TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

Ph.D. Thesis

Ontology-Driven Information Extraction

Conducted for the purpose of receiving the academic title
'Doktorin der technischen Wissenschaften'

Supervisor
Silvia Miksch
Institute of Software Technology & Interactive Systems [E188]

Submitted at the Vienna University of Technology
Faculty of Informatics

by
BURCU YILDIZ
9926103

Vienna, March 27, 2007

In loving memory and dedication to Hasan Yildiz
my grandfather and best friend ...

Acknowledgement

All praise to God ...

... and special thanks to my supervisor Prof. Dr. Mag. Silvia Miksch for her support and encouragement during my research.

Further, to my second supervisor Prof. Dr. Dipl.Ing. Gernot Salzer for leading me towards the right direction at a critical phase of my research.

Also to my colleagues at the Information Engineering Group at the Vienna University of Technology for their constructive critiques and interesting questions.

Very special thanks to my family and friends ... To my father Dr. Mag. Ramazan Yildiz and my mother Humeyra Yildiz for raising me and caring for me for so many years now. To my sister Dr. Dipl. Ing. Canan Yildiz and my brother Dipl. Ing. Ahmet Yildiz for listening to my endless speeches about my ideas and for discussing them with me. And also to Filiz Arici, Kadriye Kaya, and Dipl. Ing. Alime Öztürk for their invaluable friendship.

Abstract

Since Berners-Lee proposed and started to endorse ontologies as the backbone of the Semantic Web in the nineties, a whole research field evolved around the fundamental engineering aspects of ontologies, such as the generation, evaluation and management of ontologies.

However, many researchers were curious about the usability of ontologies within Information Systems in 'ordinary' settings, performing 'ordinary' information processing tasks. To be used within Information Extraction Systems (IESs), we consider ontologies as a knowledge source that can represent the task specification and parts of the domain knowledge in a formal and unambiguous way.

In general, IESs use several knowledge sources (e.g., lexicons, parsers, etc.) to achieve good performance. Some also require humans to generate the extraction rules that represent the task specification of the system. However, often these resources are not at hand or the dependency on them lead to compromises regarding the scalability and performance of an IES. Therefore, we wondered whether ontologies formulated in a standard ontology representation language, such as OWL, are suitable enough to represent the task specification and also the domain knowledge to some extent, which the IES can utilise as its only knowledge resource. Our aim is to identify the limits of such an approach, so that we can conclude that things can only get better from that point onwards by using other resources whenever available.

In this thesis, we propose an extraction method that utilises the content and predefined semantics of an ontology to perform the extraction task without any human intervention and dependency on other knowledge resources. We also analyse the requirements to ontologies when used in IESs and propose the usage of additional semantic knowledge to reconcile them. Further, we propose our method to detect out-of-date constructs in the ontology to suggest changes to the user of the IES. We state the results of our experiments, which we conducted using an ontology from the domain of digital cameras and a document set of digital camera reviews. After performing the experiments with a different task specification using a larger ontology, we conclude that the use of ontologies in conjunction with IESs can indeed yield feasible results and contribute to the better scalability and portability of the system.

Zusammenfassung

Seitdem Berners-Lee Ontologien als das ”Rückgrat des Semantic Web” eingeführt hat, hat sich ein ganzes Forschungsgebiet um die fundamentalen Aspekte des Ontology Engineering gebildet, wie zum Beispiel die Erstellung, Evaluierung und das Management von Ontologien.

Viele ForscherInnen waren jedoch neugierig was die Verwendbarkeit von Ontologien in ’üblichen’ Informationssystemen angeht, die ’übliche’ Aufgaben zur Informationsverarbeitung durchführen. In Informationsextraktionssystemen (IE Systeme), zum Beispiel, können Ontologien als Wissensresourcen verwendet werden, welche die Aufgabenstellung und Teile des domänspezifischen Wissens in einer formalen und unzweideutigen Form darstellt.

Im Allgemeinen verwenden IE Systeme verschiedene Wissensressourcen (e.g., Lexika, Parser, etc.) um eine gute Performanz zu erzielen. Manche verlangen auch, dass die Extraktionsregeln, die dann die Aufgabenstellung des Systems repräsentieren, von Menschen erstellt werden. Jedoch sind diese Wissenressourcen nicht immer griffbereit oder die Abhängigkeit von ihnen führt zu Kompromissen bzgl. der Skalierbarkeit und Übertragbarkeit des Systems. Deshalb haben wir uns gefragt ob Ontologien, die in einer Beschreibungssprache wie z.B. OWL formuliert sind, als einzige Wissenressource des Systems verwendet werden können um brauchbare Ergebnisse zu erzielen.

In dieser Dissertation führen wir eine Extraktionsmethode ein, welche den Inhalt und die vordefinierte Semantik einer Ontologie ausnutzt um eine Extraktion zu ermöglichen, die keine menschliche Unterstützung oder andere Ressourcen benötigt. Wir analysieren auch die Anforderungen auf Ontologien die in IE Systemen verwendet werden sollen und die Verwendung von ergänzendem semantischen Wissen um diesen Anforderungen zu entsprechen. Weiteres, führen wir unsere Methode zur Ermittlung von out-of-date Konstrukten in der Ontologie ein um den BenutzerInnen letztendlich Vorschläge bzgl. allfälliger Änderungen in der Ontologie zu machen.

Contents

Chapter 1

Introduction

Well I left my happy home to see what I could find out
I left my folk and friends with the aim to clear my mind out

Well I hit the rowdy road and many kinds I met there
Many stories told me of the way to get there

On The Road To Find Out - Cat Stevens

The invention of computers made many things easier, for example to generate, save, and access data. The invention of the internet was 'the' step to share all this data with everyone around the world. Despite many benefits, this development gave rise to the problem of extracting relevant information out of the overwhelming amount of data we are facing on a daily basis.

The Artificial Intelligence (AI) community has been dealing with this problem for some time now. The research field that comprises all the work in this area is called Information Processing. One particular sub-field deals with the extraction of certain types of relevant information from mainly text documents, which is called Information Extraction (IE). 'What is relevant information?' is the first question that comes to ones mind immediately. Not only is this question hard to answer when complicated task specifications and domains are involved, it is even harder to communicate this answer to a computer. Ontologies, being explicit specifications of conceptualisations [Gruber, 1993] can be used here to provide Information Extraction Systems (IESs) with formal and computer-understandable representations of relevant information.

Ontologies gained more attention lately, as many researchers began to think that they can serve as the backbone of the Semantic Web. The Semantic Web is an extended form of the current Web with semantics attached to the content, to make it easier for humans and machines to locate information [Berners-Lee, 1999]. But the application areas of ontologies are not bound to the Semantic Web. They are knowledge bearing artifacts and hence, can be used in any application area where a domain of interest has to be conceptualised and communicated to an Information System (IS).

In this thesis we will analyse to what extent ontologies are suitable to drive the IE process and how a classical IES has to be augmented to enable it to handle an underlying ontology properly. Further, it has to be clarified whether there are different requirements to ontologies when they are going to be used for IE than for the Semantic Web. Within this context we will also analyse what kind of management issues regarding the ontology arise and what course of action would be appropriate to dissolve them.

This thesis is centered around some fundamental research questions, which are listed in the next subsection.

1.1 Research Questions

The focus in this thesis is on the use and management of ontologies when used in conjunction with Information Extraction Systems (IESs). This scenario yields to a set of research questions, which we will state here and address later in subsequent parts of this thesis. The research questions can be considered in two parts: the first part contains questions regarding how ontology usage may contribute to the quality of IESs, and the second part contains questions regarding maintenance issues of ontologies used within IESs.

Research questions regarding effects of ontologies on IESs

How can ontologies be utilised to address main challenges of IESs, such as performance, portability, and scalability?

Do ontologies offer the potential to facilitate the performance, scalability, and portability of IESs? If yes, how can they be concretely utilised for these purposes?

Is it possible to develop an unsupervised and automatic IE method, that utilises no other resource but an input ontology?

Are ontologies suitable knowledge resources for IESs that have no access to other resources, such as lexicons or linguistic processing modules?

Are there certain requirements to the content of ontologies when they are going to be used in conjunction with IESs? If yes, what are they and how can they be reconciled?

Do ontologies have to reconcile different requirements regarding their content when they are going to be used in different settings than the Semantic Web? What if they are going to be used in conjunction with IESs? What are the requirements in such a scenario and what are the concrete measures that have to be taken to reconcile them?

Research questions regarding the maintenance of ontologies within IESs

How to detect out-of-date ontological components in a domain of interest?

Is it possible to automatically detect out-of-date components in the conceptualisation represented by the ontology of an IES? Are automatic approaches sufficient or is human intervention required?

How to query/monitor the changes themselves?

In what form can suggested changes to the ontology be utilised by the system for its future decisions? What would be the best way to represent these suggestions to the user?

How to apply changes to the ontology? Automatically or manually?

Is it desired and feasible that the IES applies suggested changes to the ontology on its own? What kind of pitfalls are expected with such an automatic approach? To what extent could human intervention enhance the quality of this task?

1.2 Contributions

Besides providing answers to the stated research questions (see Section 1.1), the main contributions of this work can be listed as follows:

1. *Analysing the requirements regarding the use and management of ontologies when used in conjunction with IESs:* Using ontologies in IESs yield to additional requirements concerning the knowledge represented in an ontology. To reconcile these requirements, we propose the inclusion of additional knowledge about qualitative, temporal, and evolutional properties of ontological structures.

2. *A method for automatically extracting information from natural language text using an input ontology:* In general IESs use several knowledge sources (e.g., lexicons, parsers, etc.) to achieve good performance. Some also require humans to generate the extraction rules that represent the task specification of the system. However, often these resources are not at hand or the dependency on them lead to compromises regarding the scalability and performance of an IES. Therefore, we wondered whether ontologies formulated in a standard ontology representation language, such as OWL, are suitable enough to represent the task specification and also the domain knowledge to some extent, which the IES can utilise as its only knowledge resource. Our aim was to identify the limits of such an approach, so that we can conclude that things can only get better from that point onwards by using other resources whenever available. We propose an extraction method that utilises the semantics of an ontology to perform the extraction task without any human intervention and dependency on other knowledge resources.

3. *ontoX: An ontology-driven IES:* During the course of our research, we implemented the mentioned automatic extraction method in Java. The system also performs change detection using input data and the predicted behaviour of the ontological constructs over time. This IES is available at the projects section of our website[1].

4. *Motivating the use of ontologies in conjunction with IESs:* By presenting our method and our experimental results, we undermined our claims that ontologies can indeed be used for IE from natural language text. We conclude that ontologies can replace knowledge and domain engineers in many cases, because it is realistic to assume that the user of an IES can build an extraction ontology using existing ontology development frameworks, which do not require any knowledge about the underlying syntax and theoretical foundations of the used representation languages.

1.3 Publications

Parts of this thesis have been published in:

Yildiz B, Miksch S. *Ontology-Driven Information Systems: Challenges and Requirements.* In: Proceedings of the International Conference on Semantic Web and Digital Libraries (ICSD 2007), Bangalore, India, 2007.

Yildiz B, Miksch S. *Motivating Ontology-Driven Information Extraction*. In: Proceedings of the International Conference on Semantic Web and Digital Libraries (ICSD 2007), Bangalore, India, 2007.

1.4 Thesis Outline

This thesis is structured in two parts. In the first part we give an overview of the basic terms and on the state of the art in the fields of Ontology Engineering and Information Extraction. The second part consists mainly of the contributions of our research, the results of an thorough evaluation phase, and an analysis on possible future research directions.

Part I: Context and Related Work

Chapter 2 : Ontologies – consists of an introduction to the field of ontologies, with basic definitions and the characteristics of ontologies. Further, it contains brief descriptions of commonly used ontology representation languages and formal logical foundations on which most of them are based on.

Chapter 3 : Ontology Engineering – deals with particular phases of the ontology life cycle, such as generation, evolution, versioning, and evaluation. For each phase

[1] http://ieg.ifs.tuwien.ac.at

it defines the concrete task and points out main challenges along with some approaches that have been proposed so far to address them.

Chapter 4: Information Extraction – contains a detailed introduction to the research field of IE. Besides giving some definitions and a historical overview, it explains the main architecture of an IES and it introduces main approaches and challenges to IE.

Part II: Ontology-Driven Information Extraction

Chapter 5: Ontology-Driven Information Systems – explains the main benefits the use of ontologies can bring for ISs in general and IESs in particular. Furthermore, it contains an analysis of different requirements to ontologies when they are going to be used in such systems and proposes methods to reconcile these additional requirements.

Chapter 6: ontoX - An Ontology-driven Information Extraction System – contains a detailed description of our proposed method for IE using ontologies and also information about the architecture and functionality of ontoX, the implementation of our extraction method.

Chapter 7: Experimental Results – contains a detailed description of the evaluation setting for the proposed methods and detailed experimental results.

Chapter 8: Conclusion and Future Work – gives an overview of the lessons learned while doing this research and points out possible research directions for the future.

Part I

Context and Related Work

Chapter 2

Ontologies

So on and on I go, the seconds tick the time out
There's so much left to know, and I'm on the road to find out

Well in the end I'll know, but on the way I wonder
Through descending snow, and through the frost and thunder

On The Road To Find Out - Cat Stevens

Originated in early Greece, the term 'Ontology' is a branch of philosophy that deals with the nature and organisation of being. Philosophers like Plato and Aristotle dealt with this branch, trying to find the fundamental categories of being and to determine whether the items in those categories can said to 'be'. The proper naming of such items was questioned by Plato. In his opinion, item names in an optimal world, would refer in everybody's minds to one and only one thing.

The computer scientists community is in general not that much interested in philosophy, but the idea of having a means to represent fundamental categories of a particular domain to establish a common understanding between interaction partners was worth considering. So it happened that ontologies became very popular in Artificial Intelligence (AI) and Knowledge Representation. Here, an ontology has been seen as yet another "*engineering artefact*, constituted by a specific *vocabulary* used to describe a certain reality, plus a set of explicit assumptions regarding the *intended meaning* of the vocabulary words" [Guarino, 1998].

In this section, we will first define the basic terms with regard to ontologies and will give a short insight into possible benefits ontologies can bring. Further, we will explain the role of formal logics in representing ontological knowledge and then will give an overview of some well-known ontology representation languages.

2.1 Definition

The literature contains many, partly contradicting definitions of an ontology[1]. However, the best-known and most quoted definition in the AI community became the definition by Gruber [1993], which is also the one we will refer to in this thesis.

An ontology is an explicit specification of a conceptualization.

The term *conceptualisation* refers to an abstract model, a simplified view of a particular domain of concern. By defining the concepts, relations, and constraints on their use in a formal way, the conceptualisation becomes explicit.

The definition by Gruber has been thoroughly analysed by Guarino and Giaretta [1995]. First of all, they stated that it is crucial to distinguish between the 'Ontology' with a capital 'O' as a branch of philosophy, which deals with the nature and organisation of being; and the 'ontology' as a particular object (engineering artefact). They further specified seven possible interpretations of the term ontology and elucidate the implications of these interpretations:

- Ontology as a philosophical discipline
- Ontology as an informal conceptual system
- Ontology as a formal semantic account
- Ontology as a specification of a "conceptualization"
- Ontology as a representation of a conceptual system via a logical theory characterized by specific formal properties or only by its specific purposes
- Ontology as the vocabulary used by a logical theory
- Ontology as a (meta-level) specification of a logical theory

Interested readers are also referred to the article of Zúñiga [2001], who explained, in pretty much detail, the differences between the philosophical meaning of ontologies and the meaning in Information Systems (ISs) by giving a deep analysis and comparison on both, Gruber's and Guarino's views.

We will use the following formal definition of an ontology through the rest of this thesis, whenever referring to ontological components.

Definition 1 *An ontology is a tuple* $\mathcal{O} := \{\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, \mathcal{H}^{\mathcal{R}}, \mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{R}}, \mathcal{A}^{\mathcal{O}}\}$*, whereas*

• $\boldsymbol{\mathcal{C}}$*: represents a set of concepts.*

• $\boldsymbol{\mathcal{R}}$*: represents a set of relations that relate concepts to one another.* $R_i \in \mathcal{R}$ *and* $R_i \rightarrow \mathcal{C} \times \mathcal{C}$*.*

[1]Note that ontologies were introduced to get a clear and common view on things, and yet the community cannot agree on the definition of the term itself.

