agree (it does not belong to the category), (iii) the network cannot be created for the candidate instance (an entry for it cannot be found in BabelNet). Recall that WebIsADb contains more of less popular instances than major semantic resources, so even if they are correct, they may not be evaluated positively by the reference resources. Thus, these candidates are presented to the user as not validated, but proposed.

Chapter 4

Document Layout Analysis

Although understanding the meaning of words and phrases is important, the additional information about the "context" in which they appear in a document may significantly improve the quality of extraction. If we know that for two pieces of text, one refers to the education and the other to employment history, then for the same semantic classes, say 'people' and 'institutions' recognized within them, the instances may have different functions and meanings, for example they can denote teachers and schools in one section, and employers and companies in the other. In this chapter, we present the problem of *document layout analysis*. We discuss the assumptions and objectives of the problem, review existing methods for recognizing document structure, and introduce our approach to the layout analysis, based on a notion of *labels* and *content*.

4.1 Problem specification

Document Structure and Layout Analysis [116, 86] is a problem of decomposing a document image into its component regions and understanding their functional roles and relationships. In other words, the field is concerned with geometrical and logical labeling of document content. Existing solutions usually focus on specific domains, such as academic papers [106] medical reports [112] and other collections of complex documents sharing some predetermined layout [105, 7, 8].

The analysis of layout and structure of a document depends of several factors: format of the document, assumptions and goals. To focus our attention, we select the Portable Document File (PDF) format, for its universality and popularity. Portable Document Format file format was developed by Adobe Systems in 1993 to represent documents independently of the hardware and software used to generate or view them. This format has now become an open standard included in the ISO (International Organization for Standardization) category and is widely

used by companies and individuals around the world to present and exchange electronic documents containing text, graphics, audio, 3D maps and many other content, reliably and securely regardless of the platform or device used. The ability to preserve and protect the content and layout of a document irrespective of the platform or software with which it is displayed, makes PDFs difficult to modify and often attempting to extract information from them can constitute a real challenge. This section presents the main features and problems of the task of recognizing two-dimensional or tabular structures from a PDF document.

The first aspect to consider is the encoding of the PDF format elements. In fact, each basic element (token, path, image) of a PDF document is contained in a rectangle whose size and position do not necessarily coincide with those of the object. This is an element of difficulty and a source of error in the process of object recognition within PDF documents.

Our objective of the document layout analysis is to obtain information about two-dimensional structure of "regions" designated within a document. The regions can be set to the whole page or to its parts, by selecting the regions manually or inferring themin a pre-processing phase. What we mean by a two-dimensional structure is in fact a table, that is a way to organize portions of text into *cells*, grouped into *rows* and *columns*. For the purpose of our task, we assume that any arbitrary combinations of document regions, even if they are not explicitly *tables*, can be represented as ones, and this representation can unify various structures and prepare them for further processing (note that tables may be irregular, or there may be multiple tables on a single page).

Moreover, as we consider *collections* of homogeneous documents, we are interested in fragments, related both to the content and structure, that appear repeatedly from one document to another. This includes typical "labels" i.e., phrases that — identically or in variants — appear in all or most of the documents in a collection, and have particular shared meaning. Our approach to document layout analysis consists in interleaving the strictly structural, two-dimensional analysis with the recognition of these specific labels. The combined information about the labels and (implicit) tabular structures will help us organize the documents along an abstract "grid" and, eventually, represent everything uniformly (with a model described in Chapter 5).

Finding a table structure in a PDF document may be relatively straightforward or very complex, depending on whether one considers regular and bordered tables or tables defined by paths in the documents, irregular or not bordered. In the case of regular or bordered tables, the difficulties are related, for example, to the clarification of cells that expand over multiple rows or columns or a combination of them. More complex is the task of recognizing unbounded or irregular tables that requires the definition of techniques for identifying areas of the document, in which the table is present, and the grid design that delimits it.

The process of recognizing tabular structures within PDF documents may be non-deterministic, especially in the case of irregular or non-bordered tables. It is easy to guess how important it is to validate the results obtained by applying algorithms for automatically recognizing tables. Such algorithms, in fact, make strong use of heuristic techniques, sometimes producing results that do not fully conform to the user's expectations.

4.2 Existing solutions

Even for the narrowed domain of recognizing tabular structures from PDF documents, there exist multiple solution proposals [118, 60]. In this section, we present selected tools that address different tasks of PDF file management. We critically evaluate them and point out which can be profitably used in our approach.

Nowadays, there is an increasing need for searching within PDF documents, extracting information, or converting entire documents into editable formats. Thus, it is desirable to have technologies and tools to search for, extract and reuse the information contained in PDF files. For these reasons, the market landscape of PDF file management products is particularly wide and varied. In fact, it is characterized by the proliferation of software, some available as online services, others in desktop versions, some specializing in more specific tasks such as creating, displaying, editing, or converting PDF files, and others that are configured as suites that integrate all the features needed for advanced PDF file management. Among the features for PDF files management, those related to extracting information from the files, is of particular scientific and practical relevance. An important part in this respect is played by a tool's ability to recognize two-dimensional structures (or tabular formats) and extract the information contained therein, to be able to handle it by another specialized software, such as Microsoft Word or Excel. This functionality is available on the market only within a small group of software systems. In this section, we illustrate distinctive features of market leading systems in converting PDF files to other formats, by comparing them with respect to the ways each of them can recognize and extract more or less complex tables.

4.2.1 PDF file management systems

The research revealed the following systems as the most comprehensive and competitive when it comes to PDF file management: ABBYY PDF Transformer+, Nitro Pro, Able2Extract Professional 10, PDF2XL and Quablo OCR. Let us introduce and shortly discuss all of them.

ABBYY PDF Transformer+ (http://www.abbyy.com/pdf-transformer/) is a PDF software that offers everything you need to work with PDF files: create, edit or comment on a PDF document, protect a PDF file with a password, discuss and collaborate with colleagues, convert or just read a PDF. It is a versatile and efficient program that offers an intuitive interface and various collaborative features. ABBYY PDF Transformer+ combines intelligent technologies with an easy to use interface to open any PDF document. One of the practical tools made available to simplify navigation in documents is bookmarks. Bookmarks can then be easily edited, linked and removed. Intuitive navigational navigation tools are the ones that allow you to navigate from one page to another, to zoom in or out, view the page, etc. With ABBYY PDF Transformer+ one can process and edit PDFs i.e., make small text changes like inserting or deleting words directly into a PDF file. Custom pages management (adding, deleting, and replacing pages, changing orientation, or creating an empty page to add background information) is also provided. ABBYY PDF Transformer+ makes it easy to create ISO-compliant PDF documents from Microsoft Word, Excel, PowerPoint, Visio, Apache OpenOffice, or any other printing application. In addition, one can reduce PDF file size by using MRC compression technology. The tool allows to quickly extract text and information from PDF files, including scanned ones. To reuse a document's content, one can copy the extracted text, tables and images, or convert the entire document to an editable electronic format such as Microsoft Office and Apache OpenOffice Writer, retaining the original layout and formatting.

Nitro Pro (https://www.gonitro.com/en/pro) offers individual users and large companies the tools to do their business intelligently and quickly. Nitro Pro offers a full range of easy-to-use comment and review tools that ensure accurate version control. One can add "sticky notes" to a document to express comments, instructions, annotations, markings, word excerpts, and text sections, using a set of tools compatible with Acrobat and other common programs. Nitro Pro allows to manipulate text, change fonts, customize layouts, and more, thanks to a complete and advanced set of PDF document editing capabilities. One can add, remove, replace and correct text and images, or edit the entire structure of a PDF file by adding and extracting and rotating individual pages. Lastly, thanks to optical character recognition (OCR), Nitro Pro allows to convert digitized documents and images into editable PDF files and to which searches can be made. Nitro Pro allows to create PDF and PDF/A files from 300 different file types. Created PDF files are compatible with industry standards and with Adobe Acrobat. PDF documents can also be created from scanned documents. Nitro Pro allows to easily convert PDF files to editable MS Office files, such as Word, Excel, and PowerPoint, while retaining original fonts, images, and formatting. One can also copy

and paste from a PDF file to Word and other MS Office files. Nitro Pro can also convert documents or a series of pages into separate pictures or PDF files. Finally, the tool allows to export entire PDF document collections into MS Office formats or image files in one operation.

Able2Extract Professional 10 (http://www.investintech.com/prod_a2e_pro.htm) is a PDF document conversion solution that effectively improves the productivity of those who work daily with PDF files on Windows, Mac, and Linux. The tool enables the conversion of PDF files to Word, Excel, PowerPoint, Publisher, AutoCAD and CSV formats by providing a rich set of content conversion and control options within an elegant and intuitive graphical interface that simplifies the entire process of converting PDF documents. With the inclusion of Optical Character Recognition (OCR) technology, Able2Extract Professional 10 can also convert scanned PDFs and images to Excel, Word, PowerPoint, AutoCAD, Publisher and more. Thanks to its powerful PDF manipulation capabilities, Able2Extract Professional 10, it makes it easy to edit PDF documents. Advanced editing capabilities allow to customize the font, color, and size of the text, make general changes to the PDF document pages, or act directly on metadata of the file or view preferences to better meet the needs of the users Of PDF documents. Able2Extract Professional 10 is a solution that also allows to create quality PDF documents from any application. Creating a PDF can be done in more than one way: one can open files directly in Able2Extract Professional 10 or use the virtual printer driver without opening the application. Able2Extract Professional 10 is widely known for its *proprietary conversion algorithm*, which is able to convert more complex PDFs to Excel, Word, PowerPoint, AutoCAD, HTML, CSV and more. The strength of Able2Extract Professional 10 consists in the ability to selectively select only the content of interest. Whether the goal is to get a formatted Excel sheet or an editable Word document, Able2Extract Professional 10 can deliver accurate conversion results thanks to a rich set of advanced and customizable conversion options. All PDF conversions can, in fact, be granularly customized by selecting a page, paragraph, or even a single line of conversions, in order to get Excel, Word or other perfectly formatted outputs that fit the user's needs. Able2Extract Professional 10 converts both native and scanned PDFs using proprietary OCR technology. The OCR engine can correct any microscopic error in the text and produce a fairly accurate digital version to be used for subsequent Business Intelligence analysis. Formatting, fonts, and colors are preserved more accurately than any other PDF file solution.

PDF2XL (https://www.cogniview.com/pdf2xl-international) allows to convert PDF files of any size and type, reliably, intuitively and economically.

To allow conversion of PDF files into multiple languages without any extra effort, PDF2XL integrates one of the best OCR technologies on the market, namely IRIS, a world leader in OCR solutions. As Microsoft's Silver Certified Partner and Development Partner of Adobe and thanks to the decade of experience in providing data conversion solutions, CogniView has managed to capture the trust of leading global companies and PDF2XL is used to convert an innumerable amount of documents each day. PDF2XL Enterprise allows to convert data to Excel format from any source of information quickly and accurately. The tool can convert scanned and image-based scanners to PDF documents as well, providing error-correction capabilities. PDF2XL Enterprise can process scanned documents in more than 30 languages. (it includes more than 30 OCR dictionaries to ensure that scanned text is accurately converted into the desired language). Conversion can also include image files such as JPG, BMP and PNG formats. Again, the conversion is supported by error correction tools. The ability to capture any file format in PDF2XL Enterprise is provided by the Virtual Printer of CogniView. The application of the "Single Page Structure" feature allows to convert PDF files of any size. The complexity of the file to be converted is also negligible for PDF2XL Enterprise: the tool is able to extract data from any type of file: documents with different tables on different pages, multiple tables on a single page, rotated page documents, documents with unreadable characters, with combinations of text and tables, etc. In particular, PDF2XL Enterprise allows to create "layouts" and use them for converting other similar PDF documents instantly. In addition to exporting data to Excel, PDF2XL Enterprise also supports other output file types such as CSV and Microsoft Word files. One can also copy data directly to clipboard. It also allows conversion of any kind of format and eliminates human errors with OCR support. It supports over 130 different languages. An additional advantage is the conversion speed (more than 500 pages per minute).

Quablo OCR (`http://www.quablo.eu/`) is a software developed by Exeura S.r.l. that supports automatic recognition and extraction of tables from PDF documents (originally produced or scanned) and images, with the ability to export the content in different formats. It provides several tools (also graphical) to deal with issues of extraction from PDF documents. Quablo OCR can automatically locate tables and export the data contained therein in different output formats. The tool's ability to automatically extract tables from PDF documents, even scanned, dramatically reduces transcription time, making the result reliable and instantaneous. Data extraction is accurate and accurate and limits the possibility of making mistakes. In addition, the graphical interface is practical and functional, allowing the use of software at the reach of everyone. Quablo has been designed to implement the flow of operations needed to locate and extract tables from data contained in

PDF documents. This stream can be divided into two phases: pre-processing, during which the information on tokens and paths contained in documents is extracted, and the actual processing that consists in exploiting that information to construct the tables. In order to ensure extensibility and ease of maintenance, Quablo is organized into modules, each of which is assigned a well-defined task. Each module receives the document to be processed and enriches it with the result of its processing (see Figure 4.1). The PDF document is captured by the system

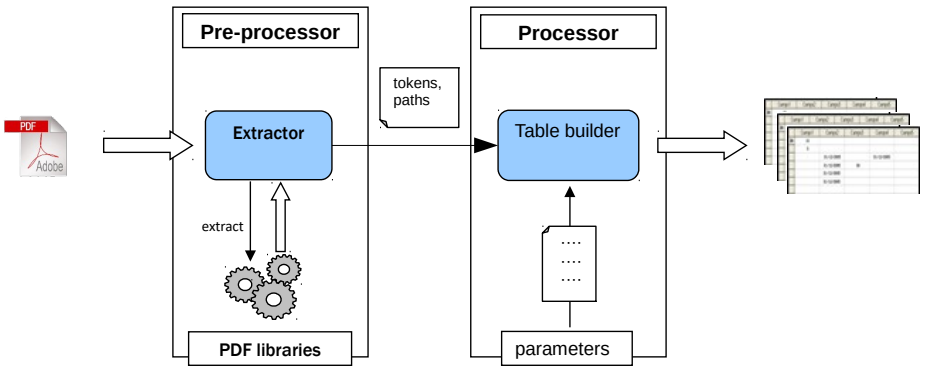


Figure 4.1: General architecture of Quablo tool

that deploys it to the extraction module, which uses the functionality of PDF libraries to extract the objects of interest – texts, paths and images – and collects them in appropriate data structures. The data structures constitute input into the table detection and creation module, which uses a set of parameters provided by the user. The output provided by Quablo is ready for any conversions or exports to other formats.

4.2.2 Table recognition challenge

As already mentioned, one of the features that are of particular importance in data extraction from PDF documents is the ability to recognize two-dimensional structures (or tabular formats) and convert them to other, editable formats, such as spreadsheets, for subsequent processing. In fact, this functionality can be applied and ensure a concrete increase in productivity in many operating environments. We now briefly discuss how this functionality is supported within the systems described in the previous section.

ABBYY PDF Transformer+ allows to copy and paste the tables from a PDF document while keeping formatting and structure intact. If necessary, one can change the separators manually before extracting the information.

Able2Extract Professional 10 allows to transfer tables of various formats directly to Microsoft Excel spreadsheets, MS Word, PowerPoint, and XPS. Noteworthy, the tool allows to save an export configuration from PDF to Excel and apply it whenever necessary, without having to redefine it from scratch. With Able2Extract Professional 10, therefore, information contained in even complex tabular structures can be converted without much effort in more usable data. OCR technology contained in the tool enables high quality conversions even for scanned documents with low resolution or image files.

Quablo OCR is a software solution specializing in extracting tables from PDF documents. The automatic recognition of the Quablo OCR tables algorithm provides accurate detection of tables with high complexity and even not bordered, but with regular structure. Tables without borders and too irregular to be automatically detected by Quablo OCR can be recognized interactively with a set of advanced, yet intuitive tools. Specifically, these tools allow to outline tables in a semi-automatic manner, or to draw, move and remove individual lines that identify rows and columns. In order to provide additional content selection tools, additional filters are available that only allow the recognition and export of tables that feature a specific structure or are positioned on specific pages of the document. Recognized tables can be easily navigated by the going between tables (next / previous recognized table) and, for each of them, a preview is provided in real-time which allows the user to verify the correct recognition of the data to be exported. One of the most important features of Quablo OCR is the ability to create and save *templates* for storing extraction schemes. Using templates is particularly useful when a user needs to extract data from structurally similar documents. In that case, by applying a pre-existing template to one or more PDF files, a user can avoid repetition of the steps needed to define an identical export configuration that was created in the past.

Nitro Pro and **PDF2XL** do not have the functionality for the recognition and export of individual tables from within PDF documents.

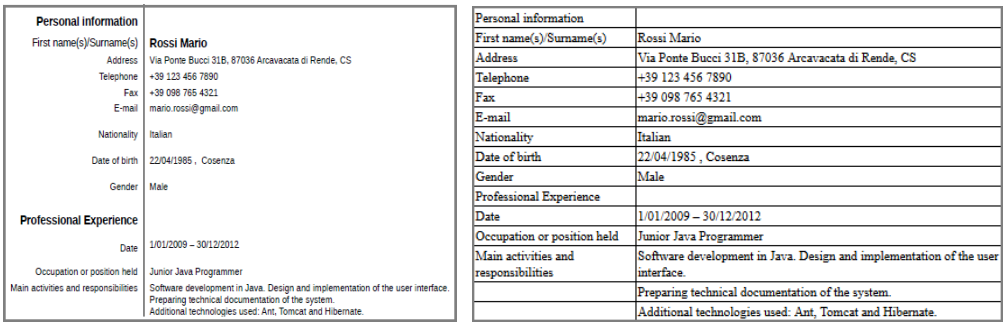
In the light of the analysis made so far, it is clear that although software solutions on the market, such as ABBYY PDF Transformer+ enable complete management of PDF documents, the strong specialization of Quablo OCR in recognition and the extraction of tables from PDF documents, and its advanced graphical and computational capabilities, make this software system the indisputable point of reference for anyone who needs to extract data, optionally starting from tabular structures, including very complex tables appearing inside PDF documents.

4.3 The label-content approach

Structural analysis of a document can significantly improve the subsequent phases of information extraction. In particular, taking into account two-dimensional elements (tables, columns, cells, etc.), allows to add some context to the document content. In the end, it is desirable to obtain a grid representation of the document. In order to achieve this objective, one can use a dedicated tool capable of detecting two-dimensional structural elements.

In this section, we present a method for document layout and structure analysis that we call a *label-content* approach. We explain what are labels and content in a collection of homogeneous documents, and we present a systematic method of analyzing and processing the documents, to obtain information about the layout and structure useful in further phases of Information Extraction.

We propose a two-step analysis mechanism. The first step involves using a two-dimensional processor that "blindly" analyzes the structure of a document in order to capture its two-dimensional aspects (see Figure 4.2). This output can



Personal information	
First name(s)/Surname(s)	**Rossi Mario**
Address	Via Ponte Bucci 31B, 87036 Arcavacata di Rende, CS
Telephone	+39 123 456 7890
Fax	+39 098 765 4321
E-mail	mario.rossi@gmail.com
Nationality	Italian
Date of birth	22/04/1985 , Cosenza
Gender	Male
Professional Experience	
Date	1/01/2009 – 30/12/2012
Occupation or position held	Junior Java Programmer
Main activities and responsibilities	Software development in Java. Design and implementation of the user interface. Preparing technical documentation of the system. Additional technologies used: Ant, Tomcat and Hibernate.

(a) CV in Europass format

Personal information	
First name(s)/Surname(s)	Rossi Mario
Address	Via Ponte Bucci 31B, 87036 Arcavacata di Rende, CS
Telephone	+39 123 456 7890
Fax	+39 098 765 4321
E-mail	mario.rossi@gmail.com
Nationality	Italian
Date of birth	22/04/1985 , Cosenza
Gender	Male
Professional Experience	
Date	1/01/2009 – 30/12/2012
Occupation or position held	Junior Java Programmer
Main activities and responsibilities	Software development in Java. Design and implementation of the user interface.
	Preparing technical documentation of the system.
	Additional technologies used: Ant, Tomcat and Hibernate.

(b) Table drawn by Quablo

Figure 4.2: Recognizing tabular structures in a PDF file

then be rearranged in accordance with domain knowledge defined by *labels*. In other words, starting from a grid structure that divides the document into cells, it is useful to proceed to a second phase of structural analysis, which takes into account semantic information in order to obtain a qualitatively better grid and more useful to subsequent extraction phases.

If we assume that we work with a set of documents to some extent related to a given template, i.e., they share some layout features, typical sections, keywords, and recurring patterns, we can exploit them to improve the process of building an abstract two-dimensional grid. For example, by analyzing a CV in Europass format, one can assume a two-column structure, where labels belonging to a suitable list of labels appear on the left, and on the right had side, there are values for each of the property denoted by the labels on the left. Moreover, we know that these

documents have specific sections such as 'Personal Information', 'Work Experience', 'Education', 'Skills', etc. These are the domain-dependent concepts that can be used for a more accurate extraction of two-dimensional elements from the input document.

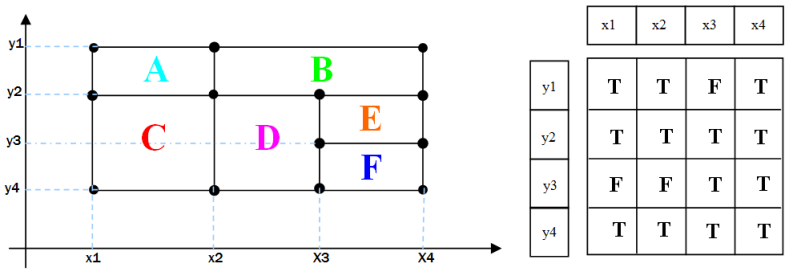
From the available solutions, we have selected the Quablo tool, suitably extended for the optimization of the abstract grid construction process. In this way, the two-dimensional processing module outputs a grid structure where cells could be unified (for example, if a label logically encloses more than one cell, these cells should be merged). From the end user point of view, the grid construction mechanism is completely transparent. One could even use another tool instead of Quablo, given that the tool provides the required functionality.

4.3.1 Recognizing the document structure

Finding a table in a PDF document can be more or less complex, depending on the table's characteristics. In particular, the recognition of bordered tables, which are defined by the grids formed by the graphical paths contained in the documents, is quite straightforward, while the one of irregular or non-bordered tables is very complex. The implementation of table recognition in Quablo and its extension to adopt the label-content technique was carried out by Exeura S.r.l who was a partner in the KnowRex research project.

Recognition of regular and bordered tables In the heart of the table recognition and extraction process, there is a *grid reconstruction algorithm*. The introduction of this algorithm not only dramatically improves precision in the recognition and segmentation of tables (to almost 98%, as reported by Quablo developers), but it is also best suited to resolve some difficult issues such as the identification of cells spanned over multiple rows or columns. The algorithm works on the segments that make up the table, identified during the extraction phase; this phase, in fact, provides the graphical objects that are first converted into straight lines (and described by the relative equation) and then subdivided by orientation (the horizontal lines are distinguished from the vertical ones). The lines are then compared to each other in order to identify all the intersection points that are collected in a suitable data structure and passed to the indexing phase. The purpose of this phase is to construct a matrix of boolean values ('true' / 'false'), as shown in Figure 4.3. An element of the matrix is set to 'true', if at the coordinate point $(row, column)$ in a document there is an intersection, and 'false' otherwise. Note that the r and c coordinates are used respectively to access the array dimensions y and x.