• $\mathcal{H}^{\mathcal{C}}$: *represents a concept hierarchy in form of a relation* $\mathcal{H}^{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{C}$*, whereas* $\mathcal{H}^{\mathcal{C}}(C_1, C_2)$ *means that* C_1 *is a subconcept of* C_2*.*

• $\mathcal{H}^{\mathcal{R}}$: *represents a relation hierarchy in form of a relation* $\mathcal{H}^{\mathcal{R}} \subseteq \mathcal{R} \times \mathcal{R}$*, whereas* $\mathcal{H}^{\mathcal{R}}(R_1, R_2)$ *means that* R_1 *is a subrelation of* R_2*.*

• $\mathcal{I}_{\mathcal{C}}$ *and* $\mathcal{I}_{\mathcal{R}}$: *represent two disjoint sets of concept and relation instances, respectively.*

• $\mathcal{A}^{\mathcal{O}}$: *represents a set of axioms.*

2.2 Characteristics of Ontologies

In general, ontologies consist of a set of concepts and a description of the relationships that hold between these concepts. But when examining them closer, one can see many differences between them. Therefore, many researchers have described characteristics of ontologies to be able to classify them. Although the names for these characteristics vary from researcher to researcher, they can be grouped based on the following two main characteristics.

According to their level of formality

The same conceptualisation can be defined using different ontology representation languages, which yield ontologies with different levels of formality. Uschold and Grueninger [1996] classified ontologies in four groups according to this characteristic:

- *Highly informal ontologies:* are ontologies that are expressed in natural language.
- *Semi informal ontologies:* are ontologies that are expressed in restricted and structured natural language.
- *Semi formal ontologies:* are ontologies that are expressed in a semi-formal defined language.
- *Rigourously formal ontologies:* are ontologies that are expressed in a rigourously formal defined language.

If ontologies are going to be used to share knowledge between humans, it is better to have an informal or highly informal ontology. But if the interaction partners are going to be computers, it would be better to provide them with at least a semi-formal ontology.

According to their level of generality

An ontology can contain information with different levels of detail. The classification by Gruber [1998] reflects this thought:

- *Top-level ontologies:* describe very general concepts, which are independent of a particular domain or task like as space, time, event, action, etc.
- *Domain ontologies:* contain a description of a vocabulary to a generic domain like as medicine or automobiles, etc.
- *Task ontologies:* contain a description of vocabulary to a generic task or activity, such as diagnosing or selling, etc.
- *Application ontologies:* are bound to both, a specific domain and a specific task.

2.3 Why to bother about Ontologies?

Ontologies are means for an agreement on the meaning of 'things' between interaction partners, either humans or computers. By committing to an ontology, an interaction partner declares that it is aware of the meaning of the vocabulary words in the ontology and that it will refer to this meaning only, guaranteeing consistency during interaction. To commit to an ontology should not require any changes in the working environments of the interaction partners, though.

Because ontologies can be used to share a specific conceptualisation among communication partners, their usage is theoretically beneficial wherever an agreement on the meaning of things is important. However, we can list some concrete benefits of ontology usage as follows:

- *Understanding:* An ontology can serve as a documentation, using which human beings can understand the underlying conceptualisation of a domain better.
- *Communication:* Ontologies can help interaction partners to communicate over a domain of interest in an unambiguous way. For this purpose, interaction partners can either send their respective ontologies to one another or commit to a shared ontology.
- *Inter-operability:* Computer systems can inter-operate in a consistent manner. This enables interaction partners to inter-operate across organisational and/or international boundaries.
- *Reuse:* There is no need to reinvent the wheel over and over again. This applies also to ontologies, because as knowledge bearing components, they can be reused by others. However, Menzies [1999] stated that the reuse of ontologies can be compared to the reuse of already programmed modules in software engineering. In many cases it takes so much time to understand

> and integrate a software module, that in that time 60% of it could have been written from scratch. Whether this applies to ontologies as well, is yet to be empirically analysed. Menzies statement could be true for relatively small ontologies, but in the case of very large ontologies that have been developed over many years (e.g., WordNet, MeSH) it could be hard to write them from scratch.

There are also several researchers that take the increasing use of ontologies critically, and doubt that ontologies will bring significant benefits as promised. A cost-benefit analysis should be done to clarify this issue. However, there is no empirical study available yet, because ontologies are not in use for so long.

2.4 Ontologies and Formal Logics

The term 'conceptualisation' is not new to the computer science and AI community. In fact, many different approaches to represent conceptualisations emerged in the past. Their differences are due to their target groups and consequently their provided knowledge representation constructs.

The focus of human centered approaches, for example, is on providing intuitive representations for humans utilising natural language or visual modelling primitives. Approaches that go one step further to provide computer-understandability as well as intuitive representations for humans, formalise the semantics of their modelling primitives. Most popular examples for this kind are Semantic Networks [Sowa, 1987]. They are based on graph theoretical semantics and consequently, their main components are concepts denoted by nodes, and relations (between concepts) denoted by directed arcs between nodes in the graph.

The level of formality provided by such 'human centered' approaches is generally considered as not enough to communicate conceptualisations to computers in an unambiguous way. So, attempts have been taken to use formal logics to represent conceptualisations.

Formal logic is the discipline that studies the principles of inference with formal content. Formal logics provide *formal languages* that have well-defined semantics, which is essential to express knowledge in a transparent and computer understandable way. This *well defined semantics* enable a common understanding of the expressed knowledge unambiguously, since the constructors of the language are defined in advance that leaves no space for different interpretations. *Automated reasoners* are needed to validate and maintain the expressed knowledge, and also to infer conclusions from already existing knowledge [Grigoris and van Harmelen, 2004].

Ontologies can benefit from these three properties of formal logic. In fact, all ontology representation languages are based on some kind of logic (e.g., predicate logic, frame-based logic, etc.). The differences in their underlying formal systems constitute the differences in their expressive power and consequently their level of complexity. On the one hand, one would wish to have a language with the highest expressiveness, which unfortunately comes only with a huge increase of the level of complexity. On the other hand, the level of complexity has a direct impact on the

computational properties of reasoning services. Because reasoning is most important during many phases of Ontology Engineering, the level of complexity has to be taken into account when developing a representation language, even if this yields to compromises w.r.t the expressive power.

Given an ontology $\mathcal{O}$ (see Definition 2.1), with an instance $i \in C_1$, and two concepts C_1 and C_2 we can describe the most important reasoning services in the context of ontology-based systems as follows:

- *Subsumption (also known as class consistency):* The task of determining whether a concept C_1 is a subconcept of a concept C_2. Many other more complex reasoning tasks can be formulated in terms of subsumption. Therefore, this service is considered as the central reasoning service in many logics.

- *Instance Checking (also known as class membership):* The task of determining whether an instance i is an instance of a concept C_1.

- *Consistency Checking:* The task of determining whether the represented knowledge in the ontology $\mathcal{O}$ is coherent or not. This task is important, because inferencing on an inconsistent ontology is meaningless. This service requires a formal definition of 'consistency', because it can differ from model to model.

- *Equivalence of Concepts:* The task of determining whether two concepts C_1 and C_2 are equivalent.

- *Querying:* The task of querying the contents of an ontology $\mathcal{O}$. For example, querying all instances of a particular concept C_1.

- *More specific services:* There are more specific services such as determining the least common subsumer to enable bottom-up ontology generation, etc.

These reasoning services differ in terms of their run-times depending on their underlying logical systems (e.g., some of them can be decided in polynomial time) [Donini *et al.*, 1996]. Therefore, it is important for representation language designers to analyse where and for what purposes their proposed representation language is going to be used and what reasoners can be provided for that language.

In order to be useful, reasoners have to be sound and complete. *Soundness* ensures that it is not possible to infer false knowledge with defined inferencing rules. *Completeness* ensures that all possible true statements can be derived from given knowledge with the defined inferencing rules. Unfortunately, not all ontology representation languages can be provided with sound and complete reasoners. In the next subsection, we will take a look at the most popular ontology representation languages and will examine, among other things, the reasoning support they have been given. Those languages are mainly influenced by two knowledge representation paradigms, namely Frame-based Systems and Description Logics. To understand the potentials and limitations of different languages, we have to understand their underlying logical foundations first.

Frame-based Systems

As the name implies, frame-based systems are centered around *Frames*. Frames were first introduced by Minsky [1975] as a data structure for representing domain knowledge. They have components called slots (in general attribute-value pairs) and slots in turn have components called facets that describe their respective properties. Slots and facets are used to state value restrictions on Frames.

Typical Frame-based systems provide constructs to organise Frames into hierarchies allowing multiple inheritance as well. In such hierarchies, classes can inherit from their super-classes' features, such as slot definitions and default values according to some inheritance strategy which can differ from system to system. The central inference mechanism in Frame-based systems is inheritance [Kifer *et al.*, 1995].

The semantics of Frames were specified only operationally and attempts for formalising their declarative part yield to the development of Description Logics.

Description Logics

Description Logics (DL) is a knowledge representation formalism that extends Frames with formal semantics and can be seen as a specialisation of first-order logic. As Frames, it is suitable to represent knowledge that is centered around classes (concepts), properties (roles), and objects (instances or individuals).

Based on the allowed constructors, different DLs can be defined, which are denoted by several literals (see Figure 2.1 [Baader *et al.*, 2003]). Besides well-defined syntax, the constructors have also formal, well-defined semantics.

ALC denotes the minimal description language. The more constructors a DL allows, the harder the reasoning tasks are to solve. Brachman and Levesque [1984] showed that for FL^- the subsumption problem can be solved in polynomial time, whereas adding the role restriction constructor makes it coNP-hard.

DL-systems have two components: an intensional component, called TBox; and an extensional component, called ABox. The TBox can be seen as a general schema for the concepts and individuals to be represented, whereas the ABox contains the individuals [Donini *et al.*, 1996].

2.5 Ontology Representation Languages

Depending on the particular task at hand we will need to represent different kinds of knowledge. Therefore, we will have to look for a language with just the right level of expressive power. Neither less nor more expressiveness would do us any good. With the former we would not be able to represent the knowledge we want, and with the latter we would have to face a much more higher complexity than necessary, making life (that is reasoning) harder than it already is. Therefore, the right choice of an ontology representation language is an important decision, as it inevitably affects all forthcoming phases of the ontology life-cycle.

Grigoris and van Harmelen [2004] state the following main requirements for ontology languages one should look for:

- well-defined syntax

Construct	Syntax	Language
Concept	A	FL_0
Role	R	FL_0
Intersection	C ∩ D	FL_0
Value restriction	∀ R.C	FL_0
Existential quantification	∃ R	FL^-
Top	⊤	FL^-
Bottom	⊥	FL^-
Atomic negation	¬A	AL
Negation	¬ C	C
Union	C ∪ D	U
Existential restriction	∃ R.C	E
Number restrictions	(≥ n R) (≤ n R)	N
Nominals	$\{a_1 \ldots a_n\}$	O
Role hierarchy	R ⊆ S	H
Inverse role	R^-	I
Qualified number restriction	(≥ n R C) (≤ n R C)	Q

S

Figure 2.1: Description Logics constructors

- well-defined semantics
- convenience of expression
- efficient reasoning support
- sufficient expressive power

In this section, we will give an overview of some ontology representation languages, all of which equipped with well-defined syntax and well-defined semantics. They differ in terms of their convenience of expression (because some of them have more human-friendly syntax than others), in terms of their expressive power, and consequently in terms of whether efficient reasoning support is available for them or not.

2.5.1 RDF and RDF(S)

The Resource Description Framework (RDF) cannot be considered as a language itself. It is a graph-based data model to represent knowledge that can be stated in form of predicate-subject-object triples, which are called statements and where the predicate relates the subject to an object [Grigoris and van Harmelen, 2004]. A typical fact that could be stated as a predicate-subject-object triple is:

Prof. Baron is lecturer of the Algebra course.

Since the model is language independent (labelled directed graph) such a statement can either be represented visually (see Figure 2.2) or using any computer understandable language (in general an XML based syntax is being used for that purpose).

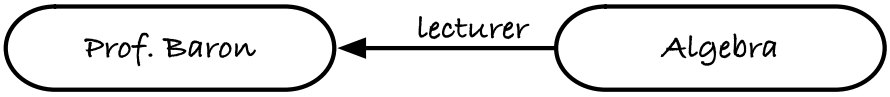

Figure 2.2: Visual representation of a typical RDF statement

What distinguishes RDF from other formalisms is its ability to make statements about statements. This is called *reification* and it is a very powerful means to represent knowledge. A typical example would look like:

The curriculum states that Prof. Baron is lecturer of the Algebra course.

However, RDF cannot be used for generating ontologies, as it does not offer any other construct then binary predicates [Grigoris and van Harmelen, 2004]. Therefore, RDF Schema (RDFS) has been introduced as an extension to RDF, which provides additional constructs with fixed semantics. Using these constructs, concepts and properties can be defined and organised as hierarchies to support inheritance.

RDF Schema is considered as a primitive language, because the kind of knowledge that can be represented using these constructs is very limited. For example it is not possible to state cardinality restrictions on properties or the disjointness of classes, etc. But, there are several ontology representation languages that can be seen as extensions of RDF Schema, such as DAML+OIL.

2.5.2 OIL, DAML-ONT and DAML+OIL

The Ontology Inference Layer (OIL) is a language that is based on three roots: Frame-based systems, Description Logics, and several Web standards [Fensel *et al.*, 2001]. The idea behind its development was to provide a language with well-defined formal semantics (Description Logics) that is highly intuitive for humans (Frame-based systems) and has a proper link to existing Web languages such as XML and RDF [Horrocks *et al.*, 2003].

DAML-ONT is another ontology representation language which is tightly integrated with RDF Schema, extending it with constructors from Frame-based representation languages.

Because of the same objectives DAML-ONT and OIL shared with each other, it was decided to combine the efforts, which resulted in the DAML+OIL language. DAML+OIL is equivalent to a very expressive DL (*SHIQ* DL) with the addition of

existentially defined classes and datatypes. This equivalence implies that subsumption reasoning is decidable in DAML+OIL. It largely discards the Frame-based structure of OIL [Horrocks *et al.*, 2003].

2.5.3 OWL

The Web Ontology Language (OWL) is a markup language for publishing and sharing ontologies on the Web. It is derived from DAML+OIL and is therefore also based on Description Logics.

The OWL working group claim that a language that fulfills all requirements for ontology representation languages is unobtainable. Therefore, they decided to define OWL as three different sublanguages, each aiming to reconcile different subsets of those requirements [Grigoris and van Harmelen, 2004]:

- OWL Full: uses all knowledge representation constructs of OWL without any restriction. It is fully upward compatible with RDF and is so powerful that it is undecidable.
- OWL DL: is a sublanguage of OWL Full corresponding to a description logic. It restricts how the constructors of OWL and RDF may be used and is therefore no longer fully compatible with RDF. However, these restrictions make efficient reasoning possible, which is often more important than compatibility with RDF.
- OWL Lite: is a syntactic subset of OWL DL excluding the use of enumerated classes, disjointness statements, and arbitrary cardinality. These restrictions make OWL Lite easier to understand and easier to implement.

Chapter 3

Ontology Engineering

I listen to the wind come howl, telling me I have to hurry
I listen to the robin's song saying not to worry

So on and on I go, the seconds tick the time out
There's so much left to know, and I'm on the road to find out

On The Road To Find Out - Cat Stevens

Ontology Engineering (OE) can be defined as the research field that comprises all phases of an ontology life cycle. The life cycle of an ontology can be considered roughly in terms of the following tasks: design, generation, evaluation, and management, which involves the whole life cycle. Although this life cycle can be refined by dividing the tasks further in subtasks, we will examine them under the mentioned basic rubrics. We will describe what constitutes these tasks, and point out the research efforts taken so far to address the difficulties related to them.

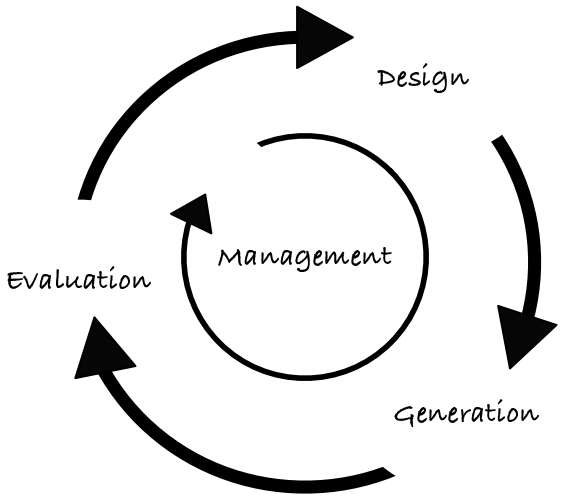

Figure 3.1: Ontology Life Cycle

3.1 Ontology Design

Since the AI community showed interest in ontologies, many ontologies have been designed and implemented. However, detailed documentations concerning the generation process of these ontologies are rare. This makes it harder to derive a standard methodology for building ontologies. Such a standard would be helpful for both, beginners and advanced ontology builders. For beginners, it would serve as a step by step 'how-to' description, whereas for advanced researchers it would serve as a check-list or guideline to improve the quality of their ontologies.

Gruber [1993] proposed some design criteria to guide the development process of ontologies for knowledge-sharing purposes. He stated that these criteria cannot be seen as guidelines for ontology generating theory, whereas more knowledge about the design process of ontologies would make it possible to evolve these criteria to working design principles:

- *Clarity:* The definitions of concepts and relations should be formulated in an objective and clear manner. If possible, the definitions should be stated as logical axioms.
- *Coherence:* The ontology should be internally consistent.
- *Extendibility:* It should be easy to extend the ontology with new ontological components (i.e. concepts and relations).
- *Minimal encoding bias:* The conceptualisation should be generated on a knowledge-level without depending too much on a particular symbol-level.
- *Minimal ontological commitment:* The ontology should only define necessary terms for the intended purpose, in order to left free space for further specialisations and instantiations.

Uschold and King [1995] took a further step and proposed a skeleton methodology for building ontologies. They stated that a comprehensive methodology should include the following stages, and additionally a set of techniques, methods, principles and guidelines to perform the required actions in these stages:

- Identifying purpose and scope
- Building the ontology
  - Ontology capture
  - Ontology coding
  - Integrating existing ontologies
- Evaluation
- Documentation

The *identification of the purpose and scope* of an ontology is an essential step, because it has a great impact on the rest of the generation process. If the purpose is not clear, you may end up with an ontology that does not fit your requirements. And if the scope is not clearly stated, you may easily miss the right level of generality.

In order to build the ontology, you first have to *capture your ontology*, that is to capture the information you like to have in your ontology. This information includes the key concepts and relations of the domain, unambiguous definitions of these concepts and relations, and terms to refer to these concepts and relations.

The identification of concepts and relations in a domain should follow also some method. You can either start by identifying the most specific concepts and then group them in categories (bottom-up) or you can start from the most general concepts and then build the underlying categories (top-down). A third way was proposed by Uschold and Grüninger [1996], that is to start with the most important concepts first, and then defining higher concepts (middle-out). Whereas the level of detail the ontology will have at the end is not predictable with the bottom-up and top-down approaches, the middle-out approach guarantees that the produced ontology will have at least the most relevant components you want.

Once you have captured the information you want in your ontology, you have to *code the ontology* in a formal language. Thus, you have to choose an ontology representation language (see Section 2.5), whereas the decision might not be so easy, because of the many languages that exist, each with its own advantages and disadvantages. The identified purpose and scope of the ontology will be helpful at this stage.

Ontologies as knowledge bearing artifacts, serve as a backbone module in most of their application areas. The quality of an ontology, thus, affects the performance of the whole application. This makes it necessary to *evaluate your ontology* in order to increase the acceptance of it by a large community. For more detailed information about the research field of Ontology Evaluation, you are referred to Section 3.4 .

Often ontologies are hard to understand by human beings who were not involved in the generation process. This is mainly because ontologies are being coded in a formal language rather than in natural language and because many of them are about not so well-known domains where the terms used to refer to concepts and relations are hard to understand as well (e.g. medical ontologies with 'very' Latin disease names, etc.). Therefore, a *documentation of your ontology* would help interested people immensely in overlooking and understanding the ontology better.

3.2 Ontology Generation

The efforts that have been taken so far to formulate a standard methodology for ontology generation imply that it is not easy to build an ontology. According to the means used for the generation process, we can differentiate between the manual, the semi-automatic, and the automatic approach.

With the *manual approach* an ontology engineer builds the ontology by hand. It is time consuming, but the involvement of a domain expert guarantees that the ontology itself and its level of granularity is most probably correct for the aimed

purpose. Today, many tools to aid ontology engineers during the ontology generation process exist (e.g., Protégé, OntoEdit, etc.). They can help with the technical part of Ontology Generation, because using such a tool, an ontology engineer does not have to bother about the underlying formal language anymore and syntax errors can be avoided as well. However, we cannot assume that the quality of the conceptualisations themselves can be improved by the mere fact that such a tool is being used, unless the tool provides on-the-fly reasoning services to check the consistency at run time or services to spot redundancies in the ontology.

With the *semi-automatic approach* a computer system assists the ontology engineer by recommending the addition of new ontological components depending on some sort of analysis on domain related data. The ontology engineer can then accept or discard these recommendations or can add components by herself if considered necessary.

With the *automatic approach* there is no human intervention at all. The system extracts ontology components from a data corpus with domain related data and builds the ontology. Whereas it is certainly less time consuming, it may lead to quality losses, because it is not guaranteed that all relevant concepts of the domain really occur in the corpus or that the extraction algorithms are good enough to extract all of the relevant components correctly.

The selection of related work in this subsection does not contain manual approaches and approaches for learning from other sources than natural language text, but only automatic and semi-automatic approaches to Ontology Learning from natural language text.

3.2.1 Some Clarifications

There is a lack of consensus upon the basic terms related to the generation process. That is why we have used the more general term 'Ontology Generation' as the heading of this section. we will state our interpretation of those terms before addressing the main techniques that are available to approach them. It should be stated beforehand, that one can differentiate between the intensional part (e.g., concepts and relations) and the extensional part of an ontology (e.g., instances of concepts and relations).

- *Ontology Learning* is the task of learning intensional components of an ontology ($\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, \mathcal{H}^{\mathcal{R}}, \mathcal{A}^{\mathcal{O}}$ as in Definition 2.1). For example to learn that the domain contains concepts 'Camera' and 'Digital Camera' and that there is a hierarchical relation between these two concepts ('Digital Camera' is subconcept of 'Camera').

- *Ontology Population (also known as Ontology Enrichment)* is the task of extracting and assigning extensional knowledge to the intensional components of an ontology ($\mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{R}}$ as in Definition 2.1). For example to learn that 'Canon A430' is an instance of the concept 'Digital Camera'.

- *Ontology Extension* is the task of adding intensional components to an already existing ontology ($\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, \mathcal{H}^{\mathcal{R}}, \mathcal{A}^{\mathcal{O}}$ as in Definition 2.1). For example to

learn that the domain also contains a concept 'Film Camera' and that this concept is also a subconcept of 'Camera'.

In the rest of this section, we will address the research fields of Ontology Learning and Population, discarding Ontology Extension, which can be seen as a derivation of the former two.

3.2.2 Ontology Learning and Population

As mentioned before, Ontology Learning is about acquiring intensional knowledge, whereas Ontology Population is about enriching the ontology with extensional knowledge, that is with instances of concepts and relations defined by the ontological components in the intensional part of the ontology. Many workshops dedicated to the field of Ontology Learning alone (e.g., ECAI 2000, IJCAI 2001, ECAI 2002, ECAI 2004) indicate that it is a very active field of research. Unfortunately, it is not yet agreed upon what exact tasks comprise Ontology Learning.

According to Maedche [2002], the Ontology Learning process comprises four phases (see Figure 3.2): import/reuse, extract, prune, and refine. To *import* an existing ontology can cause several problems, because the available ontology might contain different knowledge representation constructors than the ontology model of the learning framework. A learning framework can either simply ignore such constructors or define importing strategies to determine the way of how to transform them. Either way will yield to an ontology for which it cannot be guaranteed anymore that the imported knowledge is identical to the original. The *extraction* phase is the phase where the actual learning happens. This phase requires a description of the kind of knowledge that should be learned from the sources and appropriate algorithms that can identify and extract such knowledge. In most cases the resulted ontology will not be tailored for the intended particular application, so it will inevitably have to *pruned* be and *refined* according to the intended task at hand.

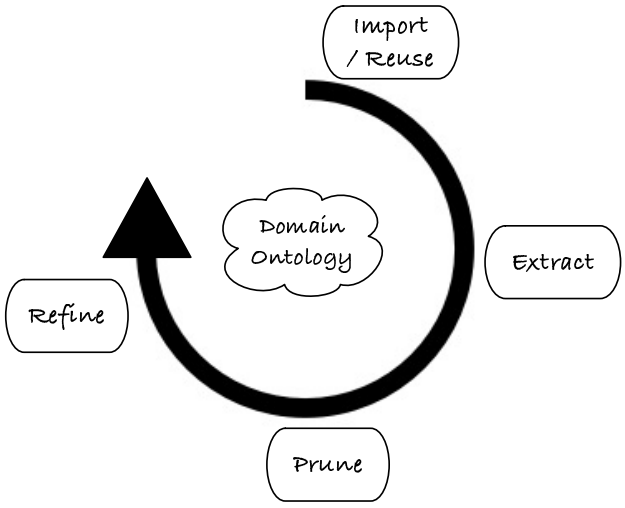

Figure 3.2: Ontology Learning Life Cycle

The Ontology Learning Layer Cake (see Figure 3.3) represents an attempt to classify the subtasks of Ontology Learning as a layered cake, where the complexity increases from bottom to top [Buitelaar *et al.*, 2005].

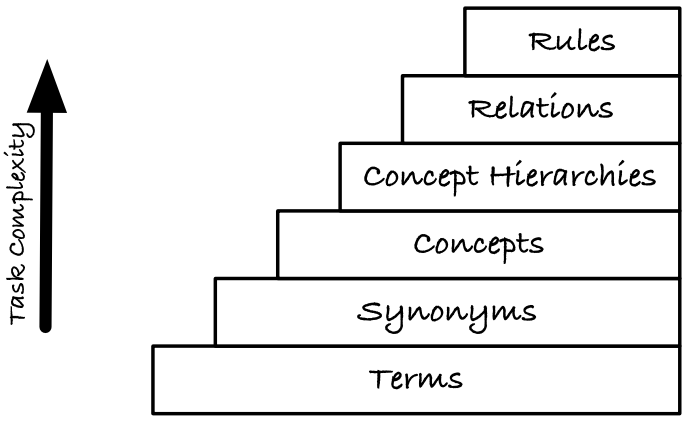

Figure 3.3: Ontology Learning Layer Cake

In their work, Buitelaar and colleagues [2005] give an overview of the state-of-the-art of the related work, which they categorise according to the layers of their proposed layered cake. Such a categorisation is necessary whenever a somehow organised comparison of different approaches is aimed. This is mainly because learning approaches can differ along too many dimensions, making a direct comparison of two randomly chosen approaches almost impossible. For instance, according to Shamsfard and Barforoush [2003], there are six main dimensions along which Ontology Learning approaches can differ from one another:

- Elements learned: Often approaches are tailored to learn only some kind of ontological components, that is concepts and concepts instances, relations (taxonomic or non-taxonomic), or axioms.
- Starting point: Approaches can differ in terms of their required knowledge for starting the actual learning process: prior knowledge (e.g., base ontology, base lexicon, etc.), input data (e.g., structured or unstructured text, etc.). Learning from natural language text will require different methods than learning from dictionaries or data schemas, which are much more structured.
- Preprocessing: Most approaches perform some kind of linguistic processing (deep vs. shallow semantic processing) on their input data before starting with their actual learning process.
- Learning method: Different approaches use different learning methods, which can in turn also be classified along the following four dimensions:
  - Learning category: supervised vs. unsupervised, online vs. offline.
  - Learning approach: statistical vs. logical, linguistic based, pattern matching, or hybrid methods.

  - Learning task: classification, clustering, rule learning, concept formation, ontology population.
  - Degree of automation: manual, semi-automatic (differing again in terms of type and level of user intervention), automatic.

- Result: Different approaches return different outcomes, for example they can return a particular ontology (type, representation language, structure) or represent their learned ontological components in an intermediate structure.

- Evaluation method: Some researchers evaluate the applied learning methods and some evaluate the final outcomes of their approaches.

We will organise our selection of related work according to the elements learned by the proposed semi-automatic or automatic generation methods, because obviously different methods have to be applied to learn different ontological components.

Learning Concepts and Instances

Learning concepts and instances is the most addressed task among existing learning approaches and are also the easiest ontological components to learn. It is mainly the task of finding and classifying words that are related to the particular domain of interest at hand.

Most approaches for concept learning are based on statistical methods and means, such as frequency measures. Frequency measures are used to identify relevant terms based on the assumption that when a term appears often in a text it is most likely relevant.

Given a corpus D with documents $\{d_1, ..., d_{|D|}\}$ and a term i, the most common frequency measures are:

- Term frequency $tf_{i,j}$: The number of times the term i occurs in document d_j.

- Document frequency df_i: The number of documents in a corpus that the term i occurs in.

- Inverse document frequency idf_i: A measure indicating the general importance of the term i relatively to the corpus.

$$idf_i = log\frac{|D|}{df_i}$$

- Term frequency-inverse document frequency $tfidf$: A balanced measure of the importance of the term in a corpus.

$$tfidf = tf_i \cdot idf_i$$

Instance learning on the other hand, can be seen as a classification problem. Most approaches perform clustering on the concepts of an ontology and then use similarity measures to identify the correct cluster for an instance at hand.

Learning Relations

Often, learning approaches only address the task of learning taxonomical relations to build a hierarchy of concepts. The *'isA'* relation is the most common relation in that context, whereas *'a isA b'* implies that *'a'* is a subconcept of *'b'* (hyponymy). Other common relations are *'partOf'* relations and several linguistic relations such as synonymy relations, etc.

Predefined lexico-syntactic patterns can be used to detect and extract particular kinds of relations from text. To get an idea of how such patterns may look like, the patterns proposed by Hearst [1998] to detect hyponymy relations from text are stated below, where NP stands for any noun phrase in the text.

such NP *as* {NP,}*{*or* | *and*)} NP

NP {, NP}*{,} *or other* NP

NP {, NP}*{,} *and other* NP

NP {,} *including* {NP ,}*{*or* | *and*} NP

NP {,} *especially* {NP ,}*{*or* | *and*} NP

It is obvious that such approaches are very limited, not only in terms of the domain language, but also in terms of the kind of relations that are covered by the set of rules. All relations that are not covered by the defined patterns (mostly non-taxonomical relations) will be completely neglected.

An approach to overcome this limitation has been proposed by Schutz and Buitelaar [2005]. The main idea behind their work is that verbs can be good indicators for related concepts. Building on top of that they first identify relevant terms and verbs from a domain-specific text collection and then automatically compute highly relevant triples consisting of a pair of concepts connected by a relation using linguistic and statistical methods.

Apart from linguistic approaches, clustering approaches can also be applied on concepts and relations in order to acquire concept and relation taxonomies. An example for that is the ASIUM system proposed by Faure and Nedellec [1999]. They build basic clusters of words using semantic similarity measures and propose the results to an ontology developer who then either accepts or discards the learned clusters.

3.3 Change Management of Ontologies

As mentioned earlier (see Section 2.1), ontologies are abstract views of specific fields of concern. These abstract views cannot be considered as static, because there are several occasions that can make it necessary to change the ontology. Such occasions may be corrections in the conceptualisation, adapting the ontology to changing facts in the real world in order to catch up with the current reality, etc.

Klein and Noy [2003] define several tasks that have to be provided for a complete change management support, as follows:

- *Data Transformation:* A change in the intensional part of an ontology (i.e., $\mathcal{C}$, $\mathcal{R}$, $\mathcal{H}^{\mathcal{C}}$, $\mathcal{H}^{\mathcal{R}}$ in Definition 2.1) may require other changes in the extensional part of the ontology (i.e., $\mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{R}}$ in Definition 2.1). For example if two concepts A and B are merged, then the instances of these classes have to be merged as well.
- *Data Access (Compatibility):* It should be possible to access data that is confirm with the old version via the changed new version. This brings the question of 'What kind of data we want to preserve?' to the forefront, because there are several dimensions to compatibility between ontology versions [Noy and Klein, 2004]:
  - Instance-data preservation: to make sure that no data is lost during the transformation from the old version to the new one.
  - Ontology preservation: to make sure that a query result obtained by using the new version is a subset of the same query result obtained by the old version.
  - Consequence preservation: to make sure that all the facts that could be inferred using the old version can still be inferred using the new version.
- *Ontology Update:* In a distributed environment, where an ontology is distributed to many users, it should be possible for users to update their local ontologies when the corresponding remote ontology changes.
- *Consistent Reasoning:* The consistency of an ontology should be maintained after changes occur. This will ensure that reasoning is still possible on the changed ontology.
- *Verification and Approval:* Interfaces to enable and simplify the validation and verification of proposed changes to an ontology by ontology developers should be provided.