Starting from the element in position $(0,0)$, the algorithm looks for the cells as follows: first, if there is a 'true' value in the considered point, the next 'true' value



	x1	x2	x3	x4
y1	T	T	F	T
y2	T	T	T	T
y3	F	F	T	T
y4	T	T	T	T

Figure 4.3: Matrix representation of a table in a PDF document

is searched for "in the right direction" (navigating the matrix appropriately). Once the value is found, the search proceeds "downwards" to identify a pair of 'true' values that, together with the previous two, denote the coordinates of a single *cell* of a table. The procedure is then repeated for subsequent positions.

Recognition of irregular or non-bordered tables For the recognition of irregular tables, a heuristic-based algorithm has been developed, that consists of the following:

1. *Localization of document areas* that contain non-bordered tables.
2. *Table construction* (which is the most delicate step of the algorithm).
3. *Building a grid* that delimits the table.

In order to *localize the areas of the document* that contain non-bordered tables, the PDF document is divided into *zones* of two types: the zones containing text and those containing tables. The detection of these zones is done by incrementally grouping text lines recognized in the document. Subsequently, the properties of the lines are evaluated, using heuristics, to establish the zones. A text line is considered as belonging to a table, if it contains a white space between one word and another that exceeds a predetermined threshold. A zone is labeled as a candidate to contain a table or not, depending on the number of lines that meet the previous condition.

Building a table is the most delicate step of the algorithm. The initial cells are constructed from recognized tokens. Then, the cells are organized into columns. At this stage, they may be merged. Finally, columns are subdivided into rows to form the final table. The tables recognized in this way are then filtered, to discard the degenerated ones and the ones that do not satisfy the constraints that express the minimum number of rows and columns in a table.

For the *grid construction*, the tables constructed in the previous step are stored with use of a matrix of objects representing the cells of the tables. These objects can be stored in multiple adjacent rows and/or columns of the array, so that they can represent cells that occupy multiple rows or columns of the table. Based on this representation, it is then possible to build a grid that encloses the table, from the matrix that represents it. This is achieved by surrounding the table cells with segments and adding the lines representing the outer edges of the table.

4.3.2 Improving the recognition with domain labels

The idea of improving the two-dimensional structure of a document with a set of domain-dependent characteristics, consists in interleaving the strictly structural analysis with some semantic annotation of content. Recognizing a domain concept in a particular place can in fact improve the outline of the recognized table, repair some imperfections and, by merging or aligning cells, prepare the "grid" representation for more efficient processing. This phase is realized by the selected processor extended with new functionality in the recognition algorithm.

To start with, the domain-specific labels are defined in a *label dictionary* with strings and regular expressions, e.g.:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<entities case-sensitive="no">
    <entity type="field" name="eu_cv_pinf_label">
        <value type="str"> Informazioni personali </value>
        <value type="str"> Personal information </value>
        <value type="regexp"> Personal inf.* </value>
    </entity>
    ...
<entities>
```

This dictionary is used in the recognition phase, to improve the two-dimensional grid representation of the document. The behavior of the algorithm that searches for the labels within a tabular structure depends on the documents characteristics. In particular, the orientation of the virtual table is set (horizontal or vertical), and the column (or row) in which domain labels are expected to appear is fixed. To focus our attention, we depict in Figure 4.4 the case of a two-column layout, where in the first column, there should appear the domain labels, and in the second – the values of the properties denoted by the labels. Then the algorithm proceeds as follows: for the selected "label" column (resp., row), it tries to match the content with the label dictionary entries. As labels may span across several rows (resp., columns), the algorithm tries to take into account the combined content of the analyzed and following rows (resp., columns) up to a fixed threshold. If a match is

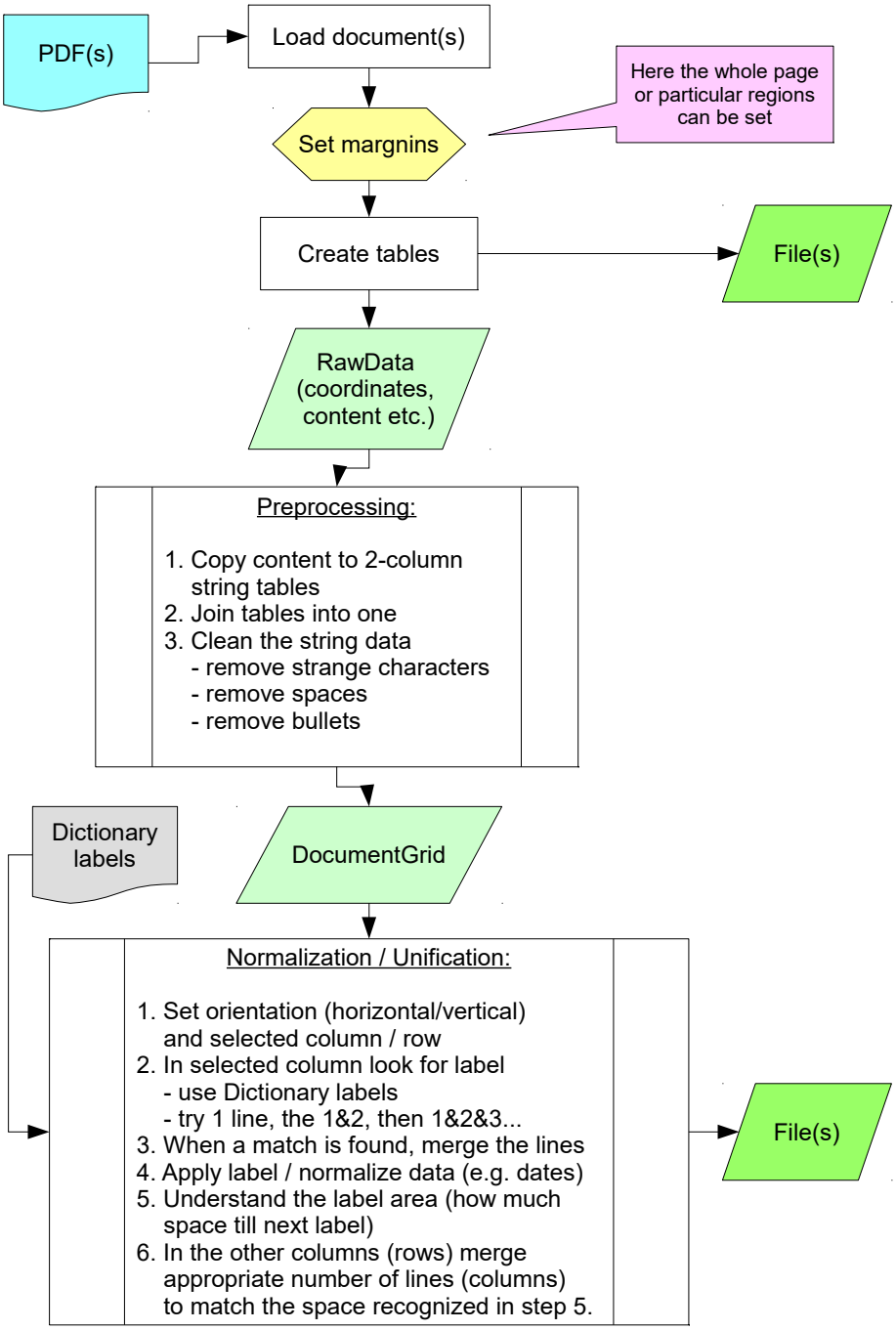
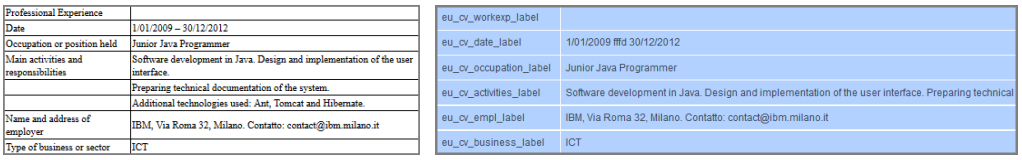


Figure 4.4: Transformation of a PDF document with table and label recognition

found, the cells are merged and the content is annotated with the recognized label. At this stage, some post-processing can be done. [1] The algorithm then proceeds to calculate the vertical (resp., horizontal) span of the area related to the recognized label. Having this value, in the other columns (resp., rows) it merges appropriate number of rows (resp., columns) into one cell and by doing this it aligns the values, potentially spanned across rows (resp., columns) to the particular heading. In the end, we obtain a more meaningful representation of the content within the tabular structure (see Figure 4.5).



Professional Experience	
Date	1/01/2009 – 30/12/2012
Occupation or position held	Junior Java Programmer
Main activities and responsibilities	Software development in Java. Design and implementation of the user interface.
	Preparing technical documentation of the system.
	Additional technologies used: Ant, Tomcat and Hibernate.
Name and address of employer	IBM, Via Roma 32, Milano. Contatto: contact@ibm.milano.it
Type of business or sector	ICT

(a) Raw tabular structure

eu_cv_workexp_label	
eu_cv_date_label	1/01/2009 fffd 30/12/2012
eu_cv_occupation_label	Junior Java Programmer
eu_cv_activities_label	Software development in Java. Design and implementation of the user interface. Preparing technical
eu_cv_empl_label	IBM, Via Roma 32, Milano. Contatto: contact@ibm.milano.it
eu_cv_business_label	ICT

(b) Annotated grid representation

Figure 4.5: Constructing a two-dimensional representation of a document

The result of the entire two-dimensional recognition and extraction phase provides a model of a document that also contains information about the positions of the one- and two-dimensional objects contained therein. The obtained grid representation is a base for further IE steps. In particular, it allow to formulate extraction rules that take into account the spatial context of text portions e.g., *after label X, there is the object Y that I need*, but at the same time are not "wired" to physical encoding of particular file, because the actual layout has been "normalized" into a grid during the layout analysis process.

[1] For instance, in the CV use case, we applied a post-processing for recognized dates such that their format is normalized and if a range of dates is given, then the number of months between the start date and the end date is calculated. Such calculations were useful to understand the duration of particular work experience.

Chapter 5

Ontological Document Representation

In order to integrate knowledge about the structure, layout and content of a document, we either need a common knowledge representation (KR) method, or knowledge processing such that it will integrate different aspects during reasoning. Ontologies [50, 53] constitute an approach to knowledge representation that is profitably used for conceptualization of heterogeneous knowledge and can be used a skeleton for different applications. In this chapter, we analyze the possibilities of unified representation of document, covering aspects of its layout, structure and content. Such holistic approach will enable us, on the one hand, to capture all the analyzed aspects and, on the other, allow to process it with the same methods. In the following sections, we review suitable knowledge representation methods with focus on ontological languages, then propose a model that captures the content, structure and layout of a document. We illustrate our proposal with an example of an ontology for specific homogeneous documents collection.

5.1 Review of knowledge representation formalisms

In this section, we review knowledge representation formalisms, and in particular we analyze various ontological languages [52, 90]. In recent years, knowledge-based formalization and sharing activities, indispensable to enable semantic information management, are attracting more and more attention, especially in distributed environments such as the Web. To represent knowledge and make it machine-manipulable, some formalism is required, that allows the information to be treated in a standardized way, and makes possible automated reasoning. The choice of formalism is usually a compromise between two opposing needs: the expressive power of the formalism and complexity of reasoning in it. Below we

present a brief list of relevant formal languages for knowledge representation.

5.1.1 Ontological languages

Ontological languages express a conceptualization of the domain of interest with classes of objects. The languages vary in their formality levels, resulting in variety of representations, from informal lexicons, through semi-formal thesauri, up to formal knowledge bases. For our purpose, we are interested in languages rooted in formal logic, to utilize the possibilities of automatic manipulation of knowledge expressed with them. Let us first discuss selected languages based on expressive formalism of *First Order Logic*.

KIF (Knowledge Interchange Format [47]) is a formal language designed to ensure interoperability and exchange of knowledge formats. It is very expressive; also, the meaning of its expressions can be understood even without a specialized interpreter. Its semantics is given by the rules of First Order Logic (FOL). The main objectives and features of the language can be summarized in three points: translatability (it is easy to implement translation mechanisms from and to a particular language with this representation of knowledge); readability (KIF's syntax makes it comprehensible to human); and the implementation capability.

CycL [80] is also based on the FOL. It allows to distinguish knowledge related to a concept from knowledge related to the terms that define it. It is possible to use higher-level logic elements such as quantification of predicates, functions, or axioms. CycL allows reification and reflection, i.e., provides the means to make statements about other statements or assertions about the process for creating the assertion itself. Moreover, it is based on models,so it is possible to discuss desires, expectations and not just statements. It also allows for default reasoning. Furthermore, it is possible to use operators for negation, conjunction, and disjunction to represent the semantics of terminology used in the language to express the rest of knowledge. With the CycL language, the entire Knowledge Base is defined and divided into micro-theories i.e., collections of concepts and links, typically concerned about the same subject, which must be contradictions-free.

First Order Logic is an expressive formalism that is undecidable in general case. Its syntax is oriented towards statements and it is not very intuitive, especially for non-experts. Thus, other formalisms have been proposed. Among them, there are *frames* whose main intuition lays in representing objects that have certain characteristics.

Frame-based languages use frames, originally introduced by Marvin Minsky, as the main tool for representing knowledge. A frame is a structure to represent a concept or a situation. Frames are supposed to capture the essence of concepts or a stereotyped situation, by aggregating all the relevant information for that concept. Frame collections can be organized in systems where frames are interconnected. The processors that work on such systems are supposed to match each frame to a specific situation, to use default values to fill unspecified aspects, and so on. Minsky explicitly wanted to leave out the idea of what a frame should do. However, the frame concept evolved over time, and now typically a specific meaning in the context of the representation of knowledge is assumed.

There are several common features for frame-based languages. First, the frames are organized into hierarchies. Second, they have slots (attributes) for which values (scalar values, references to other frames or procedures) must be specified or calculated. Finally, the properties (values, value restrictions, etc.) are inherited from super-frames to sub-frames in the hierarchy in accordance with some inheritance strategy. These organizational principles have proved to be very useful and many object-oriented languages have adopted them. From a formal point of view, semantics of frames and inheritance were only given operationally. Though the frames offer nothing new to the logic of the first order in terms of expressiveness, they offer a more concise way of expressing knowledge in the style of object-oriented languages. Moreover, by using only a fragment of the first order logic, they can offer more efficient means for reasoning.

Ontolingua [39] developed at Stanford University, is a language based on Knowledge Interchange Format (KIF). It provides an environment for navigating, creating, editing, modifying and using ontologies. It combines frame paradigms with first order predicates. It can represent concepts, taxonomies of concepts, n-aries relationships, axioms, instances and procedures. Unfortunately, it does not allow automatic reasoning.

Loom [78] was developed at the Information Science Institute (ISI) at the University of California. It is a language and environment for building intelligent applications. Loom's heart is a system of knowledge representation that is used to provide deductive support for the declarative part of Loom's language. Declarative knowledge in Loom consists of definitions, rules, facts, and default rules. Loom implements a series of KR functions. A deductive engine (also called a classifier) uses inductive techniques, semantic unification and object-oriented truth maintenance technologies for declarative representation of knowledge in a structured system to efficiently support the execution of deductable queries.

F-Logic or Frame Logic [70] was developed at Karlsruhe University in Germany. It is a logical formalism that seeks to capture the features of object-oriented approaches for computing and rendering data. It aims to bridge the gap between object-oriented object-based calculation and representation systems and deductive databases by providing theoretical base for a logic-oriented programming language that can be used both as a computational as well as representation tool. The name of the language makes reference to the frame-based languages. F-Logic supports complex objects, inheritance, polymorphism and encapsulation. The basic constructs of language are terms, consisting of function symbols, constants and variables. The formulas are constructed from the terms using a set of connectors. Language elements are meant to represent object properties and relationships between objects. Terms are used to identify objects and to access their properties. Function symbols play the role of object builders. Formulas allow to build object assertions. The main weakness of F-Logic is related to the necessary familiarity with logical and mathematical concepts needed to program in the language. The strengths of F-Logic are its extensibility and ability to represent concepts, either from object-oriented programming or from frame-based languages. The main application of F-logic is the Flora-2 system. Flora-2 integrates F-Logic with other formalisms, such as HiLog (a logical formalism that allows higher-order and meta-programming features in a computationally tractable setting) and Transaction Logic (which provides a logical foundation for state changes and side effects in a programming language).

Important descendants of the frame-based approach are the *Description Logics* [10] that capture the declarative part of the frames using a semantics based on FOL. Description Logics is a family of knowledge representation languages that deals with classes of objects, individual objects, and relations among them with use of logical *descriptions*. The basic elements of the DL languages are *atomic concepts* denoting classes, *atomic roles* for expressing (binary) relations, and *individuals* to denote instances. More complex descriptions are constructed iteratively by applying a set of *constructors*. In fact, the allowed set of constructors distinguish one DL language from another. Description Logics semantics is based on First Order Logic, restricted to unary and binary predicates. Complexity of reasoning have been studied separately for different variants of DL, as even a small alternation in the expressiveness of the language (addition or modification of allowed constructors) strongly influences the complexity of reasoning in this language (see a complexity "navigator" for different DL languages at `http://www.cs.man.ac.uk/~ezolin/dl/`). Although research on Description Logics has been conducted for a few decades, its most popular application is probably the Semantic Web ontology language OWL, described next.

OWL (Web Ontology Language [108]) is a W3C standard for representng ontologies for the Semantic Web. It can represent information about object categories (classes) and can describe their correlations. Following the DL approach, OWL describes the classes by specifying their properties. OWL has the ability to describe the various domains and ranges of properties, defining the former as OWL classes, while the ranges can be both defined with OWL classes and with externally defined data types (for example, string or integer). In OWL, one can impose restrictions on the class properties. For instance, a property can be limited so that its values belong to a given class only or are of a certain type, as defined with universal or existential quantification or by specifying their cardinality. To limit the descriptions to the ones significant only to a specific class, one can arrange the classes into a hierarchy with sub-classes and super-classes, and also provide a hierarchical schema for the properties. Another feature of OWL is the ability to declare the properties *transitive*, *symmetrical*, *functional* or *inverse* to other properties, or define for a pair of classes or properties if they are disjoint or equivalent. Finally, OWL represents individual information (property and class instances), each of which is assigned an identifier in the form of an URI. Individuals are assigned to a class and tied to other instances through properties e.g., by providing information about property values, if they are specific objects, and whether two objects are equal or distinct. OWL is based on classical logic and therefore adopts different semantic assumptions, such as Open World Assumption (OWA) or Multiple Names Assumption. It is a sophisticated language and its definition has had various influences including Description Logics, frame-based formalisms and the vision of the Semantic Web with standards such as XML and RDF. Given the diversified needs for OWL and the difficulty of satisfying them all, different versions of language have been created. OWL DL is called so, because it is based on descriptive logic, in particular, $\mathcal{SHOIN}(D)$ DL, and extension of $\mathcal{SH}$ logic with datatypes. It is characterized by a frame-like syntax and has been designed to support the industry of existing descriptive logic and has specific computational properties for reasoning systems. This version guarantees computational completeness and decisiveness. OWL Lite is a subset of OWL DL corresponding to logic $\mathcal{SHIF}(D)$. It allows only a limited number of constructions, for example it supports cardinality constraints but only allows values of 0 or 1. OWL Full, allows, unlike OWL DL and OWL Lite, that a class can be treated simultaneously as a collection of individuals and as a single object. This expressive freedom, however, pays the price of having less guarantees at the computational level. In the new version of OWL, namely OWL 2, three variants have been identified: OWL 2 EL, OWL 2 QL and OWL 2 RL – each tailored towards different kind of applications in mind and different requirements for the ontologies. Generally, the EL variant is oriented towards ontologies with complicated sub-class hierarchy (such as in medicine), QL is especially suited for ontology-based data access, and

RL supports limited expression of rules. What is important, all of them ensure tractable reasoning.

Description Logics and First Order Logic share the *Open World Assumption* which means that even when one cannot prove something to be true, based on the possessed knowledge, it cannot be assumed to be false. This is often counter-intuitive in real-world applications. Conversely, in the logic programming and databases communities, the prevailing assumption is that of a *Closed World* (if something is absent from a knowledge- or database, it is assumed to be false). There have been attempts to build "bridges" between ontologies and logic programming by either strict or loose integration of the underlying formalisms. The objective of such endeavours is usually to combine the advantages and possibilites of both approaches, or to be able to integrate new features into existing systems and tools. Some of the proposals are presented below.

AL-Log [35] is a knowledge representation system based on description and deductive logic with the support of the Datalog language. AL-log can be seen as a combination of two subsystems:

- *structural*, which allows the definition of knowledge in a structural way through concepts (classes) and relations between objects and classes; the language used to express the concepts is $\mathcal{ALC}$ (DL Attribute Language with Complements). It is structured in two parts, intentional and extensional. The intentional part describes the concepts of interest and defines their properties, while the extensional level allows to define the instances of declared concepts.

- *relational* that allows to define the connections between the described objects and the structural components. The language used is Datalog, a query language that allows to define queries in a declarative way.

The two components interact with each other, allowing to add to the deduction process of Datalog particular deduction steps from the concepts declared in $\mathcal{ALC}$. Al-log is highly expressive, but in some respects it is weak, for example it does not treat in a convincing way the aspect of negation in the relational sub-system and therefore has shortcomings caused by ignoring some knowledge of the structural subsystem.

Complex [48] extends Datalog with object-oriented constructions. It supports normal (non-disjoint) stratified programs. There are three main aspects of the

system. First, it supports a logic-based language, called C-Datalog (Complex-Datalog) and enriches this language with semantic constructs for the representation and manipulation of complex objects. C-Datalog supports two types of entities: the classes identified by an object identifier and organized within a hierarchical structure, and relations that describe the relationships between the declared classes. Second, it is a system that guarantees the accuracy and completeness of the inference, based on a bottom-up computation model, which efficiently supports the declarative semantics of language. Finally, it supports running queries using a top-down assessment strategy.

5.1.2 Desired characteristics of the formalism

In order to develop a universal yet functional model for representing a document (including perspectives of both structure and content), we have formulated a few requirements that the candidate language has to meet.

First, we would like it to be intuitive; more precisely, it should follow the object-oriented modeling paradigm. Moreover, it must be based on formal logic, so that automatic manipulation of the represented knowledge is possible. Ideally, there exist established reasoners that would support the selected language.

The language of choice must use a set of abstraction mechanisms to describe the structure of knowledge related to a document. In particular, an indispensable element for modeling any domain of knowledge is that related to the *class* or *entity* concept that can be used to describe an object, with its attributes or properties. Entities can be of a different type and each entity would belongs to a type that specifies its nature. In addition, we want to define *relationships* or *associations* between entities, as well as introduce individuals (or instances) and hierarchies of classes. In fact, it is often useful to organize the entities into a hierarchy of specialization / generalization. A class in a hierarchy lower than others is called a sub-class (and the one higher – a super-class).

5.1.3 The language of choice

The formalism we chose for our approach is inspired by the OntoDLP [96] language, which is based on logic programming and is particularly suited to design and reason on ontological knowledge bases. This choice is in accordance with our need for an ontological formalism. In this context, the OntoDLP language will be used to represent classes, relationships, and instances. The language is designed for ontological representation, and also provides powerful instruments of reasoning. It is based on disjoint logic programming (DLP) enriched with object-oriented programming concepts. Reasoning takes place with deductive mechanisms given by logical formalisms on object-oriented structures.

Classes A class is the primary notion in modeling any domain of interest. For example, assuming we want to model a banking domain, we can identify at least six classes: bank, account, branch, place, business, person.