Change management in ontologies can take several forms:

- **Ontology Modification:** changing the ontology without bothering about its consistency.
- **Ontology Versioning:** building and managing different versions of an ontology and providing access to these versions.
- **Ontology Evolution:** changing the ontology while keeping it consistent.

The terms 'ontology versioning' and 'ontology evolution' have been adopted by the ontology engineering community, as many researchers thought that these fields are similar to the fields of schema evolution and versioning in the database community. Although, many other researchers stated that there are fundamental differences

between those two fields and it is not possible to strictly distinguish between ontology versioning and evolution [Noy and Klein, 2004], the terms remained and are being used widely still.

In this section, we will introduce the research fields of Ontology Versioning and Ontology Evolution in more detail. We will also examine different kinds of change operations that are likely to occur in the life cycle of an ontology and ways to represent those changes.

3.3.1 Change Operations

To define change operations for ontologies is not easy, because one has to take into account all the possible effects a change can have on the components of an ontology. According to Klein [2004] one can distinguish between three kinds of changes:

- *Conceptual changes:* represent changes in the conceptualisation itself.
- *Specification changes:* represent changes with regard to the specifications of the conceptualisations.
- *Representation changes:* represent changes regarding the used ontology representation language.

3.3.2 Representing Ontology Changes

One issue in the context of change management is the proper representation of changes. The easiest and the most straight-forward way might be to keep track of changes in form of a change log that contains the exact sequence of changes applied to an ontology. Although easy, it might be a problem to make such change logs available to a distributed community of ontology developers and/or users. However, it can be useful for local ontology development, hence the change-log support provided by several ontology-editing tools, such as Protégé or OntoEdit.

According to Klein [2004] a comprehensive change specification should consist of at least the following information:

- an operational specification of a change,
- the conceptual relation between the old and new versions of the changed constructs,
- meta-data about the change,
- the evolution relation between constructs in the old and new version,
- and information about task or domain specific consequences of changes.

Besides change logs, Klein and Noy [2003] mention three more possibilities to access and represent changes between versions of ontologies: performing a 'structural diff', representing differences in form of conceptual changes, or representing differences in form of a transformation set. The first way is to perform a *structural*

diff as described in the work of Noy and Musen [2002]. They draw a comparison to the field of software code versioning, where the code also changes often and differences between two versions can be accessed using a process called *'diff'*, which returns a list of lines that are different in the two versions. The authors state, however, that this approach cannot be inherited for comparing two ontology versions, because the form of representation and the form of generating ontologies is completely different than with software code. So, it is possible that two ontologies are identical in terms of their conceptualisations, but differ immensely in terms of their internal representation. They propose an algorithm called PROMPTDIFF[1] for comparing two ontologies w.r.t. their structure and not their text representation. For each ontological structure in the old version of an ontology, it looks for possible corresponding structures in the new version. If there are some structures for which no direct counterparts can be found, it applies several heuristics in search for possible matches. It then tries to merge these results using a fixed-point algorithm. The authors claim that PROMPTDIFF can achieve an average recall value of 96% and an average precision value of 93%.

The second way is to represent differences between ontologies in form of *conceptual changes*. Such changes specify the conceptual relation between ontological structures in the old version and the new version. For example, a conceptual change may state that a concept A was a subconcept of B in the old version before being moved to its place in the new version.

The third way is to represent differences between ontologies in form of *transformation sets*, which contain change operations that specify how (i.e. applying which changes) the old version of an ontology can be transformed into the new version. Transformation sets differ from simple change logs, as they only contain necessary operations to achieve the intended (changed) version and not every single change applied to the ontology as in simple change logs. Furthermore, transformation sets may not contain changes in the same order as they were really applied. For example, adding new components can be grouped together, because they do not affect the rest of the ontology like as delete operations.

We think, that ontology changes can also be integrated into the ontology itself as instances of a general concept 'change'. These instances, then, can be used to save information about affected ontological components. One may think of many other ways to represent ontological changes. The important thing is to choose a way that serves the purpose of its application best.

3.3.3 Ontology Versioning

Klein [2002] defines *Ontology Versioning* as "the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology".

Ontology Versioning should enable the management of different versions of the same ontology at the same time. This functionality is essential in scenarios where developers or users of an ontology are going to access an ontology in a distributed manner. Considering that one of the major benefits of ontologies is the re-use and

[1] This algorithm is available as a plugin for the Protégé 2000 ontology-editing environment.

inter-operabilty they can provide, such a scenario is more than a theoretical one. Currently no sophisticated versioning mechanisms are available. Often, ontologies change and the old versions are lost forever, because only the latest version is accessible. Sometimes, old and new versions of ontologies are archieved, but no mechanisms are provided to highlight the differences between versions.

The first attempt to address this problem has been taken by Heflin and Hendler [2000] with introducing the Simple HTML Ontology Extensions (SHOE) as an extension to HTML to represent ontology-based knowledge using additional tags. One important facility of SHOE is that it enables ontology developers to state whether a version is backward-compatible with an old version or not. In a distributed application field where many interaction partners (e.g., applications, agents, etc.) use the same ontology, this information is essential, because it determines whether they can continue with their work as usual or they have to update their versions in order to maintain agreed-upon interaction. The work of Heflin and Hendler [2000] is also important in terms of its long-standing contribution, by starting the discussion about the problem of Ontology Versioning in dynamic, distributed, and heterogeneous environments.

However, the current trend of the Semantic Web makes more sophisticated approaches to Ontology Versioning necessary. Klein [2002] impose the following requirements on an Ontology Versioning framework:

- *Identification:* The intended definitions of ontological components have to be made clear in advance, because they represent prerequisites for change specifications.
- *Change specification:* Possible changes have to be specified according to the identification of ontological components. Since different representation paradigms provide different components, the change specifications will also differ.
- *Transparent evolution:* It should be clear what the actions are that have to be taken when particular changes occur. For that purpose, change specifications will be used to translate and relate different versions of ontological components.
- *Task awareness:* Because there are different dimensions to compatibility (e.g., preservation of instance data, preservation of query results, preservation of consequence, etc.), a framework have to be aware of the concrete task in order to provide appropriate transformations between versions.
- *Support for untraced changes:* It is often the case, that there is no track of changes that represent the steps from one version to the new one. In such cases, a versioning framework should provide mechanisms to determine whether two versions are compatible or not.

The main objectives of a versioning framework that reconcile the above mentioned requirements can also be found in Klein's work [2002]. The first objective is surely to provide *data accessibility* through different versions of an ontology. This

can be achieved, either by restricting allowed changes to only those that do not affect the interpretation of data, or by providing translations between the versions so that user queries can be translated back in order to access the data in the old version.

Consistent reasoning is another objective, as it aims to ensure that reasoning over the ontologies is not affected. In that way, it can be guaranteed that the answers to at least a specific set of queries will remain the same with different versions.

In a distributed environment, it is also important to provide *synchronisation* and *data translation* support. Whereas, the former enables the update of local ontologies with a remotely changed ontology, the latter enables the automatic translation of affected data sources to be conform with a newer version of an ontology.

Versioning is also important w.r.t. collaborative ontology development, where more than one developer wants to make changes on an ontology (*management of development*). For such a scenario, step-by-step verification and authorisation have to be supported.

3.3.4 Ontology Evolution

Stojanovic and colleagues [2002] define Ontology Evolution as follows:

> Ontology Evolution is the timely adaptation of an ontology to changed business requirements, to trends in ontology instances and patterns of usage of the ontology-based application, as well as the consistent management/propagation of these changes to dependent elements.

They further state a set of design requirements for proper Ontology Evolution [Stojanovic *et al.*, 2002]:

- It has to enable resolving the given ontology changes and to ensure the consistency of the underlying ontology and all dependent artefacts;
- It should be supervised allowing the user to manage changes more easily;
- It should offer the user advice for continual ontology refinement.

According to them the ontology evolution process can be considered in six phases (see Figure 1) [Stojanovic *et al.*, 2002]:

- *Change capturing :* This phase encapsulates the process of deciding to apply a change on an ontology. This might be forced by explicit requirements (the ontology engineer decides to make a change) or by results of automatic change discovery methods. The first kind of changes are called top-down changes, whereas the second one are called bottom-up changes. Bottom-up changes can be proposed by three different approaches to change discovery: structure-driven, data-driven, and usage-driven change discovery [Stojanovic and Motik, 2002].
- *Change representation:* In order to resolve changes, they should be identified and represented clearly and in a suitable format (see Section 3.3.2). They can be represented in form of elementary or complex changes.

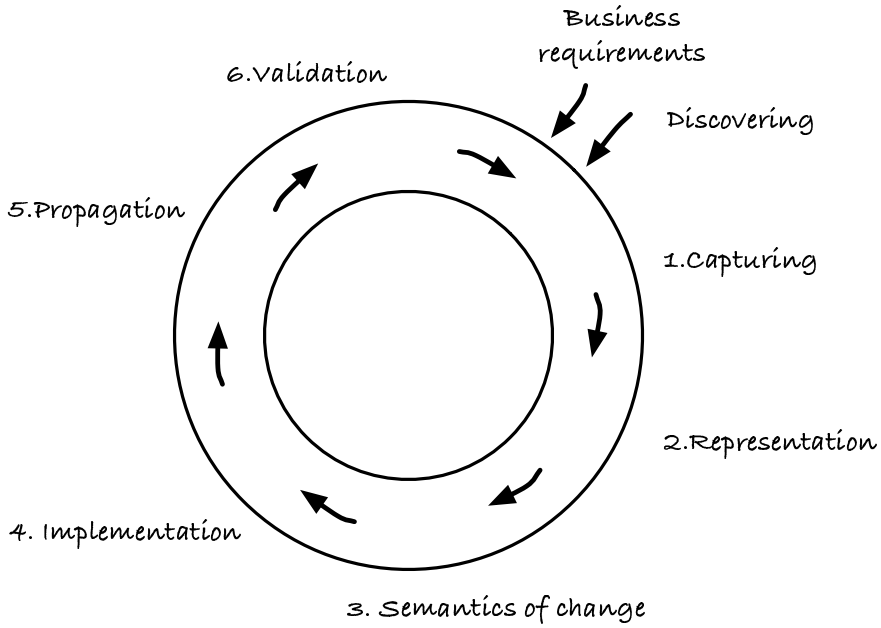

Figure 3.4: Phases of Ontology Evolution

- *Semantic of changes:* How a change can affect the ontology's consistency must be understood in advance, where the meaning of consistency depends on the underlying ontology model.
- *Change propagation:* To preserve consistency, affected artefacts should be handled appropriately as well. In a distributed environment, affected artefacts are not bound to local components of the changed ontology, but contain also distributed ontologies that reuse or extend the changed ontology, or even applications that are based on the changed ontology.
- *Change implementation:* Before applying a change, all implications of it have to be presented to the user, who then can accept or discard it. If the user agrees with these implications, all activities to apply the change have to be performed.
- *Change validation:* It should be possible for a user to validate performed changes and to reverse the effects of them when necessary.

It is essential to discover the types of changes that can occur, because they have to be handled differently. We distinguish between basic (or elementary) changes, such as deleting or adding a concept, and complex (or composite) changes that are composed of multiple basic change operations.

More important is the distinction between changes that can lead the ontology into an inconsistent state and changes that cannot. Whereas the latter class of changes do not cause any problems, the former class of changes requires special treatment. Such a treatment can be in form of an *Evolution Strategy*, set by the user

in advance to define how to resolve critical changes unambiguously. Stojanovic and Motik [2002] stated the following situations in which an Evolution Strategy could help to determine the further course of action:

- how to handle orphaned concepts
- how to handle orphaned properties
- how to propagate properties to the concept whose parent changes
- what constitutes a valid domain of a property
- what constitutes a valid range of a property
- whether a domain of a property can contain a concept that is at the same time a subconcept of some other domain concept
- the allowed shape of the concept hierarchy
- the allowed shape of the property hierarchy
- whether instances must be consistent with the ontology

Stojanovic and colleagues [2002] introduced also the term of *Advanced Evolution Strategy*. It represents a mechanism that automatically combines available elementary evolution strategies and relieves users from choosing them individually. They defined the following set of advanced evolution strategies:

- *Structure-driven strategy:* resolves changes according to criteria based on the structure of the resulting ontology.
- *Process-driven strategy:* resolves changes according to the process of changes itself.
- *Instance-driven strategy:* resolves changes to achieve an explicitly given state of instances.
- *Frequency-driven strategy:* applies the most used or last recently used strategy.

3.4 Evaluation of Ontologies

The AI community is used to have some standard measures to value the results of new approaches. In the Information Extraction (IE) and Information Retrieval (IR) community, for example, measures like Precision and Recall have emerged after conferences like the Message Understanding Conferences (MUCs) or the Text Retrieval Conferences (TRECs). Unfortunately, there was no such development in the field of Ontology Evaluation. This might be the main obstacle to the widely usage of ontologies in different fields.

As we stated earlier, ontologies can play a major role in knowledge sharing and reuse. If you cannot value the usefulness of an ontology for your purposes by using some standard measure, you will not going to risk the performance of your application by sharing or reusing the knowledge it offers. There are several questions to be answered to get a clear view of the evaluation process of ontologies.

Which measures to use?

We have to state, right at the beginning, that we cannot compare the evaluation of ontologies with the evaluation of tasks in IE and IR, because the measures of Precision and Recall cannot be directly applied. You may try to define *Precision* as the amount of knowledge correctly identified with respect to all the knowledge in the ontology; and *Recall* as the amount of knowledge correctly identified with respect to all the knowledge it should identify. Brewster and colleagues [2004] argued, that this is not so easy, because you have to define first what the 'knowledge to be acquired' actually is, as the same set of facts can be interpreted in several ways, hence can lead to different kinds of 'knowledge'.

Whereas there are no standard measures for evaluating an ontology, Gómez-Pérez [1995] stated the following three criteria:

- **Consistency:** To which extent the ontology is incapable of getting inconsistent results from valid input data.
- **Completeness:** To which extent the ontology covers the information of the real world.
- **Conciseness:** To what extent the information in the ontology is useful and precise.

The first and third criteria are important without a doubt. But the second one, about completeness, left room for speculations. This criterion is, on the one hand hard to determine, because an ontology is a subjective view of the world, and what is complete for one observer might not be complete for an other. At the other hand, it is questionable whether a real complete ontology is needed for ones specific purpose. It also contradicts with the 'minimal ontological commitment' criterion of Gruber's [1993] Ontology Generation criteria, which states that the ontology should contain only the essential terms in order to let space for further specialisation by the interaction partners.

What to evaluate?

Another major problem is, that it is not clear whether the ontology as an end product should be evaluated or intermediate products too, such as the conceptualisation as an abstract model of the real world, etc.

- Real world - Conceptualisation: To what extent fits the conceptualisation the real world or a field of concern?

- Conceptualisation - Ontology: To what extent is the chosen representation language able to express the intended meaning of the conceptualisation?
- Ontology - Application: To what extent is the ontology correctly used in a specific application?

Further it is important on which components of an ontology the evaluation should be carried out. Gómez-Pérez [1995] listed the following components that could be the subject for evaluation:

- Every individual definition and axiom
- Explicitly stated collections of definitions and axioms
- Definitions imported from other ontologies
- Definitions that can be inferred from other definitions and axioms

The documentation is another component of an ontology that probably should be evaluated. Since ontologies tend to be huge in terms of concepts and relations and are not always about well-known fields of concern, a documentation would help immensely to better understand the ontology. In this point of view, we can say that the visualisation of an ontology is also an important component that should be evaluated. There are many ways to represent an ontology visually. Unfortunately, not all of them give the user the needed insights into the ontology as they are supposed to. The evaluation of visualisations is, however, worth a research task on its own and thus will not be further explored here.

When to evaluate?

The right time for evaluation is another important issue. Should the whole ontology be evaluated at the end of the building process? Or would it be better to evaluate it after each insertion of a definition in order to change it immediately in case of a bad evaluation result? Is it more straightforward to evaluate after each phase of the building process?

We can see, how the chosen methodology for Ontology Generation can affect forthcoming actions, like here the evaluation, as the phases of the generation process would differ from one methodology to another, leading to different demands regarding evaluation.

How to evaluate?

As in any other field in AI, there is a need to automate processes. However, this is a delicate issue, especially for the evaluation task as the results are being seen as quality measures for the approaches and thus have to be correct and precise. The case of Ontology Evaluation is even harder, because both the manual and the automatic methods can guarantee neither correctness nor preciseness. During the manual evaluation the question arise by whom the evaluation should be carried on. As we stated earlier, an ontology is a subjective artefact. This led us to a position

in which someone could recommend the ontology as a 'good' one, whereas another one could say that is completely 'useless'. This measurement will also depend on the purpose of the ontology. The case of using it as a knowledge sharing artefact will differ from using it as part of an application. So, it is possible that an ontology is 'useful' for one purpose and useless for another. Whether the ontology should be evaluated by the authors, by 'objective' outsiders, or by the end-users is yet to answer.

Automatic evaluation of an ontology is a recent research issue. It is assumed that ontology learning methods can be used for this purpose. One approach is the *data-driven ontology evaluation* approach by Brewster and colleagues [2004]. It aims to automate the process by comparing ontologies with a corpus. Assume that you have an ontology about a specific field of interest at hand. Since you know what the ontology is about you can build a corpus of related documents. Having this scenario, one approach could be to apply automated term extraction on the corpus and to simply count the number of terms that overlap between the ontology and the corpus.

3.5 Ontology Visualisation

To graphically visualise an ontology is a challenging task and relatively little research has been done in this field. However, appropriate visualisation means are necessary to enhance the understanding of ontologies by users, and to support users during their ontology engineering tasks.

So far we examined several stages of the ontology life cycle and have seen that they differ in terms of their respective tasks and challenges. According to Fluit and colleagues [2004] they also differ in terms of the capabilities they require from visualisation methods to assist users performing these tasks. For *ontology generation* detailed a visualisation of concepts and their relationships is needed to enable the full understanding of the generated ontology by the user. Typically a small number of concepts and relationships will have to be visualised at the beginning of this stage, but as the ontology grows, more sophisticated visualisation methods will be needed that enable the zooming into particular parts of the ontology and that enable to visualise only certain aspects of the ontology. The task of *ontology instantiation* requires visualisation methods that differentiate between the intensional and the extensional part of an ontology and *ontology deployment* requires methods that enable querying and navigation of ontological information spaces.

In this section we will state two well-known visualisation methods that have been used to visualise ontologies, Graphs and Tree-Maps. However, we can not say that these methods are tailored especially for the use with ontologies, in fact as far as we know there is no such method.

3.5.1 Graphs

Graphs are used in many research fields to visualise information that is structured somehow in classes and relations. Because, ontologies also consist of classes and

relations between them, graphs are the first visualisation methods that came into mind to visualise ontologies.

Mutton and Golbeck [2003] visualise ontologies by generating graphs of ontologies and instance data. Their focus is on graph drawings where related classes (i.e., classes that are connected through properties) are placed near to each other whereas other nodes are evenly distributed aiming to give the user insight into the structure of the ontology.

In particular, they use the *spring embedding* graph drawing method that enables automatic generation of drawings with these properties. The spring embedding method distributes nodes in a two-dimensional plane, whereas nodes related to each other are placed closed to each other (see Figure 3.5). The method is based on a force model that actually computes the values for the placement of nodes. Mutton and Golbeck [2003] use the force model of Fruchterman and Reingold [1991] because it is widely used, effective, and relatively easy to implement. With their implementation, Mutton and Golbeck [2003] reached good results on graphs with up to several hundred nodes.

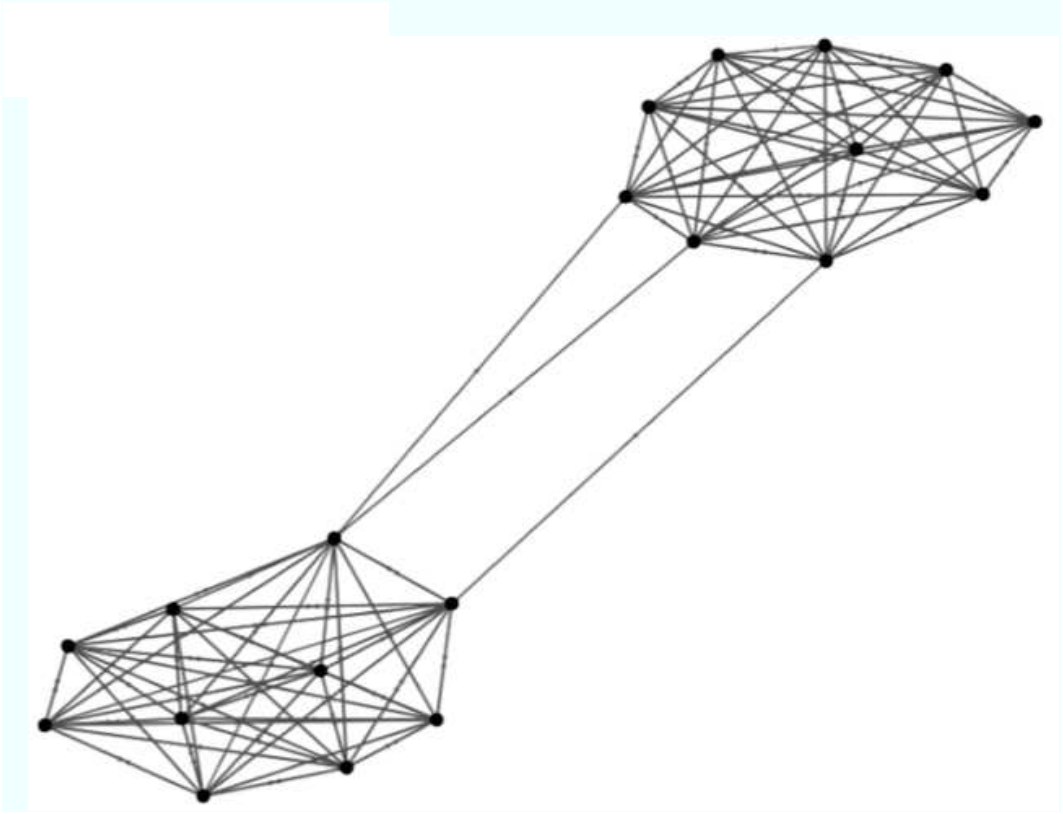

Figure 3.5: A Spring Embedding of a Graph

3.5.2 Tree-Maps

Opposed to the traditional represenatation of tree structures as rooted, directed graphs Shneiderman [1992] introduced a ”two-dimensional space filling approach

in which each node is a rectangle whose area is proportional to some attribute such as node size". In Figure 3.6 we can see a traditional representation of a tree structure, whereas Figure 3.7 shows the corresponding tree-map representation (both pictures from [Shneiderman, 1992].

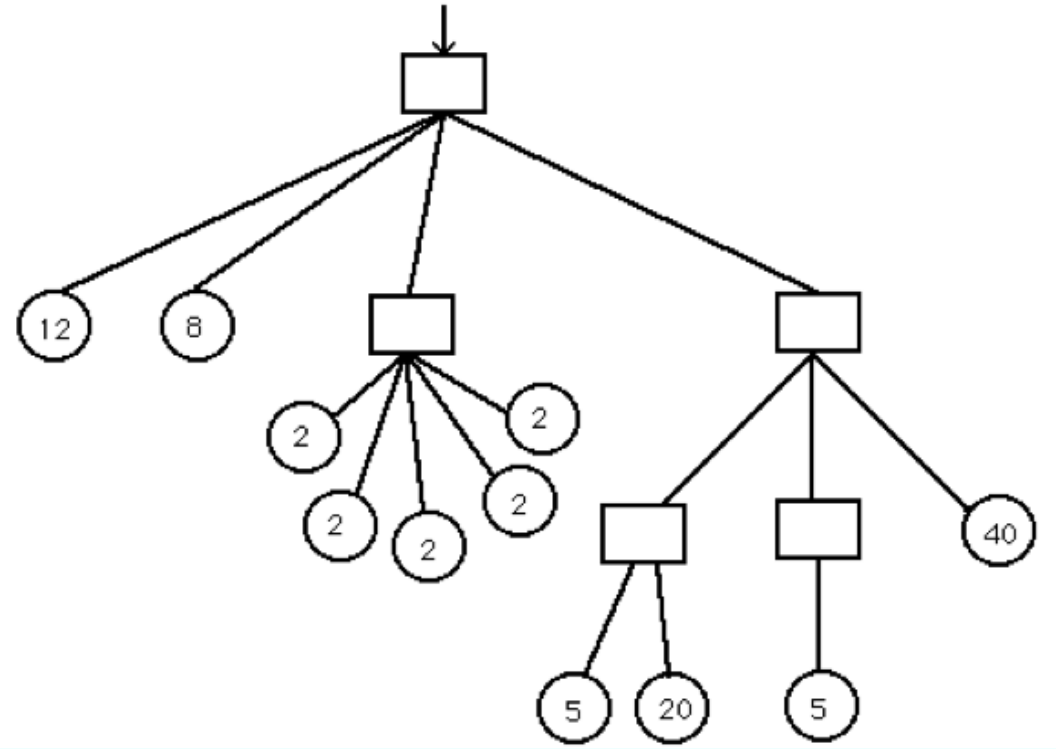

Figure 3.6: Typical 3-level tree structure with numbers indicating the size of each leaf node

Tree-maps aim to visualise complex traditional tree structures and as such are applicable on ontologies as well. Baehrecke and colleagues [2004] use them to visualise highly complex genome data available in the Gene Ontology[2] and to facilitate queries by presenting attributes of genes by size (RNA level) and color-coding (p-value).

3.6 Conclusion

A lot of work is still to be done in the field of Ontology Engineering, namely in each one of the subfield that is part of the ontology life cycle.

In the sub-field of Ontology Generation, standard methodologies have to be defined. We have seen that it is not easy to build a large ontology and that it requires, in general, highly skilled specialists if the ontology have to be built by hand. Since they are not always affordable, methodologies would help to standardise the generation process.

[2]http://www.geneontology.org/

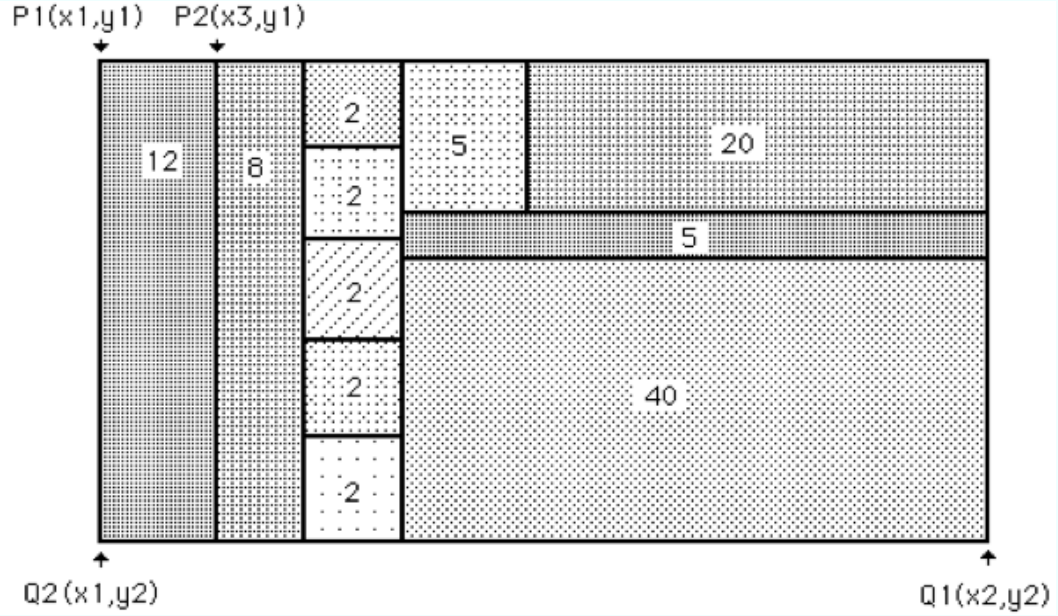

Figure 3.7: Tree map of Figure 3.6

Another issue is co-operative and multi-user Ontology Generation. Usually there are more than one human beings involved in the generation process. The problems that are present and well known in each multi-authoring environment apply also here. The different communities affected by these problems should co-operate in order to find feasible solutions.

Change management with respect to ontologies is another issue. It is clear that an ontology may not remain the same forever, because the underlying conceptualisation may change over time. In order to have an updated ontology, first the different change types and ways to handle these have to be defined. At the moment, little research is done about how to keep track of changes in ontologies.

The evaluation of ontologies is another very important sub-field. The community still lacks of a common way to evaluate ontologies. There are too many questions regarding the evaluation of ontologies starting with the choice of the measure to use, over to decide when to evaluate, to the question of what exactly to evaluate and how.

Chapter 4

Information Extraction

So on and on I go, the seconds tick the time out
There's so much left to know, and I'm on the road to findout

On The Road To Find Out - Cat Stevens

Information Extraction (IE) is defined as a form of natural language processing in which certain types of information must be recognised and extracted from text [Riloff, 1999]. For the sake of completeness, we have to state that the data in which the information is sought, do not have to be text. The data can also consist of images, sound, video, etc. But most of the research is done in the field of Text Extraction, so we will use the term IE only in this sense later on.

IE attained much interest within the last three decades fostered by the Message Understanding Conferences (MUCs) started in 1987. Since then, many Information Extraction Systems (IESs) have been developed by different sites, both from the research and the business area. This is not surprising, since both of these fields have to handle more data than ever in a fast and effective manner. What could be then more straight-forward than seek the aid of the computers to automate the process of finding relevant information from available data?

IE is a part of Information Processing (IP), which comprises the whole field related to gathering, manipulating, storing, retrieving, and classifying information.[1] Many of the sub-tasks of IP seem to be similar, due to the used methods and approaches they share, but they are different in their intended functionalities. For example, IE is often mentioned along with Information Retrieval (IR), Text Understanding (TU) or even Text Summarisation (TS). Even a MUC has the term 'Message Understanding' in its name, although there was no real message understanding carried out at the conferences. The usage of these terms should not confuse or irritate the reader.

IR describes the task of finding relevant documents within a corpus of documents and not the relevant information within a document like IE does. TU aims to understand the meaning of the text in a document and is therefore not at all directly comparable with IE. TS tries to understand the meaning and to generate a short summarisation of the document or set of documents. TU and TS differ not only in

[1] http://www.thefreedictionary.com

terms of the intended functionalities from IE, but also in terms of the kind of evaluation. These two fields cannot be evaluated using the well established measures of IE, because one have to read the returned results in order to decide whether the system had worked properly or not. Or they can be evaluated by testing whether they can give answers to questions about this document. Further, these two fields consider all the text in document as relevant, whereas in IE most of the document is considered irrelevant.

The question of what 'relevant information' actually is comes to ones mind immediately. Unfortunately, the answer cannot be given so easily because it depends highly on the current task specification. In a task specification where we are interested in person names the relevant information is clear for any human being and she would be able to identify this kind of information in a document with ease. But if your task is to extract relevant information for patient treatment for a specific decease and your input documents are medical guidelines, then it would be harder to decide which sentence in a guideline is relevant for the treatment, especially if one is not familiar with this domain. Additionally there is the problem that what is relevant for someone, is irrelevant for someone else. Thus, the clear and unambiguous definition of 'relevant' information is very important in IE.

In this chapter we will explore the aims, techniques and challenges of this interesting field of research starting with a brief history.

4.1 A Brief Historical Overview

When talking about the history of IE one can nothing but mention the Message Understanding Conferences (MUCs), a series of conferences aimed to evaluate and compare the works and results of different research groups and to foster the research in this field. We can say that these aims were reached, because a significant improvement of the developed IESs can be observed over the conferences.

The procedure at the MUCs was as follows: each participant worked on a given scenario with a set of annotated documents, the training or tuning corpus, and a set of templates which described the kind of information the developed IES has to find. After a certain amount of time (one to six months), the participants were given a new set of documents (i.e. test corpus) with which they had to test their IESs without making any changes to the systems. The extracted templates for these test set were then send to the conference organiser who compared them with his own, manually build answer keys which led to the evaluation results of each IES. At the conference itself the works and the results were presented to give the other participants an insight in the works of the others.

Over the years the following extraction tasks were introduced to the conferences [Marsh and Perzanowski, 1998]:

- **Named-Entity Recognition Task (NE):** This task corresponds to the lowest level of IE tasks and it is domain independent. It involves the identification and categorisation of proper names (organisations, persons, and locations), temporal data (dates and times) or numbers (currency, percentage).