```
class bank.
class account.
class office.
class place.
class enterprise.
class person.
```

In OntoDLP, a class statement requires the use of the 'class' keyword before the class name. In addition to the class name, one can specify attributes, that is, the characterizing properties of the objects of this class. Each attribute is defined by the pair *<attribute_name>: <attribute_type>* indicating, respectively, the attribute name and its type. The type of an attribute can be: text (string), numeric (integer), or object, that is, a type composed from other objects. For example, if you want to define more accurately the classes listed above, you could add detailed information such as:

```
class bank (name: string, asset: integer).
class account (balance: integer).
class office (bank: bank, address: place, asset: integer).
class place (name: string).
class enterprise (name: name, country: place).
class person (name: string, age: integer, father: person,
              mother: person, residence: place).
```

Objects A domain of knowledge is represented by "populating" the schema with objects (or instances). Each object belongs to a class and is uniquely identified by a property called object identifier (*oid*). In particular, it is possible to represent instances with logical facts that can be expressed as follows:

```
'PAPERBANK': bank (name: 'Paperopoly Bank', asset: integer).
```

Note that in this statement, the object identifier (in this case it is `'PAPERBANK'`) is specified, and the values for all the properties are set. The instance identifier will be used to refer to the object; This type of usage occurs, for example, in creating an 'office' class object for which complex typology types are present:

```
'fil01': office(bank:'PAPERBANK', address:'Gardens Avenue 10',
                asset:1000000).
```

In this case, `fil01` is the object identifier of the instance, while `PAPERBANK` is the object identifier of the bank attribute.

Taxonomies In object-oriented programming languages there is an abstraction mechanism called inheritance that allows to specialize or generalize classes. In OntoDLP this mechanism is realized by the binary relation *isa*, by which it is possible to indicate the subtype-supertype relationship between two classes. As for an example, different categories of people, such as clients or employees, are characterized by a number of attributes, and can be represented easily by defining appropriate sub-classes of a class 'person'.

```
class employee isa person (office: office, skill: string,
                           mentor: employee).
class client isa person (clientcode: string).
```

In this modeling, the class 'person' corresponds to a more general concept of 'client' and 'employee', which are, instead, specializations or subclasses of the class 'person'. In particular, an employee, in addition to attributes that are inherited from the class person, will have an office, a skill, and a mentor (called local attributes). A client instead has an attribute 'client code' which is a string of text. An important (and useful) consequence of these statements is that every employee instance will automatically be considered as an instance of the class person (the opposite, of course, is not valid). OntoDLP supports multiple inheritance, so it is possible to define an unlimited number of sub-classes and super-classes. For example, consider a possible customer-employee class that must contain all attributes in the client class and employee: this class could be defined as a specialization of the above classes. To do this, one needs to add both classes in the superclass statement:

```
class client-employee isa {client, employee}.
```

Based on this statement, the client class will have a set of attributes from the union of both super-classes.

Relations A fundamental feature of the knowledge representation languages is the ability to express relationships between objects in a domain. OntoDLP allows to declare object relationships using a class-like syntax: the 'relation' keyword is followed by the name of the relation and a list of its attributes. For example, 'hasOwner' and 'hasWorked' relations contain information about the accounts and their owners, and the employees and offices in which they worked. The two relations can be declared as follows:

```
relation hasOwner (account: account, owner: client).
relation hasWorked (empl: employee, office: office,
                    start-date: date, end-date: date).
```

The set of attributes of a relationship is called a *schema*, while the cardinality of it is called the *arity* of the relation. The schema of a relation defines the structure of its instances (tuples). The tuples of a relations, as with class instances, are defined by claiming appropriate logical facts. An example of a tuple is as follows:

```
hasWorked (empl: 'emp0983', office: 'fil06',
           start-date: '2006-06-03', end-date: '2009-06-21').
```

which states that the person with object identifier `'imp0983'` has worked in the `'fil06'` office from 3/06/2006 to 21/06/2009. Unlike class instances, however, the relations tuples have no *oid* (identifiers), and are therefore not directly referable. As with the classes, it is possible to represent specialization / generalization links by defining taxonomies also in relation to the relationships. For example, suppose we want to define a relationship that associates a person with an identification code. In OntoDLP we can write:

```
relation hasCodeIdentifier (p: person, code: string).
```

This relation can be further specialized if you consider different codes that can identify a person, such as the identity card number, the driving license number, tax code, etc. As a results, we obtain a taxonomy of relation:

```
relation hasDrivingLicenseId isa {hasCodeIdentifier}
  (issueDate: string, validity: string).
relation hasIdentityCardNumber isa {hasCodeIdentifier}
  (issueDate: string, validity: string, country: string).
```

As is the case for the classes, each tuple of the sub-relation possesses implicitly the definition of a tuple of the super-relation (and the opposite is not true).

Logical rules and complex terms On OnDLP you can specify logical rules. A rule is expressed, for example, in the form:

```
likes(p1:X, p2:Y) v hates(p1:X, p2:Y)
                  :- colleague(p1: X, p2: Y), X <> Y.
```

The rule's *head* consists of the `likes(p1:X, p2:Y) v hates(p1:X, p2:Y)` disjunction and is followed by the conjunction: `colleague(p1:X, p2:Y), X <> Y` which constitutes in this case the body of the rule. The intuitive meaning of this rule is: if X and Y are colleagues, X either likes or hates Y. OntoDLP rules support the use of "complex" predicates that allow to access classes and relations in an ontology. More in detail, OntoDLP rules are made up of two different types of predicates: *class predicates* and *relation predicates*. Using a class predicate: `oid:class(att_1,....,att_n)` where 'oid' is a variable or instance name and 'att_i', where $1 \leq i \leq n$ are pairs (attribute name, attribute value), you can access instances of a class. For example, `P: person(name: 'Giovanna')` is a class predicate where the oid is represented by the variable P and the attribute 'name' assumes the value 'Giovanna'. If a class has multiple attributes, it is not required that all of them are specified, but only those who are interested. Also, they

can be specified in any order. This kind of notation, in which the programmer selects the only attributes they are interested in by their name is called *non-positional notation*. OntoDLP allows to specify complex objects (i.e., objects that have object type attributes). Finally, in order to get access to the values of the attributes of complex objects, it is possible to make use of so-called complex terms. For example, if we wanted to indicate with *X* the name of the mentor of an employee *Y* we could write: `Y: employee (mentor: person (name: X))`. This is possible because the 'mentor' attribute is of a type 'person' and the language allows to directly access its attributes. By means of a relation predicate, it is possible to "navigate" the objects through their associations. This also allows to combine the declarative style of logic programming with the one of object-oriented systems.

Lists OntoDLP features also *lists* as a possible type, an ordered collection of instances that accept copies of the same instance. Given a class *C*, one can define the C-list class type, defined as $[C]$ and composed of instances (individuals) belonging to class *C*. For example, the *[string]* class represents the list of Strings. Likewise, *['This', 'That']* is the list that contains the strings *'This'* followed by *'That'*. The list construct is particularly useful for representing multi-valued attributes. Suppose, for example, that in an account domain (discussed in the example in this section) an account has associated a number of services that can be purchased in the bank (access to online services, credit card, etc.). A bank account can be represented as:

```
class account_service (cost: int).
internet: account_service (cost: 5).
credit_card: servizio_conto (cost: 10).
...
```

The definition of the bank account class can now be extended by enriching it with the new attribute for available services, such as "list of services":

```
class account(services:[account_service], budget:int).
a0001:account(services:[internet, credit_card], budget:2000).
```

where the second is a bank account instance that has two associated services, internet and a credit card.

Queries An important feature of the OntoDLP language is supporting *queries* to extract information from a knowledge base. A query on OntoDLP is a conjunction of atoms. For example, one can request the list of employees whose mentor was born in Rome by writing the following query:

```
X:employee(mentor:P), P:person(place:L), L:place(name:'Roma')?
```

In general, one can use both predicates of class and relations. Note that one can specify only the attributes whose values are important, and omit attributes that are not relevant to the specific query, obtaining a more compact specification.

5.2 Principles of the proposed model

The proposed ontology group classes grouped under two main categories: *layout objects* and *content objects*. The layout objects in the ontology define the structural elements of a document (such as cells, text fragment, tokens and delimiters), independently of a domain of interest. The user of a system that uses the ontology does not to alter it, add any classes etc. for the system to work properly. The layout objects, when it comes to the Information Extraction process, should be identified by one- and two-dimensional tokenizers during the phase of bidimensional document analysis. The content objects, in turn, cover the objects located within the content of a document. Among the content objects, we further distinguish:

1. *category markers* – that are parts of the document that identify particular sections and subsections point to some interesting or recurring elements within a collection of homogeneous data etc. They are not interesting themselves from the point of view of the target database. However, they play an important role in the extraction process by identifying pieces of document where the actual information to extract is. For instance, for a Curriculum Vitae, these are section and subsection labels, such as: 'Personal Information', 'Work Experience', 'Education' etc. For another use case of Business Process Documentation, these include e.g., 'Process purpose', 'Process Scope', 'Process Input' parts, and more similar to them.

2. *relevant objects* – that represent the actual content to be extracted from a document. Here, we further distinguished:

    - *basic objects*: which are common categories such as "email", "date" etc. that do not belong to a single domain. We have identified them as recurring in different domains, and thus potentially useful for various use cases. This part of the ontology is given to the user and it may be edited and extended as needed. The objects of this part should be recognized by annotators that use various methods: named entities, regular expressions, thesauri etc.;
    - *domain objects*: here the objects typical for a given domain are stored. For instance, for the CV domain, we are interested in objects such as: *Places, Employers, Skills, Education Institutions* etc. For business processes, we would define *Actions, Connections, Actors* and such.

This part of the ontology is intended to be designed for a particular use case. From the IE point of view, the objects from this part may be recognized by semantic annotators or semantic descriptors constructed by the user. Obviously, if a hierarchy of classes if defined, then the instances of super-classes may be inferred from the sub-classes and the subsumption relations.

5.2.1 Layout representation

This section is devoted to the layout representation. We propose to view the document as a two-dimensional artifact, what leads us to identify the basic one- and two-dimensional objects. We build complex objects based on the atomic ones.

The minimal discourse unit, essential to grasping the minimal structure of a text, is defined by the concept of "token". In fact, a token is an elementary object, that represents in the modeling the basic element returned by a text analysis process. We decided to change the keyword to identify class objects by using 'entity' instead of 'class', as it is considered more adherent to the scope and purpose of the whole project. In fact, as a framework for extracting semantic information, starting from recognized entities through specialized annotators, choosing this keyword suggests better the meaning of the underlying concept. Formally a token is defined as:

```
entity token.
```

In the ontological modeling paradigm, it is usual to introduce a "root" for all the objects within the domain, so it is implicitly assumed to be the "ancestor" of all the defined entities.

```
entity ontologyObject.
```

The definition of the token entity could be therefore better defined as:

```
entity token isa ontologyObject (value: string).
```

The relationship is-a represents the well-known subclass-superclass relationship that realizes a hierarchy between the entities involved. The token class will be populated by instances representing single text units (for example, in the simplest case, words). This will be achieved by a tokenization process that, given a string and data of the separators (or characters to be ignored, such as space), is able to divide the string into suitable token. The "value" attribute stored the piece of text associated with the recognized token.

However, this is not enough to fully represent an occurrence of a text (word) within a document. In fact, what is needed to locate the specific instance of the token class is its position within the document. In turn, to find the exact position where a token appears, it is necessary to know its coordinates: the starting position and the end position, relative to the input text input string in the tokenization

process. To this end, we introduce a *position* relation that binds a specific object (or entity) to its coordinates. Formally, this is defined as:

```
relation position(obj:ontologyObject, start:int, end:int).
```

The position relation is used for each token instance, where each of them is associated with the start and end attributes representing the initial and final position of the token in the string.

This modeling is capable of capturing the entities and their positions with assumption that the whole document is considered "one-dimensionally" (one could imagine a normalization of a document to a single very long string). In the presence of two-dimensional documents (more close to the intuitive view on documents), this model needs to be enriched to adapt to the new requirements. In order to locate atomic and more complex elements within a document, we define an internal representation of the spatial placement of one- and two-dimensional objects. In the following listing we present the structural elements of a two-dimensional document: one-dimensional objects such as token and delimiters, and two-dimensional ones such as empty and filled *cells*.

```
entity ontologyObject.
  entity layoutObject isa ontologyObject.
    entity oneDimensionalObject isa layoutObject.
      entity token isa oneDimensionalObject(value:string).
      entity delimiter isa oneDimensionalObject.
        entity startOfLine isa delimiter.
        entity endOfLine isa delimiter.
      entity biDimensionalObject isa layoutObject.
        entity cell isa biDimensionalObject.
          entity emptyCell isa cell.
          entity filledCell isa cell(value:string).
```

For each two-dimensional object, a `bi_position` relation is defined that specifies the line and column on the abstract grid of the document where the specific object appears. For one-dimensional objects that are placed inside the cells, two relations have been provided: `one_position` that denotes the position of the object within the cell and `belongs_to` which identifies the cell in which it appears by object identifiers. Below is the formal definition of such relationships.

```
relation position(obj:ontologyObject, start:int, end:int).
relation one_position(obj:oneDimensionalObject, start:int,
    end:int).
relation bi_position(obj:biDimensionalObject, xstart:int,
    ystart:int, xend:int, yend:int).
relation belongs_to(obj:oneDimensionalObject, obj2:
    biDimensionalObject).
```

5.2.2 Content representation

The main part of the proposed model is the representation of the document content. Most of the concepts in this part of the ontology will vary between the domains, but a general structure and main classes are included in the ontology regardless of the domain. In the case of homogeneous collections, we propose to distinguish even further the concepts that are somehow "between" the individual content of a document, interesting from the Information Extraction perspective, and the content that is common to all the documents in collection and that is helpful in the extraction, but not destined to be included in the target database.

Taking into account the ontological representation of a generic document provided in the previous section, the next step, for the purpose of using the specific domain information that can be tracked in the document, is to provide a correct modeling of the application domain to which the the document belongs. In this modeling operation, it becomes relevant to identify entities of interest and the relationships in which they are involved e.g.:

```
entity email isa ontologyObject(user:string,domain:string).
entity date isa ontologyObject(day:int,month:int,year:int).
```

The purpose is to describe the information of the reality of interest according to a conceptual representation of the domain of the problem. It is also a crucial step for the simplification and effectiveness of subsequent processing phases. Some of the concepts anticipated in the documents may be domain-specific, while the others may also appear in other collections and domains. This may have practical implications e.g., for the domain-independent, common classes, there may exist semantic annotators, while for the collection-specific categories, they may be missing. Thus, we have distinguished *basic objects* to denote the general concepts from the *domain objects* that are specific for a given collection.

Another perspective on the content in a homogeneous collection of documents is that of the recurring elements, typical labels. These are also domain-specific objects, but rather from the domain of the structure and layout of these specific documents, and not the topics they are concerned about. For these typical labels, we have identifies a separate category of *category markers*, as the objects themselves are not interesting from the Information Extraction viewpoint, but rather serve to localize the specific objects within the documents. In the end, we capture both perspectives with the following structure of entities:

```
entity ontologyObject.
    entity contentObject isa ontologyObject.
        entity categoryMarker isa contentObject.
        entity relevantObject isa contentObject.
            entity basicObject isa relevantObject.
            entity domainObject isa relevantObject.
```

5.3 Use case example

To illustrate the concepts introduced in this chapter, in this section we show the application of our model to a homogeneous documents collection of Curriculum Vitae. The accepted format is the one established by the Europass standard for the definition of curriculum vitae. The Europass format has a fixed structure organized in sections and subsections that allows to present candidate's qualifications and skills. A candidate must enter their personal information; provide a description of the professional experience; describe educational and training path; as well as portray in detail their skills and competences, acquired during the course of training, career or daily life. In particular, a generic curriculum vitae is structured in the following sections: *Personal information, Professional experience* (multiple blocks can be present), *Education and Training* (Multiple Blocks can be present), *Personal skills and competences, Social skills and competences, Organizational skills and competences, Technical and computer skills and competences, Other skills and competences, Further information*.

Choosing this format ensures that the input documents have roughly the same two-column layout, and that the content is similarly organized in specific sections. In addition, there is a predefined set of labels in the documents, which define the characteristics of a candidate, and a common domain of values (considering curriculum in the same language). In reality, there may be differences between documents, in fact, candidates tend to adapt the model to their needs. However, a collection of Europass documents can be considered homogeneous and a reference model can be assumed.

The application of the described approach translates into the definition of different entities, instances, hierarchies capable of modeling the specific domain considered in addition to the already-depicted basic components. With the ultimate aim of capturing the information pertaining to each section, it may be useful to identify the sections of a curriculum. This can be achieved by introducing an *label* entity that will mark the specific sections having their name as an attribute:

```
entity label isa categoryMarker(name:string).
```

An instance of this class is, for example, *personal_info_label:label(name: "Personal Information")* which represents the first section. For simplicity and uniformity of representation, the same class can be used to represent any subsections of a section. For example, the following are some of the subsections relating to the Personal Information section:

```
address_label:label(name:"Address").
telephone_label:label(name:"Telephone").
email_label:label(name:"Email").
nationality_label:label(name:"Nationality").
gender_label:label(name:"Gender").
```

These definitions allow to capture the static part of the document, but dependent on the domain. To extract information from the most variable and unpredictable parts (the actual content), one needs to provide other entities that are appropriate to the exact purpose. For example, one could define entities such as:

```
entity candidateName isa domainObject(name:string).
entity candidateAddress isa domainObject(address:string).
entity candidatePhoneNumber isa domainObject(phone:string).
entity candidateEmail isa domainObject(email:string).
```

to refer to the candidate's specific information. In addition, one could define semantic categories of interest for the application domain: :

```
entity person isa domainObject(name:string, surname:string,
    birthdate:date, id:string).
entity place isa domainObject(name:string).
entity date and is domainObject(day:integer, month:integer,
    year:integer).
entity company isa domainObject(name:string).
entity educationInstitution isa domainObject(name:string,
   country:string).
```

Graphically, the representative ontology for this use case is depicted in Figure 5.1.

Figure 5.1: Fragment of the ontology for a collection of CVs documents.

Chapter 6

Semantic Descriptors

The analysis of the semantics of document content, and of the layout and structure of a document are the two main tasks that prepare basic objects for further processing in our Information Extraction process. The unified ontological representation of these objects provides basic inference mechanisms (subsumption reasoning). In order to extract more complex objects, we need a more advanced method of knowledge processing. In this chapter, we introduce a rule-based formalism of *semantic descriptors*. We explain their application and assumptions. present the syntax and semantics and show how they can be evaluated thanks to a translation to an ASP program.

6.1 Existing rule-based solutions for IE

The semantic descriptors follows the line of rule-based IE. In fact, rules were analyzed as the first method for information extraction. Currently, more attention in academic world is devoted to statistical and machine-learning approaches. However, in a corporate world, the interest in rules reborns, as the grounded rule-based formalisms are effective and more intuitive for non-experts.

In general, Information Extraction strategies based on machine learning techniques and algorithms first analyze a (often manually annotated) document or text corpus to learn extraction rules, and then examines new documents to propose automatically generated extractions to the user. The user can confirm or correct the proposed extractions. Instead, a rule-based Information Extraction strategy does not require the learning phase. The rules for extracting information are created manually. These approaches have been applied in a variety of academic applications (Araneus [82], DEByE [74], Minerva [30], NoDoSE [1], TSIMMIS [58], SoftMealy [63], RAPIER [19], WebOQL [9], WHISK [109], SRV [43, 44], WIEN [73], and XWrap [77]. STALKER [85], W4F [99], Squirrel [24] and

RoadRunner [31]. Numerous tools have been developed in this area, offering complete and robust software packages: Denodo (www.denodo.com), Kapow Technologies (www.kapowtech.com), QL2 (www.ql2.com), Dapper (www.dapper.com),
Fetch Technologies (www.fetch.com) or Lixto (www.lixto.com).

Existing tools can be classified according to different features: the input of the extraction process, the degree of process automation, and the techniques for writing the extraction rules. Other types of classifications for Information Extraction systems take into account additional aspects: page content (columns, tables, figures), patterns and extraction methods (regular expressions, Prolog rules, NLP rules, heuristics, Markov models); degree of automation, background knowledge, output. As for the type of documents processed, the input documents of an extraction task can be classified as structured, semi-structured or unstructured. In addition, different systems work exclusively on the HTML tag structure (W4F, XWRAP, RoadRunner). while others support non-HTML inputs (RAPIER, Soft-Mealy, TSIMMIS, Minerva, Jedi, NoDo, SRV, Stalker, Lixto).

Systems can also be classified according to their "category" which allows to distinguish different approaches. One approach is that followed by HTML-aware systems. This group of systems closely depends on the structural features of HTML documents for data extraction. The systems usually use tree representations that reflect the hierarchy of HTML markup. Another approach is the one offered by Natural Language Processing techniques. In this case, the systems apply various procedures such as semantic filtering and tagging to build relationships between the elements of the document, in order to derive the extraction rules. These rules are based on syntactic and semantic constraints that help identify relevant information within the documents. A slightly different capability is offered by wrapper induction systems. The induction process consists in generating extraction rules from a set of training examples. Unlike NLP-based tools, these systems are not based on linguistic constraints, but rather on the structure of the document and this makes them more suitable for use on HTML documents. Finally, there are ontology-based systems: this class includes the systems that, given a target structure (schema) for the objects of interest, try to locate portions of data that implicitly meet the schema in the pages. The ontology-based systems provide that, given a specific application domain, one can use an ontology to locate the terms in a document and build objects.

With respect to the types of extraction rules, many systems use rules that are represented by regular grammars to identify the beginning and end of relevant data, while some use rules following the first order logic. Using regular expressions is more suitable for semi-structured inputs, especially template-based, while for unstructured documents the use of first-order logic tools has yielded better results. Despite the differences, it is possible to identify the characteristics common

to the different Information Extraction systems. Such systems allow, typically, to represent extraction rules that express patterns defined on the basis of the syntactic structure of the information to be extracted. In other words, existing approaches depend, to a large extent, on the internal representation of digitally processed documents, and therefore are specialized in specific internal formats (plain text, HTML, XML, etc.). Moreover, the formalisms used to represent the extraction rules have rather limited expressivity.

A limitation common to many solutions is the adoption of syntactic approaches, often carried out by using regular expressions (regex). Regular expressions are a simple and declarative formalism to specify patterns to extract. An advantage of regular expressions is that they can be evaluated efficiently: recognizing whether a string belongs to a regular language is feasible in linear time. However, one of the disadvantages of this approach is the limited expressive power, which is not suitable for supporting robust Information Extraction tasks. In addition, the use of regular expressions in complex tasks is often verbose, chaotic and non-modular.

Another weakness of traditional approaches is their dependence on the specific format of the document. This means that systems designed to manage HTML documents often are not equally suitable for working on documents in a different format, such as PDF. The process of extraction from such sources is certainly more complex, since these documents are not hierarchical structures as in the case of HTML. There is extensive literature about the analysis and interpretation of PDF documents [40, 42, 76] which often seeks to recreate such structures. Other aspects that are not fully managed are those that can be used to express patterns that capture two-dimensional data structures such as tables, lists, etc. Utilizing a two-dimensional representation of semi-structured and unstructured documents would allow an abstract and unifying representation of the different formats of existing documents: pattern recognition would be exploited by using semantics and spatial relationships between objects.

6.2 The semantic descriptors approach

In this section, we present the semantic descriptors approach. We introduce with examples what kind of situations can be captured, and what can be analyzed to extract interesting information from a document. We show how the semantic descriptors refer to the structure and content of a document and how the knowledge about these two aspects can be used to create and extract new objects.

The proposed language is mainly characterized by two aspects. First, it allows the definition of semantic object-oriented extraction rules. Second, it can be used on structured, semi-structured or unstructured documents. The use of an ontological support language meets the need to provide adequate semantic support for

extracting meaningful information from unstructured and semi-structured documents. To extract meaningful information from this type of files, it is necessary to analyze them from a semantic point of view using a priori and/or contextual knowledge. An ontology is a useful tool for managing information semantically, and the language itself aims to allow associations between pieces of information and ontological concepts from which new knowledge can be derived. Pattern specification rules can therefore account for ontological language objects. The approach, fits perfectly to the integration of existing techniques. It makes it possible to extract information from different document formats thanks to the high level of abstraction provided by the logical view of the document.

Note that the notion of semantic descriptor was inherited by HiLeX [79]. The idea behind the definition of that language was to provide a tool that combines simple knowledge elements to obtain and build more complex objects. In our work, we have extended the language of the descriptors defined in [79] to be able to treat both one-dimensional and two-dimensional objects in a uniform way and to allow for some intuitive constructions (such as '...' for skipping not-interesting elements in text) that represent more advanced operations on text.

The phase of formulating semantic descriptors requires some attention. In particular, the designer should build objects according to a bottom-up logic, starting from the "leaves" provided by the annotators. This is because semantic descriptors allow to organize two-dimensional objects (such as cells) and one-dimensional (such as tokens) in descriptions to build new information, encoded into more complex objects. To better understand the use of language, and before giving it a more formal definition, we provide an overview of its features by using examples.

Example 1 A user can identify parts of the document that will help to structure it, and help to localize other data portions, e.g.:

```
<cv_email_label_box()> ::- <filledCell(X)>
                           CONTAINS <cv_email_label()>
```

With this simple descriptor, we intend to create a two-dimensional entity that we call `cv_email_label_box` that defines a cell in which there is a one-dimensional object `cv_email_label`. The object we want to extract always resides on the left-hand-side of the operator "`::-`" (in the head of a descriptor), while on the right (in the body), there are objects that must be found in order to create it. In this example, we look for a cell, in particular a `filledCell` within which (this is expressed with a keyword `CONTAINS`) there is a particular domain concept, an `cv_email_label`. On the one hand, one can assume that the whole document is divided into filled and empty cells (these facts are given by bi-dimensional tokenizer). On the other, an `cv_email_label` is a concept that can be recognized by a label annotator. Thus, if we find a cell with this label inside, the cell can be

recognized as a `cv_email_label_box`. In this example, we do not use nor pass any attribute values.

Example 2 Descriptors can join several cells that appear in a document one after another (horizontally or vertically). This is useful, if we want to say that there exist a particular object, if there is a specific sequence of cells:

```
<candidateEmail(E)> ::- {E:='';} <cv_email_label_box()>
             (<filledCell(X)> CONTAINS <email(X)> {E:=X;})
```

In this example, the description should be read as: "A `candidateEmail` is a bi-dimensional object that captures two cells: the first is a cell which has been recognized as an `cv_email_label_box` and is followed by a `filledCell` that contains a (one-dimensional) object `email` with value X. The new object spans across both cells, and the value of the object `email` becomes the value of the object `candidateEmail`.

Note the introduction of an attribute here. If we want to give an attribute to the new object (the one declared in the head of a descriptor), this attribute must be initialized at the beginning of the descriptor body. Here, we initialize it with an empty value (`{E:="";}`). If we want to pass the value of an attribute from the concept in body to the concept in head, we do it with an assignment instruction, `{E:=X;}`. In this example, we use:

- a concept `email` that is recognized by a semantic annotator,
- a concept from another semantic descriptor: `cv_email_label_box`,
- a class `filledCell` given by a two-dimensional tokenizer.

This descriptor extracts an object with an attribute that stores the actual value of the candidate's e-mail address. By using the context (first there is a box with an e-mail label, and then there is a cell with an e-mail address), we ensure that, even if the CV contains a few e-mail addresses, we select the correct one, because if the e-mail address appears in this place, it must be in the Personal Information section and thus, it is the e-mail of the candidate.

Example 3 We can also aggregate the concepts and attributes extracted by other semantic descriptors to build more complex ones:

```
<personalInformation(N, S, A, P, E, Nt G)> ::-
             {N:='';S:='';A:='';P:='';E:='';Nt:='';G:='';}
  <candidateName(X)> {N:=X;} <candidateSurname(X)> {S:=X;}
 <candidateAddress(X)> {A:=X;} <candidatePhone(X)> {P:=X;}
<candNationality(X)> {Nt:=X;} <candidateGender(X)> {G:=X;}
```

This semantic descriptor aggregates results of other descriptors that extract single information about a candidate. First, we have the initialization of all the attributes, and then a sequence of concepts that must appear one after another (vertically or horizontally). Note that this aggregation must adheres to the template of the input documents (it must reflect the order in which information is given in the documents). The line breaks within the descriptor do not influence its semantics. The aggregation is checked horizontally and vertically to cover all the possibilities.

Example 4 Within cells, we can create complex one-dimensional objects by using a recurrence structure "`(sequence of terms)+`" and a keyword "`...`" that allows to skip some objects, e.g.:

```
<list_of_skills(S)> ::- {S:='';} <startOfLine> ...
          (<IndustryTerm(S1)>{S+=S1;} ...)+ <endOfLine>
```

This descriptor works for one-dimensional objects that are all located in one cell (treated as a single line). Here, we want to create a list of attributes, so we initialize the attribute `S:=''`. Then, we look for a concept `IndustryTerm`, given by a semantic annotator, append its attribute value to `S`, and place the term in a recurrence structure. The expression `(<IndustryTerm(S1)>{S+=S1;} ...)+` means that there may be some objects after the `IndustryTerm` that we ignore, and if we find another object `IndustryTerm`, we append its attribute value to the list again. By using the keyword "`...`" before the recurrence, we say that before encountering the recurrence, we can skip some objects; it means that the recurrence structure may appear anywhere between the `startOfLine` and the `endOfLine`. The descriptor creates a new object `list_of_skills` that stores as an attribute a list of `IndustryTerm` objects' attributes. The concepts `startOfLine` and `endOfLine` are delimiters given by a tokenizer.

Example 5 Finally, semantic descriptors may use the information about the placement of semantically-enriched data within the document (e.g. presence of a given object within specific section) to produce new objects that are not explicitly present in text, e.g.:

```
<list_of_practical_skills(S)> ::- {S:=''}
     <cv_work_activities_and_responsibilities_label_box()>
     (<filledCell(X)> CONTAINS <list_of_skills(X)> {S:=X;})
```

In this example, we use a concept from the domain ontology, which is a filled cell `cv_activities_and_responsibilities_label_box`, to check if the list of skills (for which a descriptor is given in the previous example) has been found in the Work Experience section (only in this section, a label for "activities and responsibilities" can be found).

6.3 Syntax and semantics

In this section, we present the syntax and semantic of the descriptors. The syntax is based on a concept of positive rules. The semantic will be explained by giving a reference to automata representation and evaluation with logic programming.

Syntax A semantic descriptor d is defined as

$$head(d)\ \ ALIGNMENT\ \ body(d)$$

where $head(d)$ represents the head of the descriptor and $body(d)$ represents its body. *ALIGNMENT* is a symbol from the set $:: -, :: |, ::$ and defines the type of *alignment* of the descriptor d. Specifically $:: -$ encodes horizontal two-dimensional alignment, $:: |$ encodes two-dimensional vertical alignment and $::$ encodes one-dimensional alignment.

Consider an ontology O and a set of Z values, such that Z is composed of two subsets Z_{int} and Z_{str} that identify the set of natural numbers and the set of strings on a given alphabet, respectively. The head of the descriptor $head(d)$ is made up of a class atom on ontology O. The body of the descriptor $body(d)$ is defined as a sequence of at most 3 *blocks* in the order: *LeftBlock*, *CapturingBlock* and *RightBlock*. Each block, in turn, consists in at most three *sequences*. In practical examples, the descriptors' bodies often consist of only one block. If there are more blocks, their internal structure is usually simple. However, the full possible structure of a descriptor body is shown in Figure 6.1.

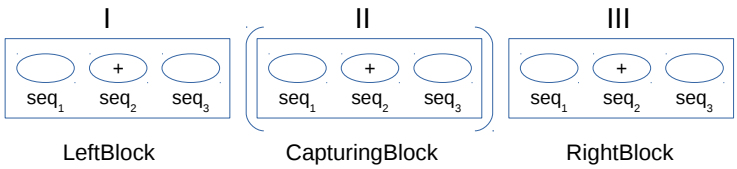


Figure 6.1: General structure of a semantic descriptor's body

To be more specific, a descriptor may have one of the following form:

- $\langle LeftBlock \rangle\ \langle CapturingBlock \rangle\ \langle RightBlock \rangle$
- $\langle LeftBlock \rangle$
- $\langle LeftBlock \rangle\ \langle CapturingBlock \rangle$
- $\langle CapturingBlock \rangle\ \langle RightBlock \rangle$

Left and right blocks have identical structures. A capturing block has additional brackets around the actual block content ("[...]") to mark the spatial range within a document that will be used for the new object created with that descriptor.

Each block consists of sequences. The following forms of blocks are possible:

- $seq_1\ (seq_2)+\ seq_3$
- seq_1
- $seq_1\ (seq_2)+$
- $(seq_2)+\ seq_3$
- $(seq_2)+$

The expression"$(seq)+$" denotes a recurrence structure. It means that the content of such a sequence may be found several times one after another.
Finally, each sequence consists of *terms*: $\langle term_1 \rangle \ldots \langle term_n \rangle$ and *instructions*. The terms represent basic objects in the content, such as tokens and delimiters, and also objects annotated by semantic annotators (Persons, Places etc.). A term can be:

- A simple term
- A term defined from simple terms as: *simple term CONTAINS simple term*
- An anonymous term defined by the symbol '...'

An instruction can be:

- an assignment instruction, for setting a value to a variable
- an arithmetic instruction, for performing simple operations on integer variables (addition, substraction multiplication etc)
- a list instruction, for manipulating lists, mainly to add an element to a list

Semantics Let us now discuss the semantic aspects in accordance with the descriptor definition that has been provided. Without loss of generality, we can consider a single block. Within each block, five different structures are allowed: *seq1 (seq2)+ seq3*, *seq1*, *seq1 (seq2)+*, *(seq2)+ seq3* and *(seq2)+*. If we consider the body of the descriptor as a regular expression of *terms*, then it is possible to construct an equivalent Finite State Machine (FSM) for it, denoted as $M(d)$. For example, in the Figure 6.2 the FSM associated with a block of the most complete form: *seq1 (seq2)+ seq3* is depicted.

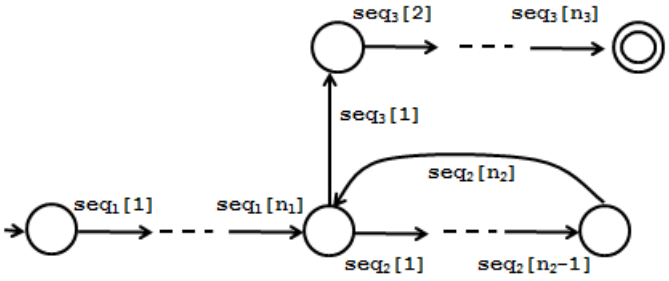


Figure 6.2: Automata-based representation of a semantic descriptor's body

This representation encodes the semantics of a block that contains no *anonymous terms* '...'. On the other hand, if an anonymous term appears, the finite state machine changes and an additional loop, that encodes the mechanism of an implicit "recurrence" of the anonymous term, is added. For example, in the Fig. 6.3, the finite state machine associated with the body of a full-size block: *seq1 (seq2)+ seq3*, where anonymous terms occur in *seq1* and *(seq2)+* sequences, is depicted.

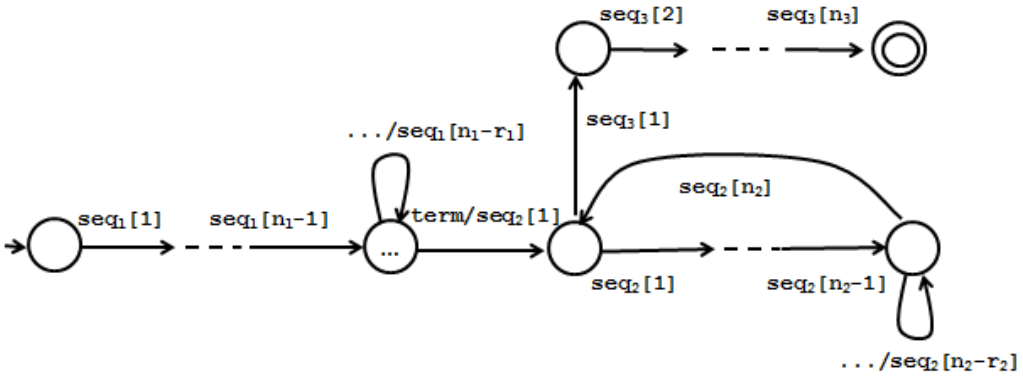


Figure 6.3: Automata-based representation of a block with *anonymous terms*

Note that the presence of a term defined by combination of simple terms: *simple term CONTAINS simple term* does not require any modification to the definition of the finite state machine, since it is solved by a simple check of a condition, if one object is within the two-dimensional boundaries (coordinates) of another. This type of term is used to check whether one *simple term* that represents a two-dimensional objects such as a cell contains another *simple term* that refer to an

one-dimensional ontological object, such as a token or a text semantically annotated with a specific category.

In the presence of two or more blocks, the finite state machine representing the whole descriptor body is obtained from the composition of the single finite state machines representing each block. Intuitively, by default, the positions to be attributed to the object defined in the head of the descriptor coincide with the range of positions starting at *LeftBlock* and ending at the positions of the *RightBlock*. However, in the presence of a *[CapturingBlock]*, the positions to be attributed to the object defined in the head of the descriptor coincide with those of the capturing block. In a sense, in the latter case, the left and right blocks constitute the "context" in which the actual object is identified.

6.4 Logic-based evaluation

In this section, we explain how the semantic descriptors can be evaluated with use of existing inference engines. We show how a single descriptor represents semantics of a set of logic rules, and how we generate them by using an intermediate matrix-based representation of finite state machines.

6.4.1 Translation to logic rules

For the translation of a semantic descriptor into a set of logic rules, more specifically, a DLV program, we use the automata representation of the descriptors introduced in the previous section. For each descriptor, we build at most three automata, an automaton for each block. The elements of the blocks are mapped onto the automata transitions and organized appropriately. These transitions contain information about the terms that are looked for in a document, and the order in which they must appear. In the end, all the automata are "concatenated", such that the last state of one automaton is followed by the first state of the consequent one. This representation is then used to generate appropriate logic rules; first, the one that represents an initial state (for the whole descriptor), and then the rules that represent transitions of the descriptor's automata. In order to reflect the "orientation" of the descriptor (horizontal or vertical), the alignment of the descriptor is recognized and used to set appropriate coordinate variables in the rules. At the end, the algorithm generates a separate set of rules whose aim is to create a new object that is defined in the head of the considered descriptor. The algorithm for a set of descriptors into a DLV program is shown in Listing 1.

Let us take a closer look at the particular stages of the algorithm, namely: (1) creating automata that represent the descriptor body, (2) building logical rules

Algorithm 1 Translate a set of descriptors into a DLV program

```
1:  procedure BUILDDLVPROGRAM(Descriptors)
2:      rules ← ∅
3:      for all d ∈ Descriptors do
4:          rules ← rules ∪ BUILDINITIALSTATE(d)
5:          automata ← CREATEAUTOMATA(d)
6:          for all a ∈ automata do
7:              if a ≠ null then
8:                  align ← ALIGNMENT(d)
9:                  rules ← rules ∪ BUILDRULES(a, align)
10:             end if
11:         end for
12:         rules ← rules ∪ BUILDOBJECT(d)
13:     end for
14:     return rules
15: end procedure
```

from the automata transitions, and (3) building rules to construct a new object defined in the descriptor head.

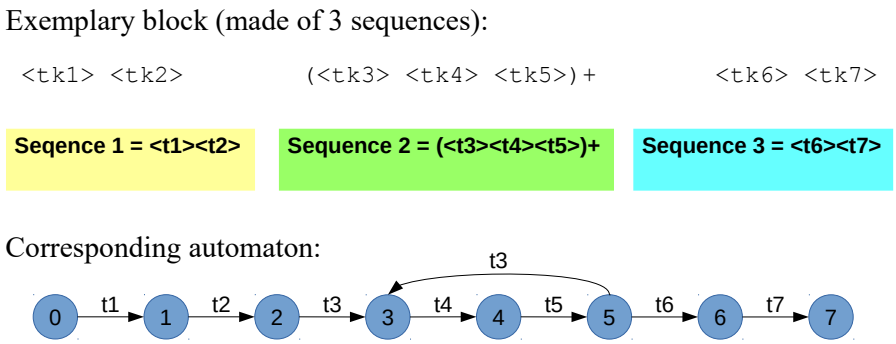
Creating automata Listing 2 presents the CREATEAUTOMATA(d) function that for each block in the descriptor creates an automaton. The transitions inside each automaton take into consideration the *offset* which ensures that the next automaton starts where the previous ends.

Algorithm 2 Constructing automata for a descriptor

```
function CREATEAUTOMATA(d)
    offset ← 0, a ← ∅
    for block ∈ {d.LeftBlock, d.CapturingBlock, d.RightBlock} do
        if block ≠ null then
            a ← a ∪ CREATEAUTOMATONFORBLOCK(block, offset)
            offset ← offset + sizeOf(block)
        end if
    end for
    return a
end function
```

Function CREATEAUTOMATONFORBLOCK$(block, offset)$ creates a matrix representation of an automaton for a block, taking into account the offset. Each automaton has a size equal to the number of terms within the associated block. The

transitions of the automaton are constructed from the sequences of terms in the associated block, as depicted in Figure 6.4. The numbers of the automaton states



	0	1	2	3	4	5	6	7
0		t1						
1			t2					
2				t3				
3					t4			
4						t5		
5				t3			t6	
6								t7
7								

Figure 6.4: Creating an automaton for a single block

are mapped onto rows and columns of a matrix, and the transitions are placed at the intersection of appropriate row and column i.e., for a transition τ from state s_i to state s_j with a transition label *term* the value *term* is placed in the (s_i, s_j) position in the matrix. Note that if we encounter recurrent structure $(t_1, \ldots, t_n)+$ in the descriptor, then the first element of the sequence placed inside the recurrence will appear twice in the matrix.

Building logic rules for the automata After constructing the automata, the algorithm build the rules. First, the initial state is built. In fact, this rule is independent from the automata structure, for it is simply a logic fact. The predicate name is constructed from the keywords: `init_conf_` and a descriptor name (predicate name in the descriptor head), and there is a set of variables: the first is always equal to 0, as it represent the initial state, and the rest are placeholders for as many variables as many attributes are defined for the considered object in the descriptor's head.

Then, the rules that represent the transitions of the automata are added. For every row *r* and for every column *c* in the matrix representation of the descriptor body, if there is a value on the position (r,c), we build a logic rule that represents the transition from state number *r* to state numbered *c* as follows:

1. We create a DLV term representing previous state (here the predicate name is `init_conf_`*descriptor-name* with initialization of attributes, or `conf_`*descriptor-name* with arguments representing position and variables).
2. We create DLV terms representing the transition term (the one that must be encounter to proceed): the term name and attributes, two- or one-position relation, optionally terms representing the keyword *CONTAINS*.
3. We build the head of rule, representing the next state (appropriate variables, attributes, location must be updated.
4. We build the rule of the above head and the body elements.

Building logic rules to create a new object Finally, there are rules that construct a new object, given by the descriptor. A new object is associated with a new unique identifier, and a `bi_`- or `one_position` relation, depending on whether this object is bi-dimensional or one-dimensional. We start by building an auxiliary rule whose predicate name is `aux_`*descriptor-name*, and whose attributes are:

1. attribute 1: a new unique identifier,
2. attribute 2: content of the new object captured by the descriptor,
3. attributes 3-4 or 3-6: 2 or 4 coordinates of the object.

Then we build a rule, in which in the head we construct a term, whose predicate name is *descriptor-name*, and the attributes (identifier and content) are copied from the auxiliary rule. We do not copy the positions. Finally, we build a rule, in the head of which we construct a term, whose predicate name is `bi_position` or `one_position`, and the attributes (identifier and position) are copied from the auxiliary rule.

6.4.2 Exemplary translations

Let us now show a few examples to illustrate the details of the translation.

Example 1 Let the semantic descriptor be:

```
<candidateEmail(E)> ::- {E:="";} <filledCell(C)>
                        CONTAINS <email(X)>{E:=X;}.
```

In this descriptor, we have only one block that contains only one sequence. We want to find a cell that contains a fragment annotated as `email`. The automata representation is very simple, it consist in only two states and one transition.

The initial state rule is built as follows:

```
init_conf_candidateEmail(0,"").
```

The following rule is as follows:

```
conf_candidateEmail(1, Lc6, Xs, Ys, Xe, Ye) :-
    init_conf_candidateEmail(0,""), % previous state
    filledCell(Id, Lc5),  % next term to find
    bi_position(Id, Xs, Ys, Xe, Ye),% the term's position
    email(IdContains, Lc6),  % an object inside
    belongs_to(IdContains,Id).
```

Finally, the rules constructing the object are as follows:

```
% auxiliary rule
aux_candidateEmail(AutoGen,Gl3, Xs, Ys, Xe, Ye) :-
             conf_candidateEmail(1,Gl3, Xs, Ys, Xe, Ye),
             #newId(AutoGen).
% building a new object
candidateEmail(AutoGen,Gl3) :-
             aux_candidateEmail(AutoGen,Gl3, Xs, Ys, Xe, Ye).
% setting the object position
bi_position(AutoGen, Xs, Ys, Xe, Ye) :-
             aux_candidateEmail(AutoGen,Gl3, Xs, Ys, Xe, Ye).
```

Example 2 Let the next descriptor be:

```
<candidateName(X)> ::- {N:="";} <eu_cv_name_label(L)>
                       [<filledCell(X)> {N:=X;}].
```

This descriptor is also relatively simple, but it is composed of two blocks, *Left* and *Capturing*. We want to find two cells following on another, the first being a specific label, and the second – a filled cell with a value. For the final object, however, we only want to extract the position of the second cell. The automata representation consists of two automata, each with one transition.

The translation is as follows: Initial state rule

```
init_conf_candidateName(0,"").
```

The following rules

```
conf_candidateName(1, "", Xs, Ys, Xe, Ye) :-
              init_conf_candidateName(0,""),
              eu_cv_name_label(Id, Lc1),
              bi_position(Id, Xs, Ys, Xe, Ye).

conf_candidateName(2, Lc2, Xe, Ys, Xe_1, Ye) :-
              conf_candidateName(1, Gl1, Xs, Ys, Xe, Ye),
              filledCell(Id, Lc2),
              bi_position(Id, Xe, Ys, Xe_1, Ye).
```

Rules building the object:

```
aux_candidateName(AutoGen,Gl1, Xs, Ys, Xe, Ye) :-
             conf_candidateName(2,Gl1, _,Ys,_,Ye),
             conf_candidateName(1, _ , _, _,Xs,_),
             conf_candidateName(2, _ , _, _, Xe, _),
             #newId(AutoGen).
candidateName(AutoGen,Gl1) :-
             aux_candidateName(AutoGen,Gl1, Xs, Ys, Xe, Ye).
bi_position(AutoGen, Xs, Ys, Xe, Ye) :-
             aux_candidateName(AutoGen,Gl1, Xs, Ys, Xe, Ye).
```

With respect to the previous example, note the difference in the rules building the new object. Here, the *Capturing* block is reflected by recalling appropriate positions that were present at particular states.

Example 3 Let the next descriptor be a vertical bi-dimensional one:

```
<personalInformation(P, E, Nt, B)> ::|
                {P:="";E:="";Nt:="";B := "";}
                <candidatePhone(X)> {P:=X;}
                <candidateEmail(X)> {E:=X;}
                <candidateNationality(X)> {Nt:=X;}
                <candidateBirthdate(X)> {B:=X;}.
```

This descriptor aggregates vertically several objects (extracted with use of other descriptors). In the automata representation, there are 5 states and 4 transition.

The translation is as follows: Initial state rule:

```
init_conf_personalInformation(0,"","","","").
```

The following rules:

```
conf_personalInformation(1,Lc11,"","","",Xs,Ys,Xe,Ye) :-
  init_conf_personalInformation(0,"","","",""),
   candidatePhone(Id, Lc11),
   bi_position(Id, Xs, Ys, Xe, Ye).
conf_personalInformation(2,Gl6,Lc11,Gl8,Gl9,Xs,Ye,Xe,Ye_1) :-
   conf_personalInformation(1,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye),
   candidateEmail(Id, Lc11),
   bi_position(Id, Xs, Ye, Xe, Ye_1).
conf_personalInformation(3,Gl6,Gl7,Lc11,Gl9,Xs,Ye,Xe,Ye_1) :-
   conf_personalInformation(2,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye),
   candidateNationality(Id, Lc11),
   bi_position(Id, Xs, Ye, Xe, Ye_1).
conf_personalInformation(4,Gl6,Gl7,Gl8,Lc11,Xs,Ye,Xe,Ye_1) :-
   conf_personalInformation(3,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye),
   candidateBirthdate(Id, Lc11),
   bi_position(Id, Xs, Ye, Xe, Ye_1).
```

Rules building the object:

```
aux_personalInformation(AutoGen,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye):-
 conf_personalInformation(4,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye),
 #newId(AutoGen).
personalInformation(AutoGen,Gl6,Gl7,Gl8,Gl9) :-
 aux_personalInformation(AutoGen,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye).
bi_position(AutoGen, Xs, Ys, Xe, Ye) :-
 aux_personalInformation(AutoGen,Gl6,Gl7,Gl8,Gl9,Xs,Ys,Xe,Ye).
```

Chapter 7

Ontology-driven IE: The Knowrex Framework

Everything that was presented so far in the thesis, will be used in this chapter to introduce *Knowrex* — a framework for information extraction from a homogeneous collection of documents. From the semantic and document layout analysis, through the ontology-based knowledge representation, to the rule-based processing — everything is integrated and working together. First, we will show what are the essential features and the architecture of Knowrex framework. Then, we will recount how the user interacts with the system: how a generic framework can be adapted to a specific task (collection of documents), and how the system transforms a set of pdf documents into an instance of a database. Finally, we will explain how particular elements have been implemented, and how respective implementation decisions influence the functionality and flexibility of the system. This chapter is partially based on the co-authored papers [2, 3].

7.1 System overview and architecture

Knowrex is an Information Extraction system. It can process both single documents and collections of them. The system has a modular architecture, it consists of a core part and external tools. In this section, we explain the main assumptions of the framework and functionality of the respective modules.

KnowRex is a *framework* that allows to develop systems for *Semantic Information Extraction* (i.e., information extraction based on the meaning of data). In our approach, what drives the whole process is a *semantic view* of the input data. It means that we start developing a new project with KnowRex by deciding what information we want to obtain in the end, and how we want to organize it. It is often possible to semi-formally model the organization of data within a document

(e.g. the DOM model for (X)HTML an XML languages). In KnowRex, however, we take a step further and allow users to define the final semantic view that is independent from the initial structure. This approach is closer to practical use cases, in which specialists are asked to populate existing knowledge (or data) bases by extracting appropriate information from a collection of documents.

When we consider a homogeneous collection of documents, we can assume that the input files share some specific features, that we can capture with a notion of a "template". Based on a template one can define an *object model* that will more formally define the sort of data contained within the documents and, to some extend, the way it is organized (sections or subsections, keywords etc.) Once a model for the collection (characterized with some template) is defined, the information extraction from a set of documents complying to this template will populate the object model with instances from each input document. This is, however, only a part of work. The other important stage of the process is formulating a mapping from the object model to a *target schema*. Such a mapping allows to reorganize objects extracted from the documents and transform them into instances of the desired semantic view (see Figure 7.1).

It is possible to define more semantic views for the same collection of data (e.g. for different use cases). One can even imagine defining several object models for the same input and target schema. KnowRex framework ensures flexibility in these respects, by separating the stages of extraction and the mapping to target schema and enabling reuse of the components.

Within KnowRex, several tools and techniques have been used, namely:

1. a *two-dimensional processor* for recognizing structural elements of documents,

2. one- and two-dimensional *tokenizers* for identifying basic elements of text,

3. *annotators* (third-party semantic annotators, natural language processing tools, pattern recognition tools etc.) that label single words or phrases as belonging to particular categories,

4. *semantic descriptors* that allow to build the object model from the objects obtained from structural and semantic analysis, and

5. *logical rules* that allow to formulate a mapping between knowledge representations (the object model and the target schema).

Deployment of a new project with KnowRex is relatively easy and consists in adapting the core of the system and the external tools to work on specific data to obtain desired results. Development of a new project is divided into two phases:

design and *runtime*. In the former, the designer works on a conceptual level defining the object model (by assuming a certain template) and the target schema, and by setting the tools and writing rules that will govern the data and information transformations. For the extraction step, the bi-dimensional processing tools, the annotators and the descriptors must be adapted, and for the mapping to target schema, logic rules must be defined. These design choices, described in details in Sect. 7.2.1, are materialized and applied in the runtime phase to extract information from the actual documents and populate semantic view(s) defined for them.

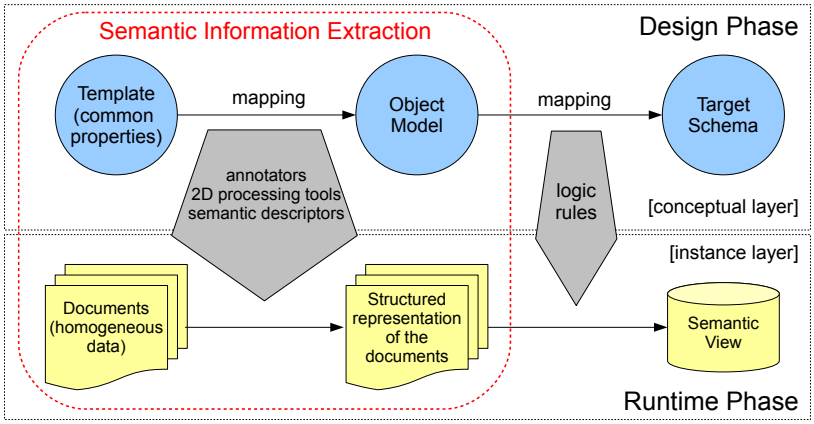


Figure 7.1: Semantic Information Extraction with KnowRex

KnowRex consists of a core system and a set of external tools (see a simplified architecture in Fig. 7.2, for a multi-perspective view see Fig. 1.1). The Semantic Information Extraction is governed by three main components: Bi-Dimensional Unit, Annotation Unit and Language Unit, that are configured during the design phase, and during runtime are responsible for consecutive stages of document analysis, information extraction and processing.

The Bidimensional Unit is responsible for a structural analysis of the input documents. In a general case, the input document may be treated as a one big cell, on which all the analysis is performed. However, the more information about the expected layout, structure or typical features of the input documents is given, the better representation will be obtained, and better quality of information extraction during later stages can be achieved. This unit produces a model of the input document that contains information about its structural elements. After the two-dimensional processing, an ontological model of the structure of the input document is obtained, that is further analyzed semantically. The goal of this unit

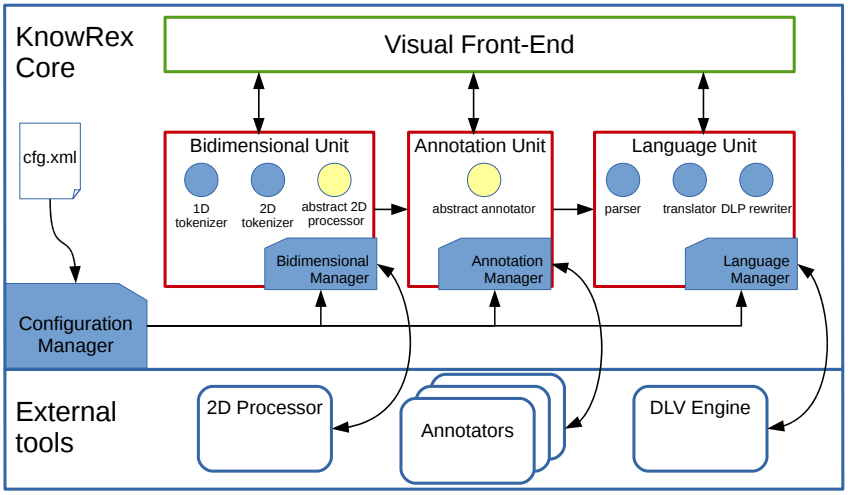


Figure 7.2: Architecture of the KnowRex system

is to recognize the elements of the structure of the document and annotate parts of document with accordance to the model described in Chapter 5.

The system is highly configurable. In the Annotation Unit, the system uses external tools to identify objects of certain classes within the document. There exist mature implementations of semantic annotators and NLP tools, often optimized for certain domains or classes, and using different techniques for recognizing entities. It is beneficial to use one or a few of them to obtain most accurate results. Also, one can write their own annotator tool for recognizing entities from selected domain, thanks to an abstract annotator interface present in KnowRex. The set of used annotators and classes to be extracted is configured in the design phase.

The Language Unit is responsible for high-level extraction of information. The semantic descriptors used here work with the results of the annotation and bi-dimensional stages. They take as input information about the structure of the document, the objects identified by semantic annotators, their placement within the document, proximity to each other etc., and build more complex objects for the object model.

7.2 Using the framework

One of the main advantages of the framework is that it is universal, yet can be adapted for specific domains. The process in which a user describe the characteristic features of their document collection is called a design phase. It basically

consists in extending a basic ontology to cover domain-specific concepts, and then specifying how these concepts should be recognized. In particular, they can either be recognized by available annotators, or by custom extraction rules specified by the user in a form of appropriate semantic descriptors. Also, in this phase a user define the target schema and mapping rules that will help the system map the ontology of extracted objects to the final format (usually relational). Once the design is done, all the settings are compiled into set of executable logical rules, appropriate configuration files etc. In the runtime phase, these settings and files are used to automatically extract information from a given collection of documents.

7.2.1 Design phase

During the design phase, a KnowRex project is configured to perform operations on a collection of documents, to obtain information desired by user, in a particular form. To do it, the designer should reflect on *what they have* and *what the want to obtain* (see Figure 7.3). The former means identifying a template which a vague

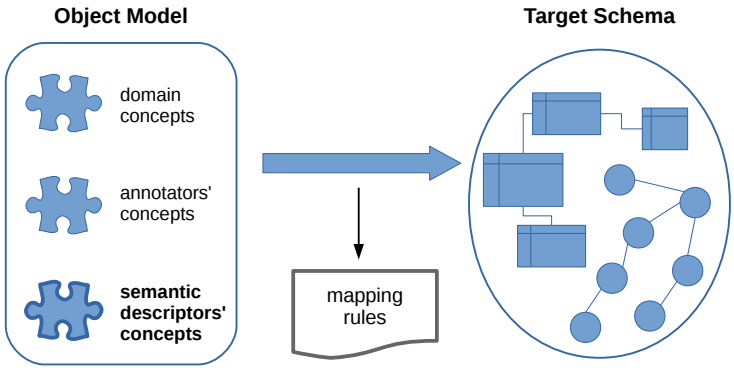


Figure 7.3: Design Phase concepts: object model, mapping, and target schema

concept for describing the common features of documents. A template must be formalized in the form of object model. Elements of it will be extracted by different components (two-dimensional processing tools, annotators, and semantic descriptors). The target schema, in turn, is typically formalized as an ontology or database schema. The designer configures the system and arranges the external tools so that the object model can be built. Then, they write logic rules that map the object model into target schema. The result of the design phase is used at runtime to process the actual documents to create the semantic view.

Definition of the Target Schema This step is crucial for the definition of a desired output of the system. The designer has to decide how to organize the information that will be extracted from the documents. The target schema may be either for a relational database or for an ontology. For the target database, we may consider only the portion of information relevant. To focus attention, let us assume the following target schema:

candidate(Id, Name, Surname, Phone, Email, Address, Nationality, License);
workExperience(Id, Company, BusinessSector, StartDate, EndDate);
candWE(IdCandidate, IdWorkExperience).

The schema should be consistent and realistic i.e., it should be easy to populate it manually, only by analyzing the input documents.

Analyzing the document template and target schema, the designer should create an object model for the input documents. The model does not necessarily represent all the information that can be extracted from the input. It may, for instance, focus only on some of the relevant sections.

Definition of the Object Model By considering the template and the target schema, the designer fixes an object model (the ontology) for considered collection, which consists of a hierarchical forest-like structure. To define it, we use the ontology languagein which one can define object types, relation types and express relationships between objects. Recall that object types are preceded by keyword **entity**, and the subclass relationship is expressed via the term **isa**. Objects may have zero or more attributes which are specified in the type definition, by giving their names and types. By default, a class inherits attributes from its super-class. Relation types can be defined by keyword **relation**, and giving a name and attributes for this relation.

Within the object model, a few types of objects are identified. First, there are layout objects — concepts that belong to an ontology representation of a document. This representation is independent of the use case, it is present in KnowRex by default and does not need configuration (see Figure 7.4).

In the second group of concepts, the content objects, there are categories that can be identified within the content of the document. This set of concepts is defined by a designer and heavily depends on the use case. Recall from Chapter 5 that within the content objects, we distinguish "category markers" – classes denoting collection-specific objects that have a specific role: they mark portions of text in the documents. A user can extend this general class by providing sub-classes for it. The class itself, however, cannot be modified (see Figure 7.5).

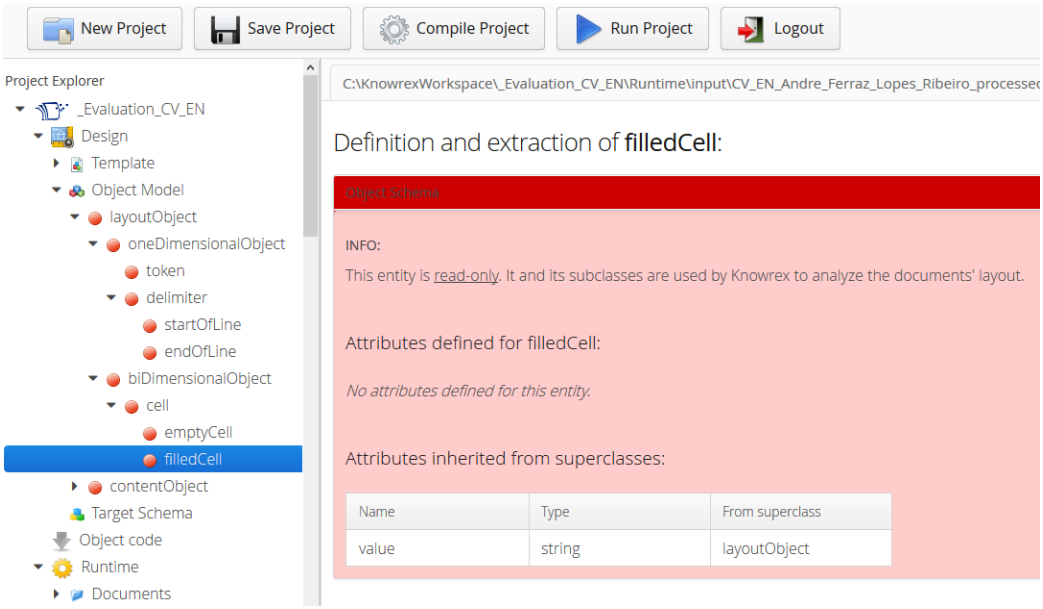


Figure 7.4: Layout objects of the ontology in a KnowRex project

Finally, the user can specify a sub-hierarchy of domain-specific classes. This part of the object model is fully editable; the user can add and remove sub-classes, define additional attributes for any class etc. (see Figure 7.6).

Arrangement of the Semantic Annotators In this step, the designer selects the annotators to be used to extract instances of a particular class, chooses classes that should be searched for, and configures each annotator: provides a mapping from the tool's output to the object model, and sets the tool's specific properties. In the case of Europass CV analysis, we have selected: StanfordNER, a custom annotator for recognizing e-mail addresses and dates, a dictionary-based annotator for recognizing skills defined in the European e-Competence Framework [1], and a label annotator based on pattern recognition that recognizes labels typical for Europass CV. Decisions about the arrangement of annotators are made experimentally. Sometimes, it is beneficial to use more than one tool for recognizing the same category. The resulting potential redundancy is not harmful, instead the recall of extraction may improve. With the ontology editor provide with KnowRex , one can simply select an annotator from available ones when editing particular object. This way, the output of this annotator will be automatically mapped to the selected ontology object.

[1] See http://www.ecompetences.eu/.

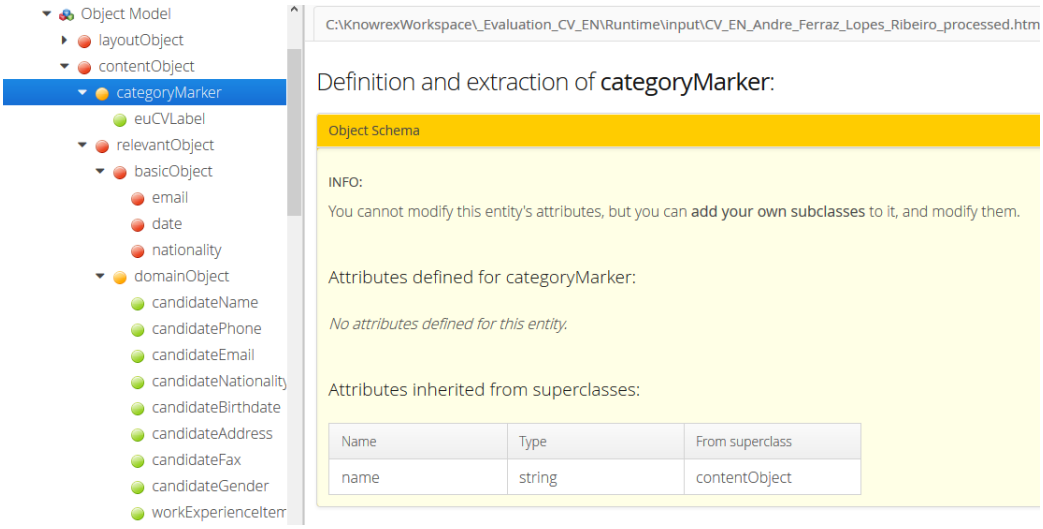


Figure 7.5: Refinement of a general "category marker" concept

Two-dimensional Document Analysis Knowing the context in which certain phrase appears is helpful for semantic information extraction. In some input data formats, e.g. pdf documents, the information about the structure is lost; while visible to human eye, it is not obvious for a machine. Thus, we need to recover the structure to obtain a meaningful representation of the input documents. To this end, this step configures an external *two-dimensional processor* and a *refinement module* inside KnowRex. As a two-dimensional processor, we have used Quablo (`http://www.quablo.eu/`) that can recognize a set of regular tables within a pdf document. The representation obtained from this tool is then improved by a special module that works with domain concepts, such as labels of the Europass template. The module produces improved structure, merging appropriate cells (for example, if a label spans across two cells, these cells will be merged). Moreover, one- and two-dimensional tokenizers (tools inside the KnowRex core) are used to identify the basic one- and two-dimensional objects of the document. In the end, we obtain a grid representation of the document that consists of two-dimensional objects (cells) containing one-dimensional ones (text fragments, delimiters). For a regular user, the basic task of this stage is to define the instances of (specification of) category markers as they will be used be the two-dimensional processor to improve the structure analysis (see the details of the methods in Chapter 4). More advanced settings of the processor are possible in a text mode.

Semantic Descriptors Specification While the semantic annotators identify single words or phrases as belonging to specific classes (producing the "leaves" of

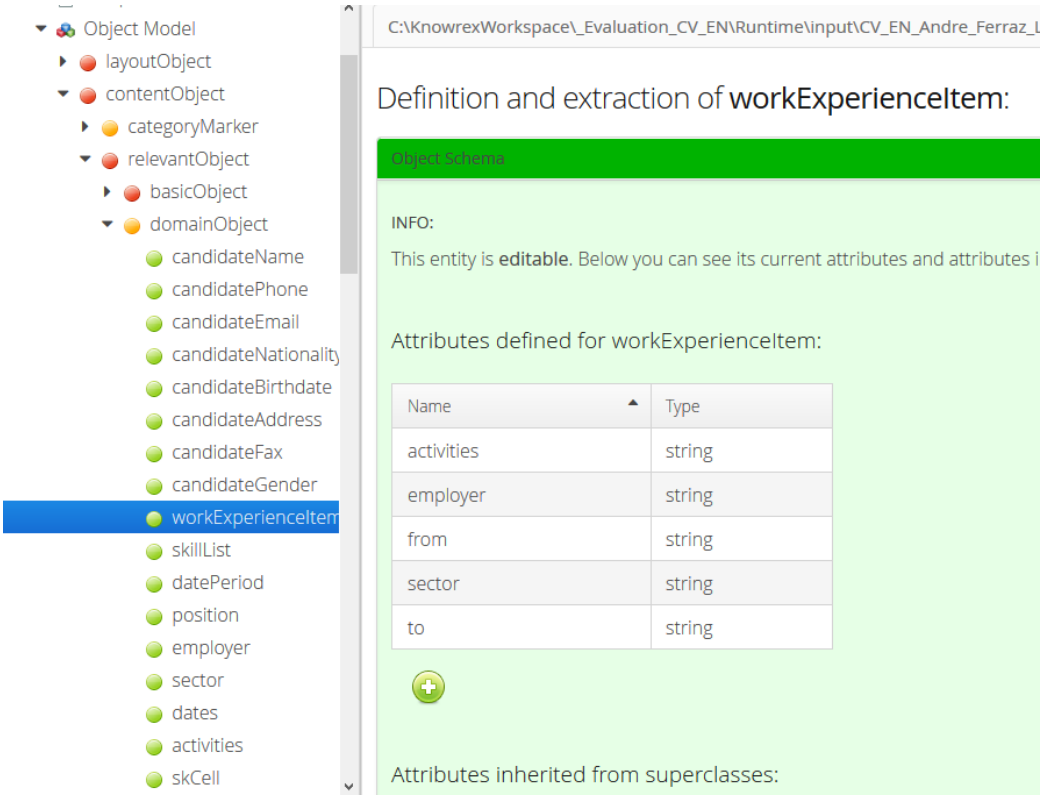


Figure 7.6: Fully editable domain-specific concepts in an ontology designed by a user for a particular KnowRex project

the object model), and the two-dimensional processing adds structure to the input, the semantic descriptors can combine and use the above information to build more complex objects. As explained in Chapter 6, semantic descriptors are rules that organize two-dimensional and one-dimensional objects into descriptions to extract additional information. This is done on several levels (cf. the examples given in Section 6.2). First, a designer should identify parts of the document that will help to localize other data portions. Descriptors can join several cells that appear in a document one after another (horizontally or vertically). This is useful, if we want to say that there exist a particular object, if there is a specific sequence of cells. We can also aggregate the concepts and attributes extracted by other semantic descriptors to build more complex ones. Within cells, we can create one-dimensional descriptors by using the operator "::". Finally, semantic descriptors may use the information about the placement of objects within the document (e.g. presence of a given object within specific section) to extract new objects that are not explicitly defined in text.

Defining a Mapping from Object Model to Target Schema The design phase in KnowRex is completed with the definition of a mapping from object model classes to the concepts of the target schema. This mapping, written in a form of Datalog rules, is used to automatically create a semantic view of the (structured) input documents during the runtime phase. In the head of rules, there are concepts from the target schema, and in the body – objects from the object model (and auxiliary objects such as candidate ID). For example, a mapping of the object model to the target schema may be as follows:

```
candidate(Id,N,S,P,E,A,G,Nt,D,L) :- ID:cv_candidate_id(Id),
           PI:personalInformation(N,S,A,P,E,Nt,D,G),
           CDL:candidateDrivingLicence(L).
workExperience(WExpId, Company, BusinessSector, Start, End)
        :- C:company(WExpId,Company,BusinessSector),
           WED:workExperienceDates(WExpId,Start,End).
```

When the design phase ends, all the decisions are saved in the configuration files of the system. The semantic descriptors are translated into logic rules.

7.2.2 Runtime phase

Once the design of the project is done, KnowRex can be run over a collection of input documents. The flow of the operations and the relations between the design and runtime phases may be observed in Figure 7.7. In the stage of the document analysis, first, a two-dimensional processor is used. Its output is then refined according to domain knowledge (specific labels, structure elements or keywords). This improved structure is analyzed by one- and two-dimensional tokenizers, tools hidden from a user, that identify the atomic one- and two-dimensional components of a document (tokens and cells). A logical fact base is obtained that represent the document as a two-dimensional "grid". KnowRex uses a two-dimensional representation of objects that helps localize them within the documents. Recall the definitions of the position relations from Chapter 5:

```
relation position(obj:ontologyObject, start:int, end:int).
  relation onePosition(obj:oneDimObject, start:int, end:int
    ).
  relation biPosition(obj:biDimObject, xstart:int, ystart:
    int, xend:int, yend:int).
relation belongsTo(obj:oneDimObject, obj2:biDimObject)
```

At the end of the two-dimensional processing stage, an ontological model of the document is obtained. It contains information about positions of the one- and two-dimensional objects within the document. For each two-dimensional object, a relation biPosition is added that specify the row and column on the document "grid", where the object appears, e.g.:

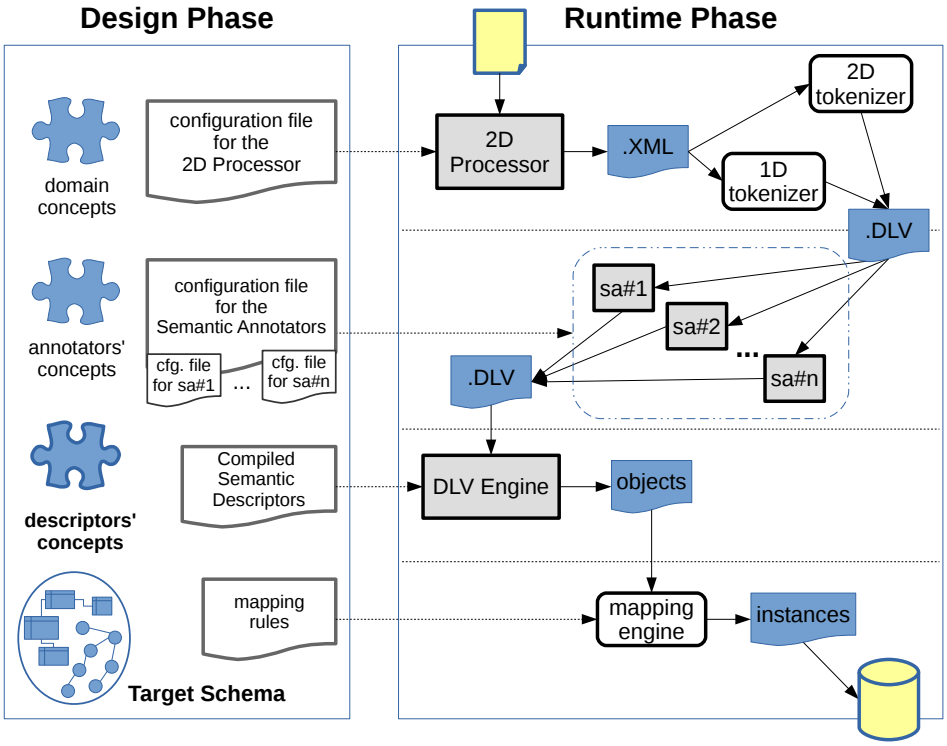


Figure 7.7: Runtime Phase of KnowRex system

```
filled19:filledCell('anna@w3.org'). biPosition(filled19,1,8,2,9).
```

For all one-dimensional objects (that are located inside the two-dimensional cells), two relations are added: belongsTo that identifies the containing cell by its id, and onePosition which denotes the position of the object within a cell:

```
tk123:token('manager'). one_position(tk123,0,6).
belongs_to(tk123,filled80). tk124:token('of').
one_position(tk124,6,7).belongs_to(tk124,filled80).
```

This representation is normalized i.e., the positions of blank spaces are omitted and the tokens follow one another. Such a representation is a reference for semantic annotators that may treat blank spaces differently.

Then comes the annotation stage, in which selected semantic annotators are run over the identified cells and label the parts of text as objects belonging to different classes (such as Places, Persons, IndustryTerms, etc.) The representation of the identified objects (new logic facts that carry information about the annotator that found the object) is added to the fact base, e.g.:

```
annS2:email('anna@w3.org'). one_position(0,10).
belongs_to(annS2,filled19).
```

Once the annotation stage is finished, the semantic descriptors which have been compiled into logic rules are executed over the facts representing the objects within a document. Each descriptor is transformed into a set of logical rules that first extract the portion of the document complying to the descriptor body, and then create a new object, specified in the descriptor head.

Each descriptor is internally represented as an automaton. After setting the initial configuration, each element of a descriptor is treated as a transition that allows to go from one state to the next one. The condition that one object must appear after another in a document is realized by checking the positions of the objects using `biPosition` and `onePosition` relations. The relation `belongsTo` checks the conditions expressed by the `CONTAINS` keyword. The attributes of the objects are passed between the rules by using variables.

For instance, the descriptor from Example 2 in Section 7.2.1:

```
<candidateEmail(E)> ::- {E:='';} <eucv_email_label_box()>
                <filledCell(X)> CONTAINS <email(X)> {E:=X;}
```

is translated into the following logic rules:

1. Extracting candidate email from the document:

```
init_conf_candidateEmail(0,"").
conf_candidateEmail(1, "", Xe, Ys, Xe_1, Ye) :-
    init_conf_candidateEmail(0,""),
    eu_cv_email_label_box(Id, Lc4),
    bi_position(Id, Xe, Ys, Xe_1, Ye).
conf_candidateEmail(2, Lc1, Xs, Ys, Xe_1, Ye) :-
    conf_candidateEmail(1, Gl3, Xs, Ys, Xe, Ye),
    filledCell(Id, Lc1), bi_position(Id, Xe, Ys, Xe_1, Ye).
conf_candidateEmail(3, Lc2, Xe, Ys, Xe_1, Ye) :-
    conf_candidateEmail(1, Gl3, Xs, Ys, Xe, Ye),
    filledCell(Id, Lc1), bi_position(Id, Xe, Ys, Xe_1, Ye),
    email(IdContains, Lc2), belongs_to(IdContains,Id).
```

2. Creating a new object for the object model, with its position:

```
aux_candidateEmail(AutoGen,Gl3, Xs, Ys, Xe, Ye) :-
    conf_candidateEmail(3,Gl3, Xs, Ys, Xe, Ye), AutoGen=#newID.