- **Multi-lingual Entity Task (MET):** The task is the same as in NE, but for Chinese and Japanese.
- **Template Element Task(TE):** In this task an output template is given, which has slots for basic information related to organisations, persons, and artefact entities. An IES has to draw evidence from anywhere in the text to fill these slots.
- **Template Relation Task (TR):** This task is about extracting relational information among entities. Examples for such relations are, employee_of, manufacturer_of or location_of relations.
- **Scenario Template Task (ST):** This task represents the top-level of IE tasks. In this task the focus is on the extraction of pre-specified events, whereas the system has to relate the event information to particular organisation, person, or artefact entities involved in the event.
- **Co-reference Task (CO):** This task is about capturing information on co-referring expressions (i.e. all mentions of a given entity).

The used scenarios and the complexity of the tasks at the conferences changed over time. In the following there is a short overview of the several conferences based on the survey of Grishman and Sundheim [1996].

MUC-1 (1987)

The set of documents were naval operation reports. Neither a specific task description nor a specific output format for the extracted information was defined. The conference served just as a platform for comparison of the different IESs without any defined evaluation criteria.

MUC-2 (1989)

The documents were again naval operation reports. The task was defined as to extract events by filling a template with ten slots for information about the event, such as the type, the agent, the time and place, and the effect of the event.

These two conferences had been initiated and carried out by Beth Sundheim under the auspices of the U.S. Navy. After the second conference, the conferences had been carried out under the auspices of the TIPSTER Text Program.

MUC-3 (1991)

The documents were articles about terrorist activities in Latin America. The defined template became more complex and had 18 slots. Formal evaluation criteria were introduced. A semi-automatic scoring program was available for the participants during the development, but the official scoring was done by the organisers.

MUC-4 (1992)

Only the template complexity increased to 24 slots. The domain and the task description stayed the same as in MUC-3.

MUC-5 (1993)

A jump regarding the task complexity can be noticed at this MUC, as the task documentation over 40 pages indicated. This time documents from two different domains were used: financial newswire stories about international joint ventures and product announcements of electronic circuit fabrication. The documents were in English and Japanese. Nested templates were used for the first time and new evaluation metrics were included in the scoring system.

In all the prior MUCs a clear trend was observed, namely that it took too much time (in general 6 months or even more) to adopt a system for a new scenario. As a reaction to the trends in the prior MUCs, a meeting was held in December 1993 during which a set of objectives for the forthcoming MUCs were defined. Among the identified goals were:

- Demonstrate task-independent component technologies of IE which would be immediately useful.
- Encourage work to make IESs more portable.
- Encourage work on 'deeper understanding' of the texts.

MUC-6 (1995)

The documents were articles about management changes. Four tasks were included in the specification: NE, CO, TE, and ST.

MUC-7 (1998)

Additionally to the task at MUC-6 the TR task was added. The documents were news articles about space vehicle and missile launches.

After this brief overview of the MUCs, we should look how well the participating sites performed for the several tasks. Table 4.1 contains the evaluation results for the different tasks through the conferences [Chincor, 1998].

A clear performance decrease in the evaluation results can be observed as the tasks became more complex, whereas an increase for almost every task can be observed from conference to conference. These values should serve as a basis of comparison for other IES developers to value the performance of their IESs.

	NE	CO	TE	TR	ST	Multilingual
MUC-3					R <50% P <70%	
MUC-4					F <56%	
MUC-5					EJV: F <53% EME: F <50%	JJV: F <64% JME: F <57%
MUC-6	F <97%	R <63% P <72%	F <80%		F <57%	
MUC-7	F <94%	F <62 %	F <87%	F <76%	F <51%	

Table 4.1: Maximum Results Reported in MUC-3 through MUC-7 by Task

EJV = English Joint Venture, JJV = Japanese Joint Venture, EME = English Microelectronics, JME = Japanese Microelectronics, R = Recall, P = Precision, F = Fallout

4.2 Architecture of an Information Extraction System

Whereas IESs developed by different persons or groups differ in terms of the intended application field and used approaches, they are similar in terms of the underlying architecture.

IESs work in a sequential mode by splitting the whole process into their subtasks, whereas there is a module for each task. These modules process their respective incoming data and hand over their results to the next module in the architecture. Depending on the specific task, the needed modules can differ from IES to IES. In Figure 4.1 [Appelt and Israel, 1999] we can see the main components of an IES, where the components on the left side are the ones which are present in almost each IES, and the components on the right side are the ones which can be added depending on the current task specification.

Tokenisation

In this module the text is tokenised into its structural components such as sentences and words. This might be relatively straight-forward for languages like English where the words are separated by white spaces. But for many languages like Japanese or Chinese where the words are not separated by white spaces, an additional segmentation module is needed.

Morphological and Lexical Processing

This module is highly language dependent and the internal structure may differ from system to system according to the language of the input data it processes. Whereas morphological analysis might not be necessary for many languages with simple inflectional morphology (e.g., English), other languages with more complex morphologic characteristics (e.g., German) will require special handling.

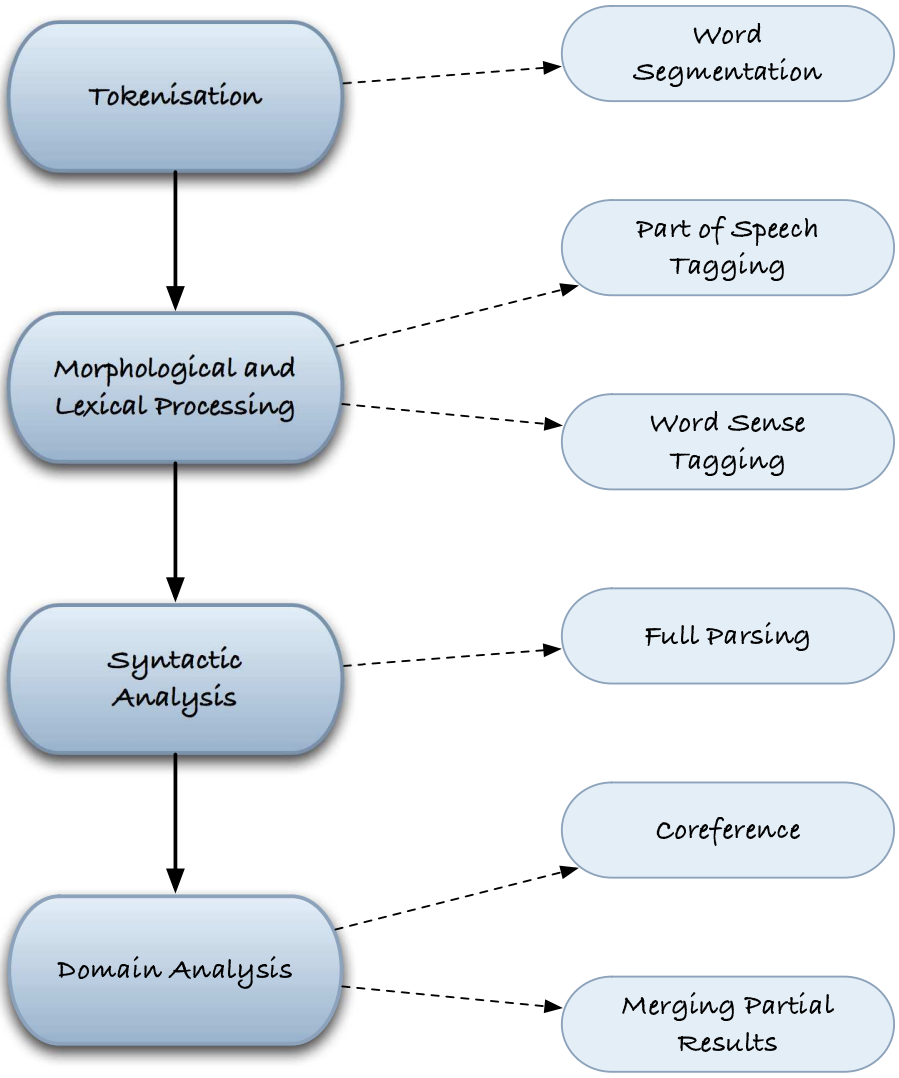

Figure 4.1: Architecture of an IES

In this module sentences are analysed in order to identify the part-of-speeches (POS) and the inflection (e.g. singular, plural, gender) of the words. The part-of-speech tagger identifies noun groups, verb groups, etc. in the sentences. This can help to disambiguate ambiguous words or to extract information where the position of a word in a sentence matters. It takes some time to train a part-of-speech tagger for a particular language and the tagging process itself takes also some time. So it is recommended to analyse the expectations from a POS-tagger and to examine if there are other, preferably cheaper ways to accomplish these.

The lexical processing is taking place in form of lexical lookup where a lexicon is being conducted that contain (domain-specific) words of the language the IES has to process. Although, one may wish to cover the language as much as possible, this very fact may lead to unintended consequences because of the increasing ambiguity in the lexicon. The preferred way is therefore to use a lexicon that contains only

domain-specific and task related words.

Syntactic Analysis

In this module the sentences are going to be syntactically analysed. A differentiation can be made between IESs using full parsing and IESs using shallow parsing. **Full parsing** means, that each sentence in the text is going to be analysed and its parse tree is going to be generated. **Shallow parsing** means that only a subset of the sentences in the text are going to be analysed. The decision of which of these should be chosen depends on the requirements of the current specification. Full parsing will give certainly a deeper grammatical insight in the text, but it is not quite clear whether the amount of time it will take is justifiable in terms of the performance increase of the system. In practice shallow parsing seems to be adequate, hence most of the IESs use this method.

Domain Analysis

This is the module in which the domain-specific processing is being done. Here, the system extracts the information which is defined as relevant for the current task specification. Some task specifications may require the information in a more complex format. For example, they may require to find coreferring expressions of entities in the text or some relational information between entities. In these cases a coreference module or a module for merging the partial results would be necessary.

4.3 Approaches to Information Extraction

For building an IES one has to choose between two approaches. The first one is the *knowledge engineering approach*, where the rules with which the IES extracts the information are build by a knowledge engineer by hand. The second is the *(semi-) automatic training approach*, where the system has to learn extraction rules by itself from annotated documents. In the following a more detailed explanation of these approaches can be found.

Knowledge Engineering Approach

As the name implies, a knowledge engineer is the backbone of this approach (see Figure 4.2). She has to be familiar with the rule-making process, the specific domain and task, and the IES itself in order to be able to generate the rules with which the IES then will extract information from documents. It is clear that it takes a plenty of time to generate these rules, because it is an iterative process. The knowledge engineer generates first a set of rules, applies these rules on a document set (tuning set), and if necessary changes the rules again to get a better coverage of the domain. Generally, many iterations are needed to get a satisfying set of rules with the correct level of generalisation. The knowledge engineer has to look whether the generated rules over- or under-fit the specific task after each iteration step. However, at the end you can be sure that the IES covers your interests at a satisfying level.

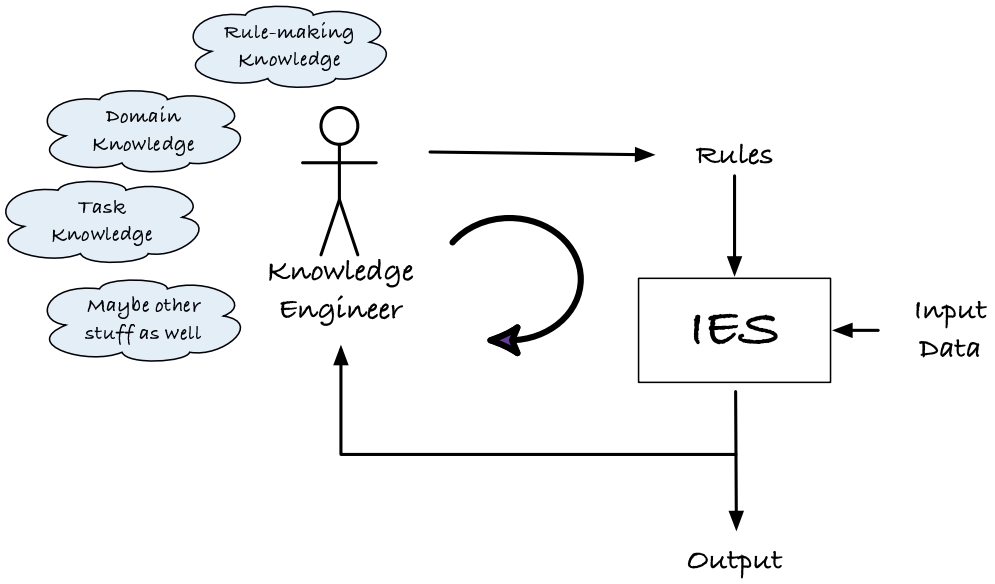

Figure 4.2: Knowledge engineering approach to Information Extraction

(Semi-) Automatic Training Approach

The focus within this approach is on automating the rule generation process fully or partially, in order to decrease the development time and the dependency upon a knowledge engineer who might not always be at hand. However, such systems require a large document set of documents from a particular domain of interest, whereas relevant information in the documents has to be annotated by a domain expert. Using these annotations the system can then derive extraction rules on its own. In general, someone who is familiar with the domain and the task will be sufficient to make these annotations (see Figure 4.3).

Grishman and Yangarber [2000] differentiate between the following four levels of human intervention within (semi-) automatic training approaches:

- Learning from fully annotated data: The human intervention involves the annotation of all the relevant information in a test document set.
- Active learning: Based on a small set of basic rules, the system selects suitable candidates for annotation and proposes them to the user who has to finally decide whether to accept or discard them.
- Learning from relevance-marked data: The human intervention involves only the annotation of relevant parts in text, for example in terms of marking whole paragraphs as relevant.
- Learning from unannotated data: Based on a very small basic rule set, the system uses a bootstrapping method to learn patterns on its own.

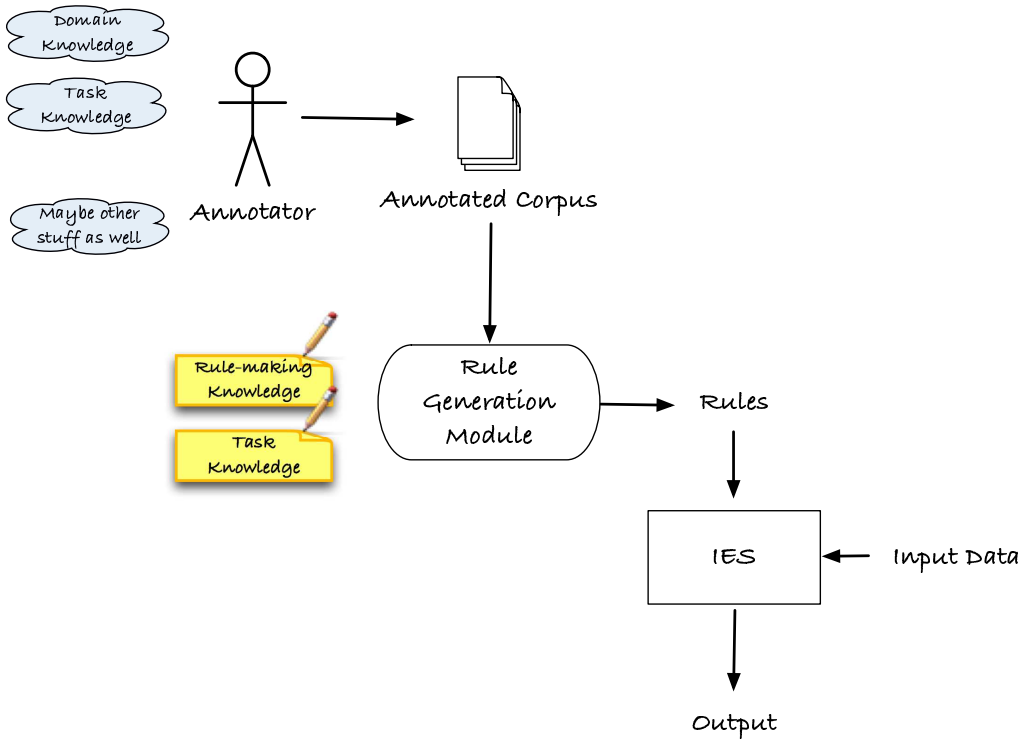

Figure 4.3: Semi-automatic approach to Information Extraction

Which approach to choose?

The *knowledge engineering approach* yield a set of rules that is likely to cover the domain and task very well. This is actually the main advantage of this approach and makes it the approach of choice in applications where highest possible precision is crucial. But a knowledge engineer might not always be at hand to generate the rules and sometimes they are not affordable. Further, it is hard to adopt the rule set by hand when a new domain or task has to be supported by the IES. Such a change in the task specification can require a completely new rule set, what makes this approach not that scalable and portable.