AutoGen : candidateEmail(Gl3) :-
      aux_candidateEmail(AutoGen,Gl3, Xs, Ys, Xe, Ye).
bi_position(AutoGen, Xs, Ys, Xe, Ye) :-
      aux_candidateEmail(AutoGen,Gl3, Xs, Ys, Xe, Ye).
```

These rules use the output of the extraction and create an object in OntoDLP, together with its one- or bi-dimensional position (and optionally, *belonging to* a cell, if it is a one-dimensional object).

Finally, the extracted objects are transformed into the instances of the semantic view (see Figure 7.8) with use of the mapping defined in the design phase. Tech-

Candidate table:

Id	Name	Surname	Phone	Email	Address	Gender	Nationality	Driving License
12	Weronika	Adrian	+39 123 456 7890	w.adrian@mat.unical.it	Via Pietro Bucci	F	PL	B
13	Anna	Falcone	+48 987 654 321	anna@w3.org	-	F	IT	A

WorkExperience table:

Id	Company	Business Sector	Start Date	End Date
1	The International School of Kraków	Education	2006	2008
2	HolidayCheck AG	Information technology	2008	2008
3	AGH University of Science and Technology	Science and education	2009	-
4	World Wide Web Consortium (W3C)	Information technology	2010	2012

CandWE table:

Id Candidate	Id Work Experience
12	1
12	2
12	3
13	4

Figure 7.8: Table output of the input documents

nically, this is done by additional logic rules that create instances for the target representation from the objects (in OntoDLP) of the object model.

7.3 Implementation principles

In this section, we describe how selected solutions have been realized during the development of the KnowRex framework. We show how the implementation principles we followed and the design choices we made enabled us to develop a functional, universal and flexible system.

7.3.1 Ontology-driven extraction

The process of Information Extraction in KnowRex is based on the ontology that covers all the classes expected to be encountered during processing a collection of documents from a given domain. Upon the extraction, the object model (OM, the ontology) is populated with the found objects. The objects may be recognized by semantic annotators, tokenizers, semantic descriptors and/or ontological inference (in particular by reasoning about subclasses and superclasses). Populating the whole object model requires appropriate configuration and use of a number

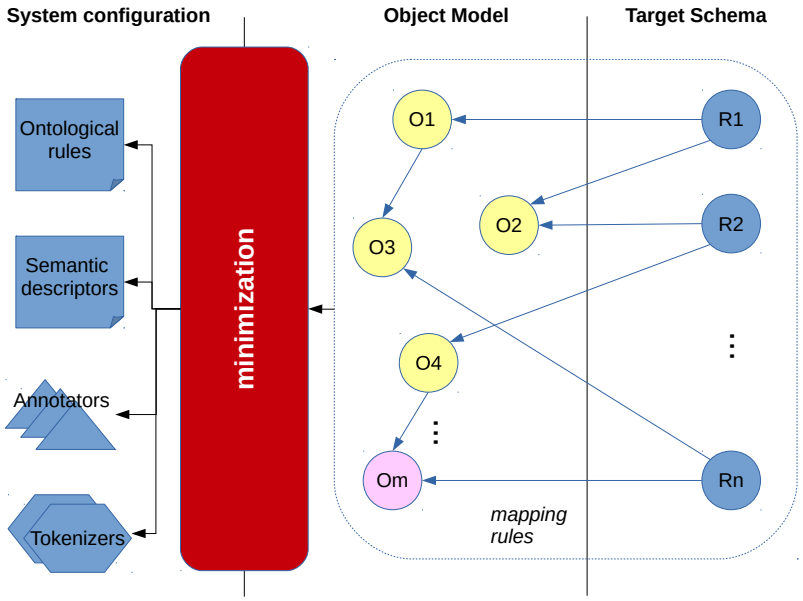


Figure 7.9: Minimization process for reducing the not needed extraction

of annotating tools, as well as rewriting and executing all the defined semantic descriptors and ontological relations compiled into logic rules.

Recall that the Information Extraction process follows the bottom-up approach. First, simple objects are recognized by annotating single words or phrases. This is done in the two-dimensional analysis phase by tokenizers, as well as in the annotation phase by semantic annotators. Then, based on these simple objects, more complex ones can be created, by executing the compiled semantic descriptors. Moreover, the ontological subsumption inference is applied to infer the instantiation of any superclasses of the found classes. To populate the target schema (TS), the mapping from the object model is used. The relations defined in the target schema depend on the objects from the object model.

Although this approach is intuitive and practical, it may result in generating facts irrelevant from the point of view of the target schema. Recall that the object model is independent from the target schema and even for the same ontology, a user may change the desired output. For a particular extraction task (which in practice means: for a selected set of target relations), it may happen that not all of the ontology objects must be instantiated. The cost of the extraction may be significantly reduced if the system configures only those tools and include only those descriptors and ontological relations that are necessary to obtain the objects needed for the selected relations. Therefore, we perform a minimization process

(see Fig. 7.9). In order to select only necessary objects and know which tools to configure and which rules to include, we analyze all the dependencies among objects, and build an appropriate "Dependency Graph". In the graph, the nodes are the objects that must be extracted, and the arcs are constructed based on the dependencies, both taxonomical and those related to the Information Extraction process.

Regarding the first group (taxonomical dependencies), the ontology contains the hierarchy of objects. The superclass-subclass relations map to logical rules of the form:

$$\forall x.\, subclass(X) \rightarrow superclass(X)$$

or in the DLV format: `superclass(X):- subclass(X).` To materialize the whole knowledge stored in the ontology, a great number of rules have to be added to the program. However, for particular IE task, not all of them are always needed. In fact, if we need an object of a class *a*, then we should find in the document either an instance of this class or an instance of one of its (direct or indirect) subclasses (see Fig. **??**). If we find an X that is an instance of a subclass of *a*, then using the rule presented above we can conclude that it is also an instance of the class *a*.

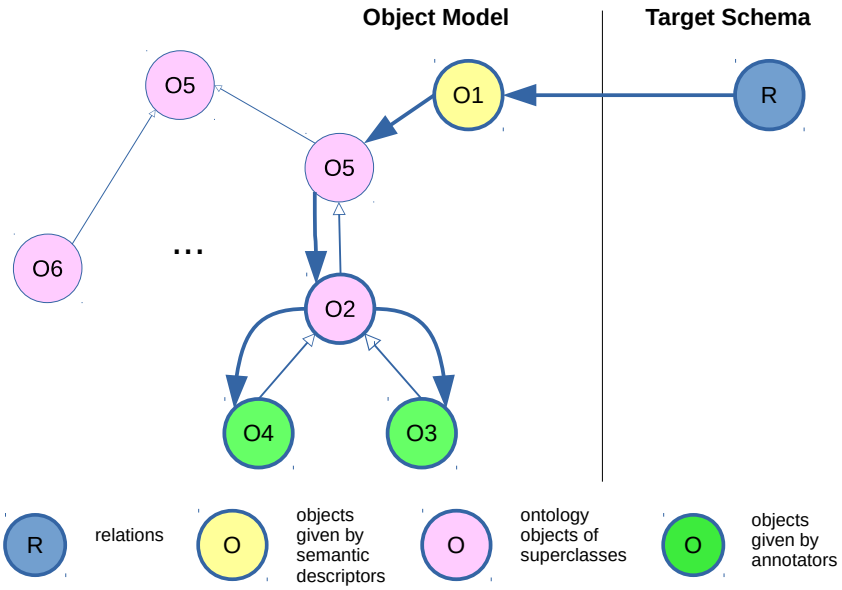


Figure 7.10: Selecting appropriate classes from the ontology

As for the second kind of dependencies – the ones resulting from the Information Extraction process, they are depicted in Figure 7.11. In particular, target

schema's relations depend on the objects extracted by the semantic descriptors as defined in mapping rules. Objects in semantic descriptors' heads depend on the objects in their bodies. The object in the descriptors' bodies may be: objects defined in other semantic descriptors' heads, one- and two-dimensional objects of the document "grid" that are recognized by tokenizers, objects recognized by annotators with use of appropriate mapping from annotator-specific entities or regular expressions to the ontology objects, or objects not directly recognized by any tool but being instances of superclasses of other classes.

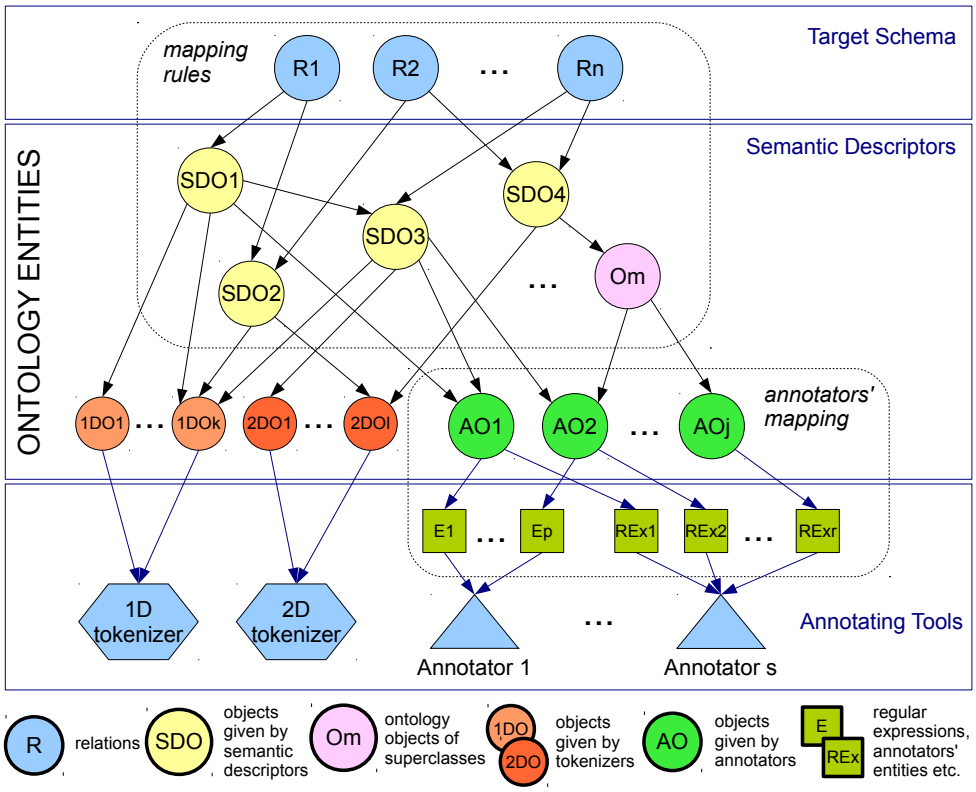


Figure 7.11: Dependencies between the target schema relations, object model objects, and tools' configuration

Let us now define the Dependency Graph, based on which we can determine which objects we need for the particular extraction task, and consequently which tools we have to configure and which logical rules we must include. Let us define:

- *MR* – set of mapping rules (from object model to target schema),
- *SD* – set of semantic descriptors,

- TR – set of taxonomic subclass relations represented as logic rules,
- If r is a rule or a semantic descriptor, then $H(r)$ denotes the head of this rule/descriptor and $B(r)$ – the body of it.

Let the structure of the dependencies be defined as a directed graph $G =< N, A >$, such as the set of nodes N is partitioned into two disjoint sets:

$$N = N_R \cup N_O$$

where

- N_R – set of the selected relations in target schema,
- N_O – set of objects defined in the ontology.

In turn, the set N_O is partitioned into the following (not pairwise disjoint) sets:

$$N_O = N_{DesObj} \cup N_{SupObj} \cup N_{DocObj} \cup N_{AnnObj}$$

where

- N_{DesObj} – set of objects defined in semantic descriptors' heads:

$$o \in N_{DesObj} \iff \exists d \in SD \text{ such that } o \in H(d)$$

- N_{SupObj} – set of objects defined as superclasses in the ontology:

$$o \in N_{SupObj} \iff \exists r \in TR \text{ such that } o \in H(r)$$

- N_{DocObj} – set of objects recognized by tokenizers (one- and two-dimensional elements of the document "grid")
- N_{AnnObj} – set of objects recognized by annotators

and the set of arcs is a subset of the following set A_{full}:

$$\begin{aligned} A \subseteq A_{full} = \{ & (r,s) \colon r \in N_R \wedge s \in N_{DesObj} \wedge \exists rule \in MR \text{ s.t. } r \in H(rule) \wedge s \in B(rule), \\ & (s,s') \colon s, s' \in N_{DesObj} \wedge \exists d \in SD \text{ s.t. } s \in H(d) \wedge s' \in B(d), \\ & (s,b) \colon s \in N_{DesObj} \wedge b \in N_{DocObj} \wedge \exists d \in SD \text{ s.t. } s \in H(d) \wedge b \in B(d), \\ & (s,a) \colon s \in N_{DesObj} \wedge a \in N_{AnnObj} \wedge \exists d \in SD \text{ s.t. } s \in H(d) \wedge a \in B(d), \\ & (o,o') \colon \exists t \in TR \text{ s.t. } o \in H(t) \wedge o' \in B(t) \} \end{aligned}$$

Construction of the graph for a particular case is presented next.

Constructing the dependency graph The dependency graph is built based on the selected ontological and IE-realted dependencies. Starting from selected target relations, we proceed to the objects needed to obtain them, and repeat it recursively up to the objects that do not depend on others but are directly recognized by some tools. For each object the following things must be considered:

1. Is the object defined by one or more semantic descriptor(s) (are there descriptors that have this object in their head)? If so, we should add all the objects in the bodies of these descriptors to the graph.

2. Is the object a superclass of some other objects? If so, the sub-classes objects should be added to the graph.

We proceed to the added nodes and repeat the analysis, expanding the graph. The algorithm for constructing the dependency graph is shown in Listing 3.

Traversing the graph Once the graph is constructed, it can be traversed in order to determine what should be configured to obtain the desired objects. For all the selected target relations, we have to follow the sub-graph containing their dependencies and analyze the objects found in this sub-graph. And exemplary dependency sub-graph for a single target relation can be observed in Figure 7.12.

The dependency graph is in fact a forest. Each tree has a root in a node representing a selected target relation. The trees may share some of the nodes, although in practice it is not frequent. Therefore, for the traversal we chose a slightly modified Breadth-First Search traversal that we start for all and only those nodes that belong to the N_R set. The algorithm is presented in Listing 4.

It may happen that a specific object is defined by more tools of the same method, e.g. two semantic descriptors, three annotators etc., or more than one method as depicted in Fig. 7.13. For instance, it may by defined by a semantic descriptor, but can be also recognized by a semantic annotator or may be inferred from the ontology, if a subclass has been recognized. Such redundancy is not harmful. On the contrary, it may improve the recall of the information extraction. Thus, for each node, all the possibilities must be explored i.e., we perform the following analysis over each node:

1. Is the object defined by one or more semantic descriptors? If so, add these descriptors to the program.

2. Is the object a superclass of some other objects? If so, add the (direct) subsumption rules to the program.

3. Is the object recognized by a semantic annotator? If so, consult the annotators' mapping and localize the mapping of the object to a particular annotator and its entity. Enable this annotator and this entity for the annotator.

Algorithm 3 Constructing the dependency graph.

Input: Set of desired relations: $DesRelations \subset TargetSchema$
Output: Dependency Graph $G =< N, A >$

Create empty queue: $DesObjects \leftarrow \emptyset$,
for all $r \in DesRelations$ **do**
  for all $rule \in MR$ such that $r \in H(rule)$ **do**
    for all o such that $o \in B(rule)$ **do**
      $N := N \cup \{o\}$
      $A := A \cup \{(r, o)\}$
      if $o \notin DesObjects$ **then**
        $DesObjects.enqueue(o)$
      end if
    end for
  end for
end for
while $DesObjects$ is not empty **do**
  $current \leftarrow DesObjects.dequeue$
  for all $descriptor \in SD$ such that $current \in H(descriptor)$ **do**
    for all o such that $o \in B(descriptor)$ **do**
      $N := N \cup \{o\}$
      $A := A \cup \{(current, o)\}$
      if $o \notin DesObjects$ **then**
        $DesObjects.enqueue(o)$
      end if
    end for
  end for
  for all $rule \in TR$ such that $current \in H(rule)$ **do**
    $N := N \cup \{o : o \in B(rule)\}$
    $A := A \cup \{(current, o)\}$
    if $o \notin DesObjects$ **then**
      $DesObjects.enqueue(o)$
    end if
  end for
end while

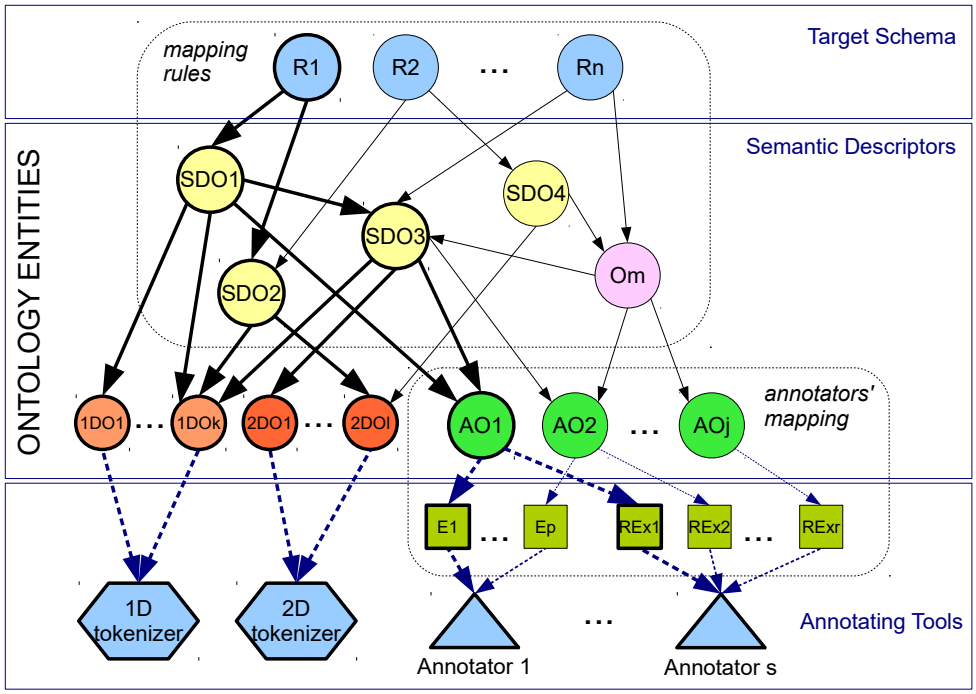


Figure 7.12: Exemplary relation and its dependency path

4. Is the object a layout object? If so, configure appropriate tool for two-dimensional analysis.

In order to avoid multiple analysis of the nodes, we mark the visited nodes.

The results of the minimization process are visible in the configuration of the project as the process itself is realized once the design is finished and a user select the 'Compile' functionality. First, only some of the available annotators are shown as activated. Second, the number of all the defined and the only needed descriptors are presented to the user.

7.3.2 Annotation engine

An important part of research (and subsequent implementation) was directed at the design of an annotation framework capable of integrating and specializing the annotation capabilities provided by the number of available tools. This is referred to as *annotator engine*, a processor capable of providing end users with various annotation tools through an interface level that simplifies and uniforms their use, as well as the integration of new annotation systems.

The choice of features that an annotator to integrate into the platform should

Algorithm 4 Traversal of the dependency graph with the analysis of the objects.

Input: The dependency graph $G =< N, A >$

Output: (1) Set of required mapping rules: *RecMR*, (2) set of required semantic descriptors: *RecSD*, (3) set of required taxonomical rules: *RecTR*, (4) set of pairs of the required annotators and their selected entities: *RecAE*, and (5) set of pairs of the required tokenizers and their selected entities: *RecTE*.

Create empty sets: $RecMR \leftarrow \emptyset$, $RecSD \leftarrow \emptyset$, $RecTR \leftarrow \emptyset$, $RecAE \leftarrow \emptyset$ $RecTE \leftarrow \emptyset$
Create empty queue: $Obj \leftarrow \emptyset$,
for all $rel \in N_R$ **do**
    $Obj.enqueue(rel)$
    while Obj is not empty **do**
        $current \leftarrow Obj.dequeue$ ▷ Beginning of the object analysis
        if $current \in N_R$ **then**
            for all $rule \in MR$ such that $current \in H(rule)$ **do**
                $RecMR.add(rule)$
            end for
        else ▷ N_R set is disjoint with the ontology objects set N_O
            if $current \in N_{DesObj}$ **then**
                for all $d \in SD$ such that $current \in H(d)$ **do**
                    $RecSD.add(d)$
                end for
            end if
            if $current \in N_{SupObj}$ **then**
                for all $d \in SD$ such that $current \in H(d)$ **do**
                    $RecSD.add(d)$
                end for
            end if
            if $current \in N_{DocObj}$ **then**
                for all $(tok, entity)$ such that $tok.entity = current$ **do**
                    $RecTE.add(tok, entity)$
                end for
            end if
            if $current \in N_{AnnObj}$ **then**
                for all $(ann, entity)$ such that $map(ann.entity) = current$ **do**
                    $RecAE.add(ann, entity)$
                end for
            end if
        end if
        Mark $current$ as analyzed ▷ End of the object analysis
        for all $(current, next)$ such that $(current, next) \in A$ **do**
            if $next$ is not marked as analyzed **then**
                $Obj.enqueue(next)$
            end if
        end for
    end while
end for

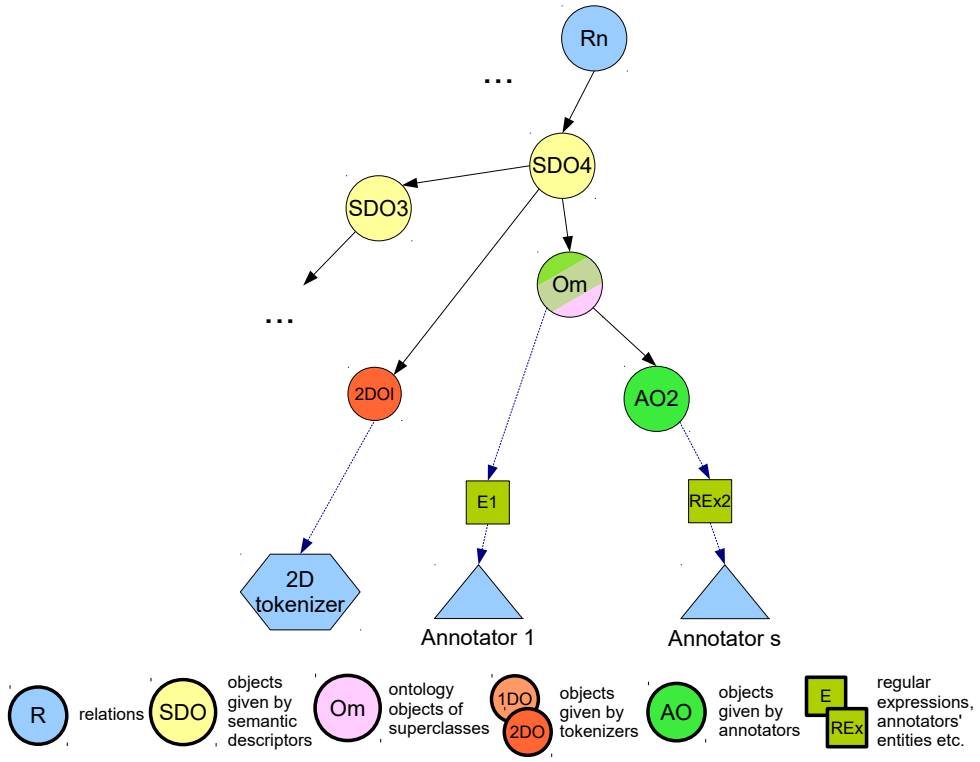


Figure 7.13: Dependencies of an object recognized with different methods

exhibit, was largely driven by the requirements stated for the implementation of the annotation suite itself. In fact, the criteria of choice have been oriented towards the "semantic" annotators, that is, the ones that are able to use ontological knowledge for the implementation of the annotation tasks. Among these a further selection was made on the basis of the specific features.

First of all, with respect to the accepted input type, annotators must be able to handle inputs in the form of files or, more generally, text strings. In addition, only the automatic configurable annotation systems were considered, which can interfere directly with a machine. Another discriminating element was related to the type of output provided as a result of the annotation process. In particular, in order to use the results of the annotation, it was considered indispensable to acquire (also in different formats) information about the positioning of the annotated text within the considered document (starting and end position of the annotated text).

Integrated annotators (discussed in Chapter 2) are available in KnowRex for a regular user (designer), who can select them and particular entities supported by them during the design phase. In addition, the KnowRex system provides

the expert user with an interface (API) that can specialize in implementing user-definable ad-hoc annotation mechanisms.

There is a particular case of a "label annotator", constructed ad hoc for the recognition of the typical labels in a given collection of documents. This annotator uses regular expressions for finding and extracting entities that match the given patterns. To make this mechanism more flexible and general, there is an additional configuration file for this annotator.

Chapter 8

Experimental Evaluation

In this chapter, we show how the solutions proposed in the thesis work on real world examples. The experimental evaluation we conduct is divided into parts that concentrate on individual tasks. First, we analyze the generation of a lexicon for a semantic annotator and then we focus on the layout analysis of a document. We exhibit some statistics and portray individual examples to illustrate the actions and effects of the solutions.

8.1 Automatic lexicon generation

In this section, our attention goes back to the semantic analysis and annotation. Specifically, we evaluate the automatic lexicon generation tools described in Chapter 2. For our tests, we have selected a number of different domains, to demonstrate different challenges we faced and addressed when solving the problem. Let us consider examples that illustrate different challenges of the problem.

Amusement parks in Europe Let us say that we want to explore fun parks in Europe, and we are looking for something like *Efteling* or *Gardaland* – two places we visited and liked. Let us put these two words as seeds. A representative entity network for them, limited to the hypernymy relations, consists of only 14 nodes and 13 edges. Moreover, for each words, only one sense has been found which for *Efteling* denotes a 'theme park', and for *Gardaland* – an "amusement park'. An optimal common ancestor is the 'amusement park' (represented by an entity $\langle\{\texttt{wn:8494231n}, \texttt{bn:00003695n}\}, \{\texttt{amusement park}, \texttt{fun park}, ...\}\rangle$), that is a superclass of the 'theme park'. The shared relation is $\langle\{\texttt{wd:P17}\}, \{\texttt{country}\}\rangle$ and the image set contains two units: *Italy* for *Gardaland*, and *Netherlands* for *Efteling*. We further expand a network for *Italy* and *Netherlands*, and we obtain that both are 'nations', 'states' and 'European countries' that share a relation 'con-

tinent' and have the common image of it, namely *Europe*. Moreover, they are both 'members of' a number of international organizations (for space limitations, we do not list all of them). They are also both 'located in time zone' *Central European Time*. As *Italy* and *Netherlands* are famous entities with numerous relations, we set the limit of our analysis limit to stop immediately after collecting their common properties with exactly the same values. After querying WebIsADb and evaluating new instances with BabelNet, we obtain the following expanded list of entities: *Portaventura*, *Euro disney*, *Tivoli gardens*, *Europa park*, *Legoland*, *Terra mitica*, *Parc Asterix*, *Disneyland Paris* and *Puy du fou*.

Cities or movies? Let us consider *Rome* and *Budapest* – words that have a lot of meanings. With 27 possible senses for *Rome* and 6 for *Budapest*, we obtain a representative entity network whose underlying graph (limited to hypernymy relations) consists of 248 nodes and 430 edges. For this quite a big structure, the sense detector still computes the optimal sense combinations very efficiently. In fact, we obtain 8 "best" combinations of senses, associated with their 11 optimal common ancestors. The disambiguated meanings include: capital cities (with optimal ancestors: 'national capital' and 'provincial capital', bands ('people', 'rock group'), films ('movie', 'work of art') and more. We can select the intended sense, or add a new seed to clarify our intentions. Let us add the word *Vienna* (19 senses) to the seed set. The resulting graph has 294 nodes and 558 edges. Here, the situation with best sense combinations is even more complicated, because for each seed, there are two examples of musical albums or singles. So, even though some senses are discarded, e.g. 'rock group', we obtain 9 best combinations of senses grouped under 7 optimal common ancestors. If we now add another seed, say *Zagreb* (2 senses), we get a slightly bigger graph, but we obtain a single best combination of senses denoting the capital cities with optimal common ancestors: {'national capital','provincial capital'}. Further analysis reveals that the seeds share the relation: 'country' and again, we expand the network for the countries related to the our seeds and obtain that all of them are in Europe.

The problem of entity set expansion is not a new topic. Over the last two decades, there has been work on gradually improving the classsification methods and expansion algorithms. More recently, the improvements concentrate on details such as: the quality of the seeds and their influence on the process, metrics for patterns and seeds evaluation, etc. The approaches described in literature come with some small examples, but a systematic comparison with the real tools is difficult, because they are not available on-line. To the best of our knowledge, there is no complete system for entity set expansion available for comparison now.

The last years brought the explosion of interest in the word representation,

in particular, the so-called *word embeddings* i.e., techniques that map words or phrases to vectors or real numbers, usually learned with neural networks over large corpus. The word representations may take into account the words around them in text context (the *bag-of-words* approach), but also some more information such as part of speech of the word, possible senses etc. The collection of word embedding are available online for use in different applications, including the entity set expansion. The interest in ESE is visible in the corporate world. Only a few years ago, such functionality was available in Google Docs spreadsheets, where one could write a few examples and ask the spreadsheet to expand the set. This feature has been now deactivated, and currently the underlying solution — which unfortunately is closed and private to the company— is used in the Google *Knowledge Graph* for information retrieval. Still the results are not fully satisfiable, depend on various factors: time, user settings etc. (it may be even hard to reproduce some of the results) and hard to compare methodically.

With our approach, we address the old problem in a moderm semantic way. Instead of relying on the lexical level, we utilize the online *semantic resources*, that were not available before, to build a better representation, based on semantic relations. The graph-based approach have been considered before, e.g., [71] proposed a graph structure that represented popularity of nodes and their potential to generate new seeds. Also the semantic resource we use are graph- or network-based. Our approach allows to leverage existing resources, and we believe that with the theoretical foundations and efficient ASP-based implementation of prototypes, that we already have, we can build, with further engineering effort, an integrated, configurable system.

8.2 Document layout analysis

This section is devoted to the analysis of documents in terms of their structure and layout (see Chapter 4). Recall that our label-content approach to document layout analysis consists in recognizing two-dimensional, tabular structures within a PDF file, and then refining the obtained tables with domain-dependent labels, specified by user. The result of the analysis is an ontological representation of one- and two-dimensional objects: cells, tokens and delimiters, equipped with their *positions* within a document, and a graphical representation of them that is helpful for further processing (e.g. formulating semantic descriptors). Working on a collection of over 80 curricula, we obtained promising results, and also noticed concrete limitations that we describe at the end of the section.

For testing our approach to the two-dimensional document analysis, we adopted the following assumptions:

The test requires: (i) defining the dictionary of “category markers” for a given

homogeneous collection of documents, and (ii) (optionally) configuring the parameters of the two-dimensional processor.

The input is: a collection of homogeneous documents in PDF format and the configuration settings.

The output consists in: two-dimensional representation of the documents (in a form of logical facts and a HTML document).

Test preparation, pre-processing: For each file in the test, we counted (manually) the number of labels present in a file that should be recognized, and the number of content cells that should be assigned to particular labels (i.e. correctly aligned within the refined structure). We evaluated both (i) the effectiveness of label recognition (taking into consideration correctly recognized cells – "true positives", incorrectly recognized ones – "false positives" and not recognized "false negative"), and (ii) the quality of the final two-dimensional representation, which follows the accuracy of aligning content cells to their respective labels.

Evaluation criteria: We decided to use the well known criteria of: precision, recall and F-measure. Given the number of true positives *tp*, false positives *fp* and false negatives *fn*, Precision is defined as follows:

$$Precision = \frac{tp}{tp + fp}$$

Recall:

$$Recall = \frac{tp}{tp + fn}$$

and the F-measure:

$$F = 2 * \frac{Precision * Recall}{Precision + Recall}$$

With respect to the labels, the meaning of true/false positives and negatives is standard; true positive means that a label has been recognized correctly (a given portion of text has been assigned the correct meaning), false positive – that a label was recognized, where it should not have been, and false negative – that a portion of text that should have been recognized as a label, was not. To unify the results of recognizing labels and refining the structure with domain concepts, we "overloaded" the meaning of true/false positives/negatives for content cells, such that *tp* denote correctly aligned cells, *fp* mean that a cell was aligned (additionally) with a label that should not be and *fn* denote the cells that were not aligned properly although their respective labels were correctly recognized.

We tested four collections of Curriculum Vitae in Europass standard. Two of them were in English and the other two in Italian. Moreover, we analyzed two versions of the Europass standard (older, "classical" and newer, "modern" view see Fig. 8.1). For both variants, we identified the *template* i.e., informal description

Personal information

First name(s) / Surname(s)	**Weronika Adrian**
Address(es)	Via Pietro Bucci
Telephone(s)	+39 123 456 7890
E-mail	w.adrian@mat.unical.it
Nationality	Polish
Gender	Female

Work experience

Dates	10.2009-present
Occupation or position held	Research & Teaching Assistant
Main activities and responsibilities	Research: Semantic Technologies, Logic Programming, Knowledge Engineering; Teaching: Unix/GNU/Linux, Algorithms and Data Structures, Operating Systems, Semantic Web
Name and address of employer	AGH University of Science and Technology
Type of business or sector	Science and education
Dates	8.2008-9.2008
Occupation or position held	Software Developer
Main activities and responsibilities	Web Programming (html, css, javascript, php, ajax, mysql)
Name and address of employer	HolidayCheck AG
Type of business or sector	Information Technology

(a) Classical *Europass* view

europass Curriculum Vitae

PERSONAL INFORMATION Weronika Adrian

Via Pietro Bucci, (Italy)
+39 123 456 7890
w.adrian@mat.unical.it
http://wtadrian.eu

Sex Female | Nationality Polish

WORK EXPERIENCE

10/2009–Present **Research & Teaching Assistant**
AGH University of Science and Technology
- Member of the GEIST Research Group (http://geist.agh.edu.pl), IEEE (http://www.ieee.org) and the Polish Artificial Intelligence Society (http://pssi.agh.edu.pl)
- Main research areas: Artificial Intelligence, Semantic Web, Rule-Based Systems, Logic Programming
- Teaching: Introduction to UNIX/GNU/Linux, Operating Systems, Knowledge Engineering, Algorithms & Data Structures, Creativity Workshop, Semantic Web Technologies.

08/2008–09/2008 **Software Developer**
HolidayCheck AG
Web Development (html, css, js, php, ajax, mysql, pdo)

09/2006–01/2008 **IT teacher**
The International School of Kraków

(b) Modern *Europass* view

Figure 8.1: Curricula in different variants of Europass standard