With the *(semi-) automatic training approach* plenty of time can be saved, because the rule generation process is automated. Furthermore, there is no dependency to a knowledge engineer. The one thing the system requires for sure is enough training data and often a domain expert to do the annotations.

Unfortunately, it is possible that the system generates a rule set that is tailored exactly for the given annotated document set. In such a case the IES will work fine for the documents it was trained on, but will simply fail when processing unseen documents. This is a typical example for over-fitting of the rules to the training documents. It is also possible that, as usual with corpus based approaches, the annotated corpus does not contain all the needed relevant information of the domain. Therefore, the corpus of documents should be selected very carefully to increase the likelihood that the rules will fit a large portion of the domain. But to ensure this, either again human intervention or good document retrieval systems are needed.

Appelt and Israel [1999] suggest to use the knowledge-based approach, when resources like lexicons and rule writers are available, training data is scarce or expensive to obtain, extraction specifications are likely to change, and highest possible performance is critical. However, to use the automatic training approach when resources and rule writers are not available, training data is cheap and plentiful, extraction specifications are stable, good performance is adequate for the task. They also suggest that different modules of an IES can be developed according to different paradigms. For example, one can develop a rule-based name recognizer that learns domain rules, or a statistical name recognizer that operates on hand-generated domain rules when data is scarce.

As you can see, both approaches have their advantages and disadvantages. Therefore, as a developer of an IES, you have to analyse a priori the setting in which your system is going to be used and to clarify your expectations from the system. The two main properties of an IES, which are going to be affected by this fundamental decision are:

- *Portability:* describes the ease with which a system can be adopted to a new domain or task.

  The portation of an IES to a new domain is considered to be hard for both approaches, because, in both cases, the rule set has to be generated from scratch.

  The portation of an IES to a new task, on the other hand, yields to different kinds of inconveniences with the two approaches. For the knowledge engineering approach this means that the rule set has to be adopted by adding or removing some rules, which are capable of performing the new task, while preserving the rules about the domain. For the (semi-) automatic training approach this means that the annotator has to go through all the document set in order to annotate parts that represent relevant information according to the new task at hand.

- *Scalability:* describes the ease with which a system can be adopted to changes in the task specification.

  Different changes in the task specification have to be addressed differently with the two approaches. A change like 'besides city names do also extract organisation names' requires the addition of a few rules with the knowledge engineering approach, but requires the annotation of all organisation names in the document set with the (semi-) automatic training approach.

Appelt and Israel [1999] wrote "Although the core of an extraction system contains some domain independent components, domain-dependent modules are the rule. This domain dependence arises through the incorporation of domain-specific extraction rules, or the training of the system on a corpus of domain-relevant texts."

Because of these possible effects, the following aspects should be well examined before choosing one of the approaches. Such an examination will help to see with which one of these you would be better of [Appelt, 1999].

- *Available training data:* How much training data is necessary to get a good working rule set for a particular task and how much is available?
- *Available resources:* Are linguistic resources (e.g., lexicon, parser) as well as personal resources (e.g., knowledge engineer, annotator) available?
- *Stability of final specification:* Is it predictable whether or what kind of changes in the task specification will appear? If yes, which approach would make it easier to adopt or scale the system to cover a changed specification?
- *Required performance level:* Is a very high performance crucial for the systems' application?

4.4 Evaluation

A quantitative way to evaluate the performance of a system is always desired but cannot always be found, because of the very nature of its respective tasks. We have seen that the first attempt of performing a quantitative evaluation of IESs is introduced at the MUC-3. The conference organisation had a set of answer keys and had to compare these with the results of the IESs to measure the performance of a system. The nature of the IE task makes it possible to define input-output sets, which in turn makes it possible to carry out a quantitative evaluation over the IESs.

A lot of evaluation measures emerged over the years for evaluating different aspects of an IES [Lavelli *et al.*, 2004]. However, only two of them have been widely accepted: Recall and Precision.

Recall is a measure for evaluating the completeness of an IES, that is to determine how much of the relevant or correct information has been actually extracted. Whereas *Precision* is a measure for evaluating the correctness of an IES, that is to determine to which extent the extracted data is actually relevant or correct.

$$\textit{Recall} = \frac{correct\ answers}{total\ of\ possible\ correct\ answers}$$

$$\textit{Precision} = \frac{correct\ answers}{answers\ produced}$$

High Recall means that most of the available relevant information in the input data has been extracted. High Precision means that most of the extracted information was really relevant.

It is hard to optimise both values at the same time. If you work towards a high precision value, it is possible that relevant information in the input data will be missed or ignored. On the other hand, if you work towards a high recall value, it is possible that non-relevant parts of the input data will be extracted, too. Therefore, you have to decide on which aspect (completeness or correctness) you need to concentrate for your particular task.

Further, we cannot say that there are agreed-upon threshold values for these two measures that indicate the 'usefulness' of an IES, rather they differ according to different tasks. However, Cowie and Lehnert [1996] suggest that 90% Precision will be necessary for IESs to satisfy information analysts.

To weight the impact of the Recall and Precision values on the final evaluation the *F-Score* has been introduced that combines Recall and Precision in a single measure.

$$\textit{F-Score} = \frac{(\alpha + 1) * Precision * Recall}{(\alpha * Precision + Recall)}$$

The F-Score where Recall and Precision are evenly weighted is also called the F_1 measure and is computed as follows:

$$F_1 = \frac{2 * Precision * Recall}{(Precision + Recall)}$$

4.5 Challenges of Information Extraction

We have seen so far many aspects of IE and that decisions by the selection of modules and approaches for the IES can affect the end results immensely. Unfortunately, there are other factors as well that make IE even harder [Appelt and Israel, 1999]:

- *Language:* The language of the texts can make it necessary to include an additional module to capture the orthography and morphology of the language (e.g. German).
- *Text:* The structure of the input data itself can be a big obstacle. Textual data can be unstructured, semi-structured or structured, each of these require different handling. Sometimes the input data contains tabular data from which it is also very hard to extract information. The length of the input data is another issue. If the input text is too long than IR techniques might be necessary in order to identify the relevant parts to apply further IE processing on these parts only.
- *Genre:* The genre of the input data should be analysed before starting to build the IES, because data from different genres will require different handling. Whereas e-mails are free text and thus have no structure at all, scientific papers will have a specific format, which could be utilised.
- *Task:* As we have seen earlier, there are different tasks an IES can be build for. Whereas the entity identification is relatively simple the scenario template task or the coreference task will require additional modules in the IES.

The challenges that IES have to face can be summarised as follows [Yildiz, 2004]:

- *Higher Precision or Recall Values:* Since the IESs are evaluated with these two measures, every developer wants to achieve high recall or precision values. It is hard to optimise both, Recall and Precision at the same time. Sometimes it might be more important to extract all the information relevant without bothering that some of the extracted information is not relevant. In this case Recall has to be optimised. But if one wants that the extracted information is relevant without bothering much that some relevant parts will not be extracted, then Precision has to be optimised.

- *Portability:* IESs are in general developed for a specific task. Because interests can change over time, it could be necessary to adopt an IES to a new field of interest. An adaptation could be required in terms of the domain, the language, the text genre or the type of data.

- *Scalability:* The scalability problem can be examined further in two dimensions. First, an IES should be scalable in terms of the amount of data the IES is able to process. Second, an IES should be scalable in terms of the data sources it can handle.

4.6 Conclusion

The Message Understanding Conferences (MUCs) can serve as a starting point for any researcher in IE because of their educational potential, by giving a good overview of which obstacles the participant sites had to face over the years and how they had overcome them. Another thing that makes the MUCs so important is that the evaluation criteria and procedure commonly used in the IE community today originated from these conferences.

Two main approaches are common to build an IES: knowledge engineering approach and automatically learning approach. The **knowledge engineering approach** requires a knowledge engineer with the domain and task knowledge to build a set of rules. It takes much time, but the generated rule set will likely cover most dimensions of the task and the domain. To make the process less time-consuming, the **automatically training approach** tries to automate some or all parts of the rule generating process. It is perhaps less efficient, because the generated rules may not cover the domain and task as with the other approach, but the system depends not on the skills of a knowledge engineer anymore.

We can conclude that IE is not a trivial task at all and many factors have to be kept in mind before starting to build an IES in order to get a good one.

Part II

Ontology-Driven Information Extraction

Chapter 5

Ontology-Driven Information Systems

But sometimes you have to moan when nothing seems to suit yer
But nevertheless you know you're locked towards the future

So on and on I go, the seconds tick the time out
There's so much left to know, and I'm on the road to findout

On The Road To Find Out - Cat Stevens

In the first part of this thesis, we have stated that ontologies can be used to establish a common understanding about conceptualisations between interaction partners, enabling inter-operability between them and further the reuse of knowledge by third parties. We also examined several issues related to ontological engineering tasks that arise because of the nature of the ontology life cycle. In that context we examined existing approaches that address those tasks and their limitations.

It has to be stated that most of the available research regarding ontologies had the Semantic Web as their application field in mind. We know that the Semantic Web is an extended form of the current Web, where machine readable semantics are added to the content available on the Web [Berners-Lee, 1999]. As such, it represents a largely distributed and heterogeneous application field. However, these properties are not shared by many application fields where ontologies can be useful as well. Therefore, it has to be analysed where and how ontologies can be used and how the requirements to ontologies differ from the ones examined during the research done so far for ontologies in the Semantic Web.

Ontologies, being explicit specifications of conceptualisations [Gruber, 1993], can play a major role in many of todays Information Systems (ISs) as knowledge bearing artifacts. Hence their increased use in several application fields, which makes it possible to observe requirements for their smooth integration in several kinds of ISs.

In this chapter, we will examine how ontologies can be utilised to generate scalable and portable Information Systems (ISs) in general and Information Extraction Systems (IESs) in particular.

5.1 Ontologies for Information Systems

When we are going to build an IS we will have to provide the IS with some kind of domain and task knowledge. We cannot expect that the IS predicts what we want and just works like that.

If we want, for example, an application that can compute graph drawings with as little edge crossings as possible, we would have to tell the IS what a graph is, what kind of graphs we want to process (e.g. planar, non-planar), how an edge crossing is defined, etc. All this information will be, in general, implicitly coded in the systems' architecture. This implies, that other people, who want to build similar applications cannot make use of this implicit knowledge, unless they examine the code of the application.

Sometimes the required knowledge cannot be coded easily. Ontologies can help out here, because they are appropriate for representing many kinds of complex knowledge. Further, we have seen that ontologies are means for making this knowledge explicit, and so sharable and reusable.

So, it is not surprising that ontologies are in use in many ISs by now. But before using an ontology in an IS just because it is trendy, we should look whether it is adequate or necessary for the systems' intended purposes.

Guarino [1998] analysed the roles ontologies can play in ISs. By looking at the impact an ontology can have on an IS, he distinguishes between a temporal and astructural dimension. The *temporal dimension* describes whether an ontology is used at development time or run-time, whereas the *structural dimension* describes in which way an ontology can affect the components of an IS (i.e., application programs, information resources, and user interfaces).

Temporal dimension

Using an ontology *at development time* means that we have an ontology and that we have to build our system according to the conceptualisation represented by this ontology. Whether the ontology was an already existing one or we had to build it from scratch is not important here. The point is, that by using an ontology, the system developer has been freed from making conceptual analysis on his own (i.e., knowledge reuse) and that consistency is guaranteed among other systems that committed to the same ontology.

Using an ontology *at run time* can take two forms: ontology-aware IS and ontology-driven IS. An *ontology-aware* IS is a system that is aware of the ontology and can use it whenever needed. An *ontology-driven* IS, on the other hand, is a system where the ontology is yet another component of the system that co-operates with the other components [Guarino, 1998].

Structural dimension

Going further to the structural component, we can see that ontologies can be used in connection with the *application program component* as we have seen in the example about graph drawings. Most of such components contain the domain knowledge coded implicitly. At development time the ontology can help to generate these parts

where the knowledge was coded implicitly. At run time we can use the explicit knowledge in an ontology as a knowledge component for the system (compare to knowledge-based systems) which would increase the systems' maintainability, scalability and portability.

Ontologies can also be used in connection with *information resources*, such as databases. At development time they can help, for example, to generate database schemas, since they represent also a conceptualisation of a domain. At run time they can be used for information integration as a mediator between the incoming information and the database.

Their usage in connection with the *user interface component* may not be that obvious. If we see the user interface as a component that reflects the conceptualisation to the user, ontologies may help by building the user interfaces at development time. At run time the ontology could be made accessible to the user through the interface enabling the user to query and browse the ontology [Guarino, 1998].

5.1.1 Obstacles on the Way

Despite the benefits ontologies apparently can offer, it is not yet a common approach amongst IS developers to integrate and use ontologies in their systems. The main reason for that is perhaps that it still takes more time for a developer to build an ontology-driven application than a usual application. There are several factors related to the ontology life cycle (see Figure 3.1) that cause the additional time and costs required to build an ontology-driven IS:

- *Obtaining an ontology:* The first thing that has to be clarified before developing an ontology-driven IS, is how to obtain the ontology itself. Either, the developer will have to look for already existing ontologies in that domain or to generate a new ontology.
  - Ontology Import and Reuse: Although a large set of ontologies have been developed and made publicly available for many domains by now, the developer still has to understand the ontology in order to refine it for the particular task at hand.
  - Ontology Generation: If there are no appropriate ontologies available to integrate or to reuse, the developer will have to generate an ontology by hand or using (semi-) automatic ontology generation methods (see Section 3.2).
- *Maintaining an ontology:* It is most likely that the application field of an IS will undergo some changes over time. One may decide to either support a new domain or to support additional services, causing changes in the task specification. Such changes will make it necessary to change the underlying ontology as well.

To reduce the integration and run-time costs of ontologies in ISs, an Ontology Management Module (OMM) should be integrated into the IS that provides ontology

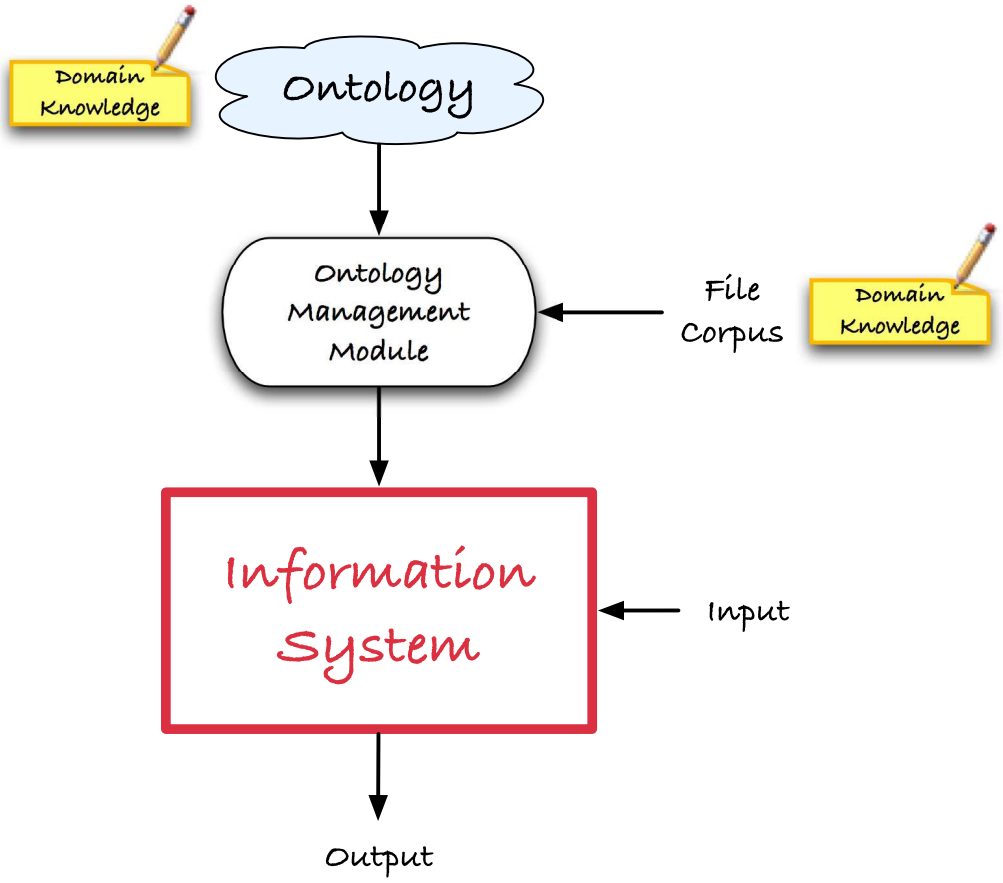

Figure 5.1: General architecture of an ontology-driven Information System

generation methods and accurate ontology management services. The general architecture of such an IS could be as in Figure 5.1.