of the layout and structure of the documents. In particular, both variants share the two-column layout and some of the labels ('Personal Information', 'Work Experience' etc.). The older, classical layout, is more detailed, in a sense that almost all the content cells in the second column of the layout can be aligned with the respective labels from the first column (see Fig. 8.1a). Instead, in the newer version of Europass, some of the information is grouped together, e.g. personal information details, or work experience items ordered by dates of employment (cf. Fig. 8.1b). This has an impact on the further phases of information extraction. More specifically, the resulting two-dimensional representation will influence the development of semantic descriptors, which obviously will have to respect the two-dimensional representation.

Because a significant number of typical labels is shared among the collection, we decided to develop a single "label dictionary". It consists of around 60 label instances described with approx. 240 strings and regular expressions. This statistics reflects how we decided to describe particular labels, sometimes with exact string, other times with regular expressions, to keep the design phase as simple and intuitive as possible. This number could be easily minimized by using more optimized expressions. Note also that the dictionary covers two languages. For a single label in a fixed language, we estimate that one typically needs from 1 to 3 "descriptions" to keep the balance between readability and compactness.

The results of the conducted tests have been summarized in Tables 8.1 and 8.2.

#	Dataset (# of documents)	Test	Precision	Recall	F-measure
1	Dataset_CV_EN (16)	labels	96.25%	71.30%	80.60%
		content	100.00%	78.04%	86.31%
		TOTAL	**97.95%**	**74.05%**	**83.08%**
2	Dataset_CV_IT (50)	labels	98.35%	61.56%	74.25%
		content	99.76%	68.34%	79.27%
		TOTAL	**99.01%**	**64.73%**	**76.70%**
3	Dataset_NewCV_EN (51)	labels	97.99%	87.77%	91.14%
		content	97.07%	92.51%	94.10%
		TOTAL	**97.51%**	**89.87%**	**92.58%**
4	Dataset_NewCV_IT (7)	labels	100.00%	56.80%	69.00%
		content	99.40%	88.42%	92.59%
		TOTAL	**99.73%**	**70.83%**	**81.35%**

Table 8.1: Results of two-dimensional processing of documents.

The best results have been obtained for the dataset 'Dataset_NewCV_EN' (curricula in English, following the modern view). This may be related to the fact that although all the documents were acquired from the Web, for the modern format there exist generating them tools that produce very accurate documents, while the older format documents were probably edited manually. The 'CV_IT' dataset has the worst recall of all. Low recall in label recognition in curricula in Italian may be related to the presence of special characters e.g. "à'" that were not recognized by the two-dimensional processor. Dataset CV_IT has significantly many poorly recognized documents (for 30% of documents, the F-measure was lower than 70%), the small datasets 'CV_EN' and 'NewCV_IT' are more balanced (see Table 8.2).

#	Dataset name	F>90%	80%<F<90%	70%<F<80%	F<70%
1	Dataset_CV_EN	31.25%	25%	18.75%	25%
2	Dataset_CV_IT	20%	**34%**	16%	**30%**
3	Dataset_NewCV_EN	**74.51%**	7.84%	7.84%	9.80%
4	Dataset_NewCV_IT	28.57%	28.57%	28.57%	14.29%

Table 8.2: Results of two-dimensional processing of documents (2).

The tests revealed some technical limitations. In particular, the use of Quablo restricts the number of documents in a collection to 20 documents to be processed at once, otherwise, the tool throws and exception. Moreover, the nested tables, such as language competence tables in the old Europass format, are difficult to recognize and are either "flattened" during the analysis or not recognized at all.

As a possible solution to this particular problem,we improved the process of invoking the Quablo processor. Instead of setting the "region of interest" to the whole page, we first analyze the page layout to identify candidate regionsand then invoke Quablo for all the regions separately. The technique for this pre-processing has been described in a Master thesis related to the work presented in this thesis. We have presented the combined approach of purely geometrical and structural-semantic analysis in [4].

Chapter 9

Discussion and Conclusion

In this work, we addressed a few interesting problems of Information Extraction. We considered the document analysis from different perspectives: from the semantic analysis of single words and phrases, through the analysis of layout and structure of a document, up to analyzing typical elements shared within a collection of documents. The objective of this thesis was to propose a novel method of information extraction from a collection of similar documents that would combine and profitably use these different levels of analysis. The results of the work consists, from the one side, in a number of detailed solutions that address the particular tasks, and from the other, in presenting a framework that integrates all of them and a system that realizes the proposed method.

9.1 Summary of the results

We have analyzed the landscape of modern *semantic annotators* and in particular, evaluated their adaptability and possibilities to integrate their results. It appeared that despite the plethora of available tools, only a small subset can be used within a bigger framework, using well defined interface describe with a public API. The additional requirement we posed, to allow for mapping of the annotation results to a single ontological concept, further diminished the available annotators pool. Based on the analysis of the selected annotators, we have designed and developed and "abstract annotator" that is able to integrate and treat in a uniform way the results of multiple annotation tools. The interface we propose can serve as a guideline for annotators developers; extending our framework with a new tool is simple and comes down to implementing a few required methods.

While for general concepts, appearing in multiple domains (such as: 'place', 'person', 'e-mail', 'date' etc.) there exist annotation solutions, recognizing domain-specific concepts poses a greater challenge. The domain terms can either be not

"popular" enough to be taken into account, or they may be even assigned a different meaning within particular domain that does not agree with a "common" understanding of the word. This is why the best annotators in this field are those which work using *lexicons* of the domain terms, often supported with example instances. In this thesis, we proposed a new method for automatic lexicon generation based on integration of Web-accessible knowledge stored in the so-called *semantic resources*, such as WordNet, BabelNet or Wikidata. We described a formal model of an *entity network* by which we represent the integrated knowledge and showed how starting from a set of words, the network can be built and expanded, and, consequently, how the common category of the objects, represented by these seed words, can be formulated. We also tackled the problem of words ambiguity and proposed a graph-based solution the determines their "best" senses in the context, by introducing a concept of an *optimal common ancestor*. These newly designed techniques, supported by tools implemented in ASP, allows to build lexicons for any domains starting with a set of seed words. This approach is intuitive even for non-experts, as the user interested in extracting some category, has only to provide the system with some positive examples of the desired class.

The next problem addressed in this thesis, was the structure and layout analysis of complex documents. We concentrated on PDF format and analyzed the files both structurally – using table recognition techniques, and semantically – looking for domain-specific *labels*. As a result, we proposed a method that, given a PDF file, produces a "grid" representation of it, in which common (section) markers are aligned with the parts of text that are semantically associated to them. The approach has been realized, by extending the Quablo tool, produced by Exeura Srl. (the actual implementation of the algorithm has been carried out by the company as their work in the KnowRex project).

With respect to the ontological representation, we have analyzed the possibilities of ontological language and selected the formalism of OntoDLP to build a generic model of a document. In our approach, we incorporated both structure and content objects, and equipped them with properties that describe their spatial relations. We have combined the two layers: the one of a two-dimensional "grid" representation of a document that is made of one- and two-dimensional *layout objects* such as cells, tokens etc.; and the other in which the actual *content objects*, both *relevant objects* and *category markers*, appear.

On such representation we were able to build "semantic extraction rules", by means of the formalism of *semantic descriptors*. In this thesis, we have extended the capabilities of the descriptors language, introducing useful and intuitive constructs that allows a user to describe desired objects. The automatic translation of these descriptors into logic rules allows for an evaluation performed by a logic reasoner. In particular, we have applied the Answer Set Programming and used the DLV reasoning engine for this.

9.2 Future work

During the research, we have identified further problems and areas for improvement. The research directions for future work are summarized below:

1. **Common relations analysis for the lexicon generation.** Currently, the semantic relations of example (seed) objects, are treated all in the same way, without taking into account their specificity. However, they could be analyzed in more details, using resources such as DBPedia and Wikidata. In particular, the information about the relations' properties such as *transitivity*, *symmetry* etc. would allow for more intelligent processing of them. Moreover, the relations could be also clustered to avoid relying only on exactly the same relations when comparing objects.

2. **Improving the expansion and evaluation phases of the lexicon construction.** One idea for this point is the transition from a *set of properties* that describe the desired category to a human-readable and machine-ready *category description*, that could be used to query selected semantic resource for further instances. Finally, the new instances, instead of being simply accepted or rejected, could be scored based on the degree to which they agree with the intended category.

3. **Automatic identification of common labels within a collection of documents.** Currently, the "common labels" used in the structure recognition phase are given by user in a form of a label dictionary. In this dictionary, the user must explicitly write string patterns that represent the intended labels. Thus, it would be desirable for a system to automatically recognize repeating markers that appear in all or most of the documents and propose an initial set of the labels, together with their associated patterns, to the user.

4. **Tuning the framework implementation.** The execution of the system over collections of documents is relatively slow due to the number of different operations that must be performed consecutively. As future work, the speed of the execution should be increased. The interface of the framework could be further simplified, for instance the calibration of the input documents could be purely visual, instead of relying on manual parameters setting.

One of the most satisfying results of the thesis is the complete proposal of the Information Extraction framework that successfully integrates different perspectives of document analysis. This forms a basis for further improvements, that can be added to selected components, and contribute to the overall effectiveness of the system. New application domain and types of document collections may also inspire new refinements and solutions that will build on the generic approach.

Bibliography

[1] Brad Adelberg. NoDoSE – a tool for semi-automatically extracting structured and semistructured data from text documents. *SIGMOD Rec.*, 27(2):283–294, 1998.

[2] Weronika T. Adrian, Nicola Leone, and Marco Manna. Ontology-driven information extraction. *CoRR*, abs/1512.06034, 2015.

[3] Weronika T. Adrian, Nicola Leone, and Marco Manna. Semantic views of homogeneous unstructured data. In Balder ten Cate and Alessandra Mileo, editors, *Web Reasoning and Rule Systems - 9th International Conference, RR 2015, Berlin, Germany, August 4-5, 2015, Proceedings*, volume 9209 of *Lecture Notes in Computer Science*, pages 19–29. Springer, 2015.

[4] Weronika T. Adrian, Nicola Leone, Marco Manna, and Cinzia Marte. Document layout analysis for semantic information extraction. In *16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017)*. Submitted.

[5] Weronika T. Adrian, Marco Manna, Nicola Leone, Giovanni Amendola, and Marek Adrian. Entity set expansion from the web via asp. In *International Conference on Logic Programming (ICLP 2017)*, 2017. Accepted.

[6] Raghu Anantharangachar, Srinivasan Ramani, and S. Rajagopalan. Ontology guided information extraction from unstructured text. *CoRR*, abs/1302.1335, 2013.

[7] Apostolos Antonacopoulos, Christian Clausner, Christos Papadopoulos, and Stefan Pletschacher. Historical document layout analysis competition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1516–1520. IEEE, 2011.

[8] Emilia Apostolova and Noriko Tomuro. Combining visual and textual features for information extraction from online flyers. In *EMNLP*, pages 1924–1929, 2014.

[9] Gustavo O. Arocena and Alberto O. Mendelzon. WebOQL: restructuring documents, databases, and webs. *Theor. Pract. Object Syst.*, 5(3):127–141, 1999.

[10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, New York, NY, USA, 2007.

[11] Wolf-Tilo Balke. Introduction to information extraction: Basic notions and current trends. *Datenbank-Spektrum*, 12(2):81–88, 2012.

[12] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of IJCAI '07, Hyderabad, India*, pages 2670–2676, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[13] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, 2003.

[14] Sean Bechhofer, Leslie Carr, Carole Goble, Simon Kampa, and Timothy Miles-Board. The semantics of semantic annotation. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 1152–1167, 2002.

[15] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. Oiled: a reason-able ontology editor for the semantic web. *KI 2001: Advances in Artificial Intelligence*, pages 396–408, 2001.

[16] Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, Lynn Andrea Stein, et al. Owl web ontology language reference. *W3C recommendation*, 10(02), 2004.

[17] Kalina Bontcheva, Leon Derczynski, Adam Funk, Mark A Greenwood, Diana Maynard, and Niraj Aswani. Twitie: An open-source information extraction pipeline for microblog text. In *RANLP*, pages 83–90, 2013.

[18] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[19] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proc. of AAAI/IAAI, Orlando, Florida*, pages 328–334, 1999.

[20] Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari. I-DLV: the new intelligent grounder of DLV. *Intelligenza Artificiale*, 11(1):5–20, 2017.

[21] José Camacho-Collados, Mohammad Taher Pilehvar, and Roberto Navigli. A unified multilingual semantic representation of concepts. In *Proc. of ACL'15*, pages 741–751, 2015.

[22] Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Coupled semi-supervised learning for information extraction. In *Proc. of WSDM '10*, pages 101–110, 2010.

[23] Andrew Carlson, Justin Betteridge, Richard C Wang, Estevam R Hruschka Jr, and Tom M Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 101–110. ACM, 2010.

[24] Julien Carme, RÈmi Gilleron, AurÈlien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducer. *Machine Learning*, 66(1):33–67, 2007.

[25] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled F. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Trans. on Kn. and Data Eng.*, 18(10):1411–1428, 2006.

[26] Luying Chen, Stefano Ortona, Giorgio Orsi, and Michael Benedikt. Aggregating semantic annotators. *Proc. VLDB Endow.*, 6(13):1486–1497, August 2013.

[27] Laura Chiticariu, Yunyao Li, and Frederick R Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP*, number October, pages 827–832, 2013.

[28] Fabio Ciravegna, Alexiei Dingli, Jose Iria, and Yorick Wilks. Multistrategy definition of annotation services in melita. In *ISWC 2003 INTERNATIONAL SEMANTIC WEB CONFERENCE*. Citeseer, 2003.

[29] Fabio Ciravegna, Alexiei Dingli, Daniela Petrelli, and Yorick Wilks. User-system cooperation in document annotation based on information extraction. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 122–137. Springer, 2002.

[30] Valter Crescenzi and Giansalvatore Mecca. Grammars have exceptions. *Inf. Syst.*, 23(9):539–565, 1998.

[31] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of VLDB '01*, pages 109–118, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[32] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. Gate: An architecture for development of robust hlt applications. In *Proc. of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 168–175, 2002.

[33] James R. Curran, Tara Murphy, and Bernhard Scholz. Minimising semantic drift with mutual exclusion bootstrapping. In *Proc.PACLING'07*, pages 172–180, 2007.

[34] Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, Ramanathan Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John A Tomlin, et al. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the 12th international conference on World Wide Web*, pages 178–186. ACM, 2003.

[35] Francesco M Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Al-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[36] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[37] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open information extraction: The second generation. In *IJCAI*, volume 11, pages 3–10, 2011.

[38] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[39] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: A tool for collaborative ontology construction. *International journal of human-computer studies*, 46(6):707–727, 1997.

[40] Bettina Fazzinga, Sergio Flesca, Andrea Tagarelli, Salvatore Garruzzo, and Elio Masciari. A wrapper generation system for pdf documents. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 442–446. ACM, 2008.

[41] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[42] Sergio Flesca, Elio Masciari, and Andrea Tagarelli. A fuzzy logic approach to wrapping pdf documents. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1826–1841, 2011.

[43] Dayne Freitag. Information extraction from HTML: application of a general machine learning approach. In *Proc. of AAAI/IAAI, Madison, WI, US*, pages 517–523, 1998.

[44] Dayne Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2):169–202, 2000.

[45] Tim Furche, Georg Gottlob, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Little knowledge rules the web: Domain-centric result page extraction. In *Web Reasoning and Rule Systems*, volume 6902 of *LNCS*, pages 61–76. 2011.

[46] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[47] Michael R Genesereth, Richard E Fikes, et al. Knowledge interchange format-version 3.0: reference manual. 1992.

[48] Sergio Greco, Nicola Leone, and Pasquale Rullo. Complex: an object-oriented logic programming system. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):344–359, 1992.

[49] Ralph Grishman. Tipster architecture design document version 2.3. Technical report, Technical report, DARPA, 1997.

[50] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[51] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.

[52] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[53] Nicola Guarino. Formal ontology and information systems. In *International Conference On Formal Ontology In Information Systems FOIS'98*, pages 3–15, Trento, ITALY, 1998. Amsterdam, IOS Press.

[54] Ramanathan Guha and Rob McCool. Tap: a semantic web platform. *Computer Networks*, 42(5):557–577, 2003.

[55] Sonal Gupta. *Distantly Supervised Information Extraction Using Bootstrapped Patterns*. PhD thesis, Stanford University, 2015.

[56] Sonal Gupta and Christopher Manning. Improved pattern learning for bootstrapped entity extraction. In *Proc. of CoNLL'14*, pages 98–108, 2014.

[57] Sonal Gupta and Christopher D. Manning. Distributed representations of words to guide bootstrapped entity classifiers. In *Proc. of HLT-NAACL'15*, pages 1215–1220, 2015.

[58] Joachim Hammer, Héctor García-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, and Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. *SIGMOD Rec.*, 26(2), 1997.

[59] Siegfried Handschuh, Steffen Staab, and Fabio Ciravegna. S-cream—semi-automatic creation of metadata. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 358–372. Springer, 2002.

[60] T. Hassan and R. Baumgartner. Table recognition and understanding from pdf files. In *Proceedings of ICDAR '07*, pages 1143–1147, Washington, DC, USA, 2007. IEEE Computer Society.

[61] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[62] Alexander Hogenboom, Frederik Hogenboom, Flavius Frasincar, Kim Schouten, and Otto van der Meer. Semantics-based information extraction for detecting economic events. *Multimedia Tools and Applications*, 64(1):27–52, 2013.

[63] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the Web. *Inf. Syst.*, 23(9):521–538, 1998.

[64] Ruihong Huang and Ellen Riloff. Inducing domain-specific semantic class taggers from (almost) nothing. In *Proc. of ACL'10*, pages 275–285, 2010.

[65] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. Sensembed: Learning sense embeddings for word and relational similarity. In *Proc. of ACL'15*, pages 95–105, 2015.

[66] Jing Jiang. Information extraction from text. In *Mining Text Data*, pages 11–41. 2012.

[67] Aditya Kalyanpur, James Hendler, Bijan Parsia, and Jennifer Golbeck. Smore-semantic markup, ontology, and rdf editor. Technical report, DTIC Document, 2006.

[68] Soner Kara, ʻOzgʻur Alan, Orkunt Sabuncu, Samet Akpinar, Nihan K. Cicekli, and Ferda N. Alpaslan. An ontology-based retrieval system using semantic indexing. *Information Systems*, 37(4):294 – 305, 2012.

[69] Vangelis Karkaletsis, Pavlina Fragkou, Georgios Petasis, and Elias Iosif. Ontology based information extraction from text. In *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, volume 6050 of *LNCS*, pages 89–109. 2011.

[70] Michael Kifer and Georg Lausen. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *ACM SIGMOD Record*, volume 18, pages 134–146. ACM, 1989.

[71] Zornitsa Kozareva, Ellen Riloff, and Eduard Hovy. Semantic class learning from the web with hyponym pattern linkage graphs. In *Proc. of ACL'08*, pages 1048–1056, 2008.

[72] Trausti Kristjansson, Aron Culotta, Paul Viola, and Andrew McCallum. Interactive information extraction with constrained conditional random fields. In *AAAI*, volume 4, pages 412–418, 2004.

[73] Nicholas Kushmerick. Wrapper induction: efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, 2000.

[74] Alberto HF Laender, Berthier Ribeiro-Neto, and Altigran S da Silva. Debye–data extraction by example. *Data & Knowledge Engineering*, 40(2):121–154, 2002.

[75] John Lafferty, Andrew McCallum, Fernando Pereira, et al. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[76] Mario Lipinski, Kevin Yao, Corinna Breitinger, Joeran Beel, and Bela Gipp. Evaluation of header metadata extraction approaches and tools for scientific pdf documents. In *Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '13, pages 385–386, New York, NY, USA, 2013. ACM.

[77] L. Liu, C. Pu, and W. Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proc. of ICDE, San Diego, CA, USA*, 2000.

[78] Robert MacGregor. Retrospective on loom. *Information Sciences Institute, University of Southern California, Tech. Rep*, 1999.

[79] Marco Manna, Ermelinda Oro, Massimo Ruffolo, Mario Alviano, and Nicola Leone. The HιLεX system for semantic information extraction. In *Trans. on Large-Scale Data- and Knowledge-Centered Systems V*, volume 7100, pages 91–125. 2012.

[80] Cynthia Matuszek, John Cabral, Michael J Witbrock, and John DeOliveira. An introduction to the syntax and content of cyc. In *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, 2006.

[81] Andrew McCallum. Information extraction: Distilling structured data from unstructured text. *Queue*, 3(9):48–57, 2005.

[82] G. Mecca, P. Atzeni, A. Masci, G. Sindoni, and P. Merialdo. The Araneus Web-based management system. *SIGMOD Rec.*, 27(2):544–546, 1998.

[83] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[84] Qian Mo and Yi-hong Chen. Ontology-based web information extraction. In *Communications and Information Processing*, volume 288 of *CCIS*, pages 118–126. 2012.

[85] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Aut. A. and M.-A. Syst.*, 4(1-2):93–114, 2001.

[86] Anoop M Namboodiri and Anil K Jain. Document structure and layout analysis. In *Digital Document Processing*, pages 29–48. Springer, 2007.

[87] Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2):10, 2009.

[88] Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

[89] Eyal Oren, Knud Möller, Simon Scerri, Siegfried Handschuh, and Michael Sintek. What are semantic annotations. *Relatório técnico. DERI Galway*, 9:62, 2006.

[90] Óscar Corcho and Asunción Gómez-Pérez. A roadmap to ontology specification languages. In *Proceedings of EKAW '00*, pages 80–96, London, UK, 2000. Springer-Verlag.

[91] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proc. of EMNLP '09*, pages 938–947, 2009.

[92] Fuchun Peng and Andrew McCallum. Information extraction from research papers using conditional random fields. *Information processing & management*, 42(4):963–979, 2006.

[93] Jakub Piskorski and Roman Yangarber. Information extraction: past, present and future. In *Multi-source, multilingual information extraction and summarization*, pages 23–49. Springer, 2013.

[94] Borislav Popov, Atanas Kiryakov, Angel Kirilov, Dimitar Manov, Damyan Ognyanoff, and Miroslav Goranov. Kim–semantic annotation platform. In *International Semantic Web Conference*, pages 834–849. Springer, 2003.

[95] Lawrence Reeve and Hyoil Han. Survey of semantic annotation platforms. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 1634–1638, New York, NY, USA, 2005. ACM.

[96] Francesco Ricca and Nicola Leone. Disjunctive logic programming with types and objects: The dlv+ system. *Journal of Applied Logic*, 5(3):545–573, 2007.

[97] Ellen Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816. MIT Press, 1993.

[98] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proc. of AAAI '99 and IAAI '99*, pages 474–479, 1999.

[99] Arnaud Sahuguet and Fabien Azavant. Building intelligent Web applications using lightweight wrappers. *Data Knowl. Eng.*, 36(3):283–316, 2001.

[100] Sunita Sarawagi and William W Cohen. Semi-markov conditional random fields for information extraction. In *Advances in neural information processing systems*, pages 1185–1192, 2005.

[101] Luís Sarmento, Valentin Jijkoun, Maarten de Rijke, and Eugenio Oliveira. "more like these": growing entity classes from seeds. In *Proc. of CIKM'07*, pages 959–962, 2007.

[102] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. Active hidden markov models for information extraction. In *International Symposium on Intelligent Data Analysis*, pages 309–318. Springer, 2001.

[103] Julian Seitner, Christian Bizer, Kai Eckert, Stefano Faralli, Robert Meusel, Heiko Paulheim, and Simone Paolo Ponzetto. A large database of hypernymy relations extracted from the web. In *Proc. of LREC'16*, 2016.

[104] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden markov model structure for information extraction. In *AAAI-99 workshop on machine learning for information extraction*, pages 37–42, 1999.

[105] Anikó Simon, J-C Pret, and A Peter Johnson. A fast algorithm for bottom-up document layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):273–277, 1997.

[106] Mayank Singh, Barnopriyo Barua, Priyank Palod, Manvi Garg, Sidhartha Satapathy, Samuel Bushi, Kumar Ayush, Krishna Sai Rohith, Tulasi Gamidi, Pawan Goyal, et al. Ocr++: A robust framework for information extraction from scholarly articles. *arXiv preprint arXiv:1609.06423*, 2016.

[107] Marios Skounakis, Mark Craven, and Soumya Ray. Hierarchical hidden markov models for information extraction. In *IJCAI*, pages 427–433, 2003.

[108] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide. W3c recommendation, World Wide Web Consortium, 2004.

[109] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Mach. Learn.*, 34(1-3):233–272, 1999.

[110] Michael Thelen and Ellen Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 214–221. Association for Comp. Linguistics, 2002.

[111] Michael Thelen and Ellen Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proc. of EMNLP '02*, pages 214–221, 2002.

[112] Martin Toepfer, Hamo Corovic, Georg Fette, Peter Klügl, Stefan Störk, and Frank Puppe. Fine-grained information extraction from german transthoracic echocardiography reports. *BMC medical informatics and decision making*, 15(1):91, 2015.

[113] Melanie Tosik, Carsten Lygteskov Hansen, Gerard Goossen, and Mihai Rotaru. Word embeddings vs word types for sequence labeling: the curious case of cv parsing. In *VS@ HLT-NAACL*, pages 123–128, 2015.

[114] Victoria Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: science, services and agents on the World Wide Web*, 4(1):14–28, 2006.

[115] Maria Vargas-Vera, Enrico Motta, John Domingue, Mattia Lanzoni, Arthur Stutt, and Fabio Ciravegna. Mnm: Ontology driven semi-automatic and automatic support for semantic markup. In *Intern. Conf. on Knowledge Engineering and Knowledge Management*, pages 379–391. Springer, 2002.

[116] Kwan Y. Wong, Richard G. Casey, and Friedrich M. Wahl. Document analysis system. *IBM journal of research and development*, 26(6):647–656, 1982.

[117] Fei Wu and Daniel S Weld. Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127. ACL, 2010.

[118] Burcu Yildiz, Katharina Kaiser, and Silvia Miksch. pdf2table: A method to extract table information from pdf files. In *IICAI*, pages 1773–1785, 2005.

[119] Richard Zanibbi, Dorothea Blostein, and James R Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.