The responsibilities of the OMM regarding the phases of the ontology life cycle differ from the general responsibilities for ontology management in conjunction with the Semantic Web. In a scenario where an ontology is used to capture the domain knowledge needed for an IS and where the focus is on portability and scalability, the requirements that the OMM has to reconcile are different.

By examining existing work in that field, one can observe that in many cases additional knowledge about components in the ontology is needed to perform the task at hand more accurately. Often researchers use an abstract ontology model to integrate existing ontologies and to enrich this knowledge with their proposed additional knowledge.

For the case of ontology learning from text documents for example, Cimiano and Völker [2005] argued in a similar way and attached a probability (confidence level) to ontological components learned by their system. Doing this, they aim to enhance the interaction with the user by presenting him the learned structures ranked according to their confidence level or by presenting him only results above a certain confidence threshold.

Tamma and Bench-Capon [2002] motivated an extended ontology model to characterise precisely the concepts properties and expected ambiguities, including which properties are prototypical of a concept and which are exceptional, as well as the expected behaviour of properties over time and the degree of applicability of properties to subconcepts. The authors claim that this enriched semantics is useful to describe what is known by agents in a multi-agent system. Since we are dealing with ISs in general and IESs in particular, not all of their proposed meta-properties are of interest for us. We use only the property describing the properties' expected behaviour over time, for it can help during the ontology management phase when the ontology has to be adapted to changes in the domain.

In the following we will take a look at the requirements to an OMM w.r.t. the phases of the ontology life cycle. Hereby, we think that the main challenge in the case of ontology-driven ISs is, that often all of these requirements have to be reconciled at the same time to provide the needed services, whereas in the context of the Semantic Web often only few of them are demanded.

Ontology Generation

It should not be hard to generate an ontology for a particular task specification at hand, if the ontology is not that large. However, changes in the task specification would require the adaptation of the ontology if not the generation of a new ontology. For an IS to be fully portable and scalable the generation process of an ontology should be automated. No matter how an ontology for an IS has been build, it is necessary to mark the ontological components with additional semantic knowledge indicating the level of confidence (property: *confidence_level*) the generator has in a particular ontological component. If the ontology is being generated by hand, the ontology developer has to model this kind of knowledge into the ontology. Whereas, if the ontology has been generated automatically, the generation module has to compute the level of confidence. Other modules of the system will likely use this kind of knowledge to base their decisions on.

Ontology Integration

To provide maximum flexibility, a scalable and portable IS should be able to react to new-coming standards. Further, it certainly should be able to combine different ontologies in different representation languages. To ease this procedure, the OMM should be based on an abstract ontology model, rather than on a particular representation language.

Change Management

An ontology used in conjunction with an IS should not be considered as a static artifact, because the changes in the task specification or the domain have to be reflected on the ontology as well. To automate the change detection in the domain, the OMM should preferably provide data-driven change detection. This can be achieved by providing the OMM with a file corpus of relevant documents to the domain. This

process would be further eased by marking components of the ontology with additional semantic knowledge indicating their estimated behaviour over time. In their proposed extended ontology model, Tamma and Bench-Capon [2002], propose an attribute (property: *value_change_frequency*), which indicates whether a component is allowed to change its value over time or not, by marking them with a value like 'final', 'frequent', etc.

There are two other additional components that are needed to allow automatic change detection from a file corpus: source-link components and change components. *Source-link components* represent links between the ontological structures in the ontology and their respective occurences in the file corpus. If documents are added to or removed from the file corpus, these links can be used to detect which components in the ontology are affected by the change. *Change components* represent actual changes in the ontology. Every addition, deletion, or edition can be represented in form of additional change instances, with appropriate properties about the kind of change, the date of change, etc. These change components also allow to keep track of the evolution of the ontology over time.

To provide some of these functionalities a developer may use the existing work of Cimiano and Völker [2005]. They present a framework for data-driven change discovery with several integrated ontology learning approaches. They represent the learned knowledge at a meta-level, using an abstract ontology model, which they call Probabilistic Ontology Model (POM). The integrated learning approaches in the ontology are able to learn is-a, instance-of, part-whole, and equivalence relations and restrictions on the domain and range of relations. Their POM also contains links of the ontological structures to corresponding documents from which they were derived; allowing the user to understand the context of a particular structure and allowing the system do react to changes in the document corpus. We think that both of these additions to the components of an ontology are essential for the use of ontologies in ISs. Further, they claim that systems that want to support data-driven change discovery have to keep track of all changes to the data. Such a system should also allow for defining various change strategies, which specify the degree of influence changes to the data have on the ontology or the POM respectively.

Having explored the requirements to ISs and related ontologies that have to be reconciled when ontologies are going to be used in ISs in general, we are going over to our actual focus, namely IESs. In the next section, we will examine how ontologies can be utilised to generate more scalable and portable IESs.

5.2 Ontologies for Information Extraction Systems

Information Extraction (IE) is currently an important and popular research field, for it tries to extract relevant information from the overwhelming amount of data we are facing on a daily basis. In Chapter 4, we defined the term of IE and gave a little bit of historical background information to the research field and its challenges.

In Section 4.3 of the mentioned chapter, we have seen that there are two approaches to IE, namely the knowledge engineering approach and the (semi-) automatic training approach. Ontologies can be used in conjunction with both approaches as a specification of the conceptualisation of the current domain and task, that is the specification of the relevant information the system actually has to find.

With the knowledge engineering approach, the knowledge engineer provides the system with extraction rules that cover the domain and task of the IES. Using an existing ontology, the developer can commit to the ontology by generating the rules (compare Figure 5.2), and does not have to perform a domain analysis by her own, unless she has to develop the ontology by herself as well. Using an ontology in combination with this approach can increase interoperability between systems that commit to the same ontology.

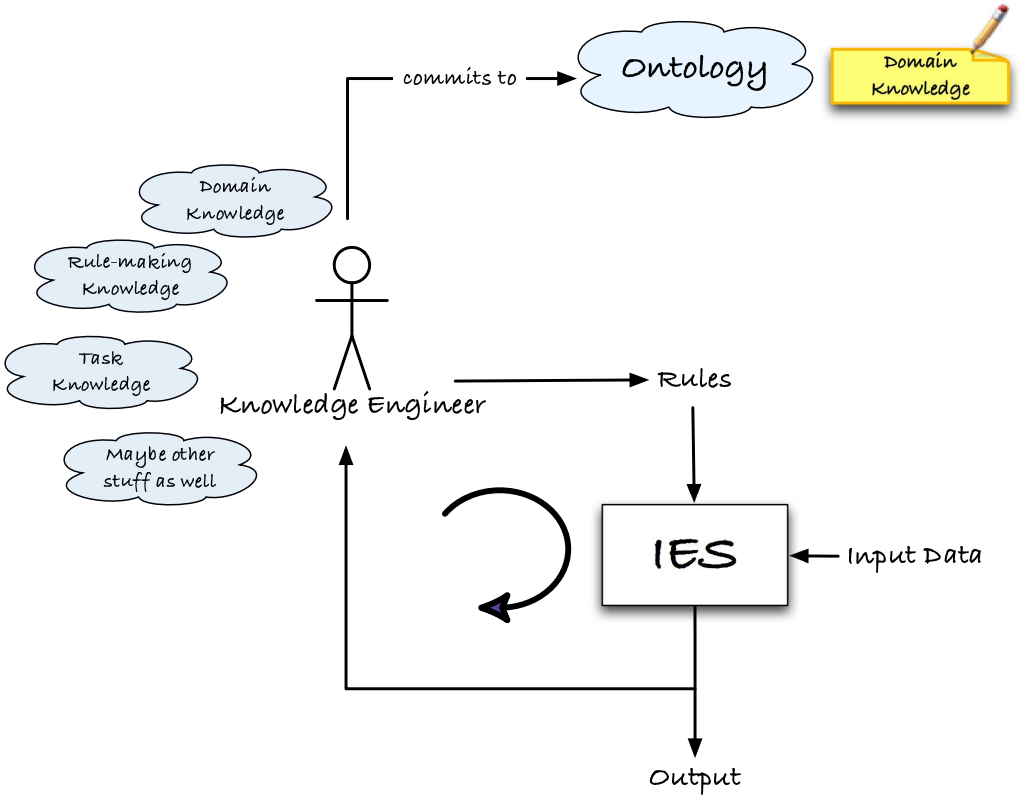

Figure 5.2: Knowledge engineering approach to IE using an ontology

With the (semi-) automatic training approach the aim is to automate some or all parts of the rule generation process to decrease the human intervention and thus to decrease the development time of an IES. For this approach, one or more human annotators had to mark relevant pieces of data in a large document set, from which the system then can learn extraction rules to extract also information from unseen data. However, annotators often do not agree among themselves about the relevancy of pieces of data. Ontologies can be used here (compare Figure 5.3) to achieve a consensus about relevant data by specifying the task knowledge in an unambiguous way. It is clear that neither the time needed to do the annotations, nor the time to

adapt the annotations when the specification or the domain changes is reduced just because using an ontology.

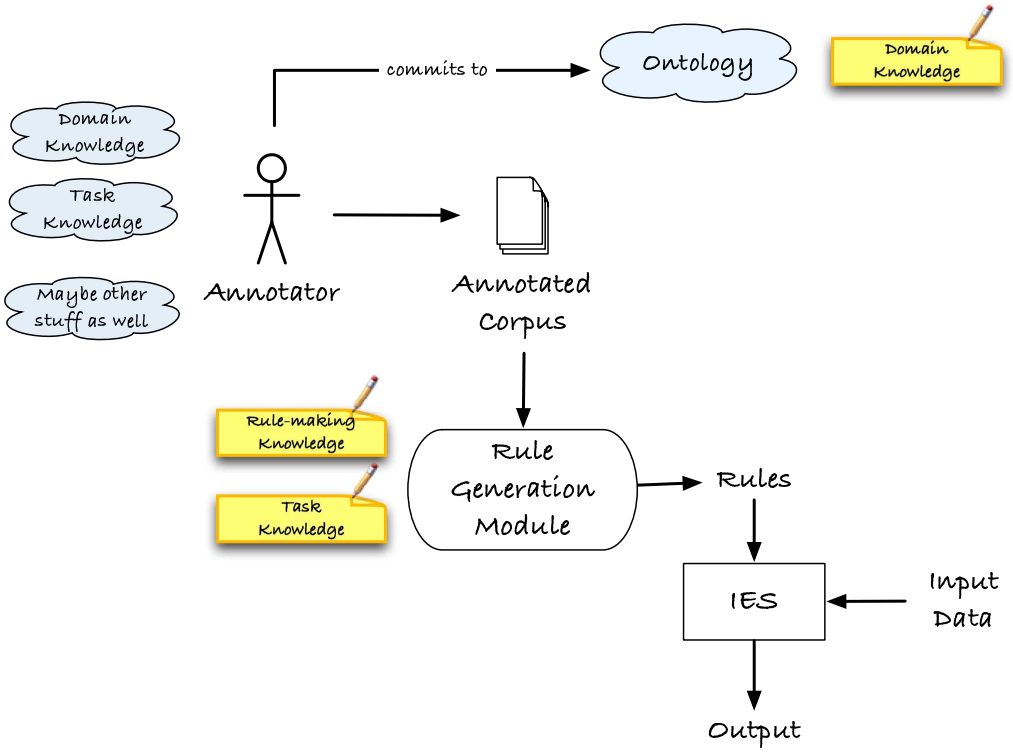

Figure 5.3: (Semi-)automatic training approach to IE using an ontology

Aitken [2002] presents an approach to learn information extraction rules from natural language data using Inductive Logic Programming (ILP). He proposes the use of an ontology as a reference to which an annotator can commit to while annotating the data with ontology terms. The supervised induction algorithm then uses the annotations to generate extraction rules.

Besides the two mentioned scenarios where an ontology can be used to which either the knowledge engineer or the annotator can commit to, one more scenario could be of interest. Namely, to automate the rule generation process fully by using an ontology as a source of domain knowledge and task specification (see Figure 5.4).

This scenario particularly aims to facilitate the portability and scalability of an IESs. Portability and scalability are important challenges, IES developers have to face. As we have seen, the whole generation process is not easy and takes time. Therefore it is understandable that changes in the domain or the task specification are not at all welcome. In the worst case, such a change would mean that the rule generation process has to be performed again. IESs that follow the fully automatic approach using ontologies are much more easier to adopt to changing specifications and domains. However, the nonexistent human intervention will most likely yield to a decrease of the system performance. To compensate this effect, the ontology

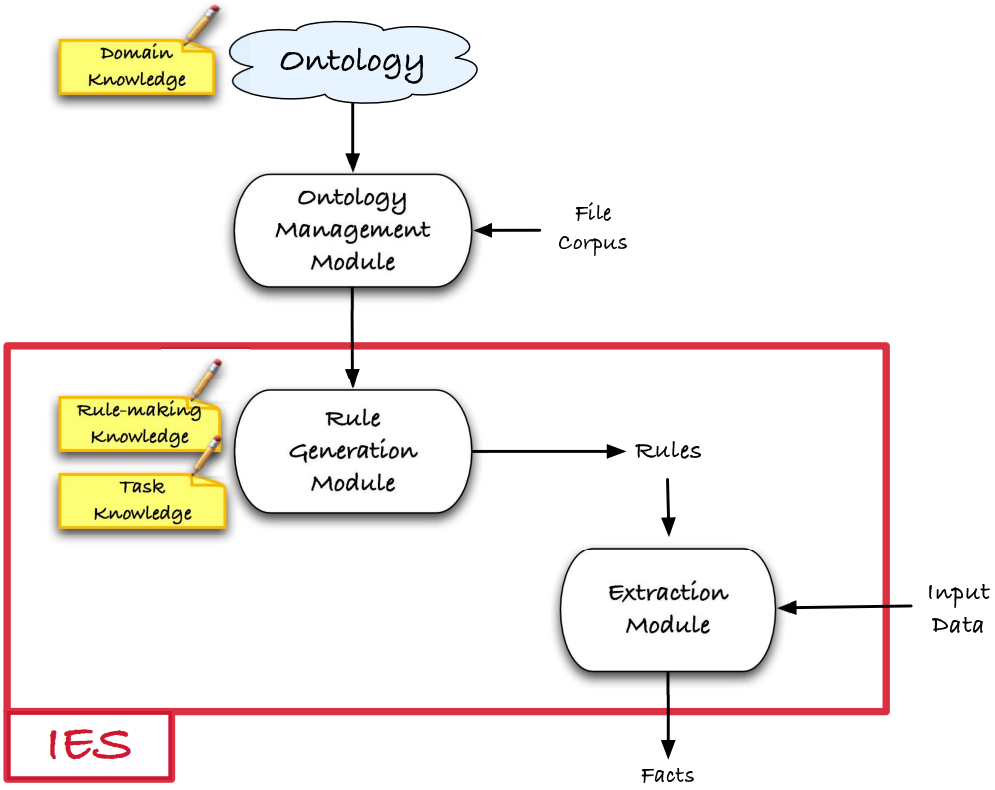

Figure 5.4: Fully automatic approach to IE using an ontology

that is being used has to reconcile several requirements, which are stated in the next section.

5.2.1 Requirements to Ontologies in IES

The IE community has made already the attempt in the direction of ontology-based IE. However, existing work differs, as usual, along several dimensions:

- What kinds of ontologies are used?
  The question of which kind of ontology should be used in an IE process is not easy to answer, because the different ontology types are useful for different scenarios. The level of detail of an ontology is very important for the performance of an IE tool. It is not always necessary that the ontology models the domain of interest with all its relations to full extent. Therefore, the intended function of an IE tool should be analysed before an ontology type is chosen.

- How the ontologies were generated?
  Ontologies can be generated either manually, semi-automatically or automatically. As with the rule generation process the manual approach would take more time, but the generated ontology would be likely at the right level of generality. The semi-automatic and automatic approaches would take less

time, but we could not be sure that the generated ontology would contain all the information we wanted it to contain. Again, the intended function and the required level of performance of the IE tool has to be kept in mind while choosing the way in which the ontology should be generated.

- At which state in the whole IE process are the ontologies used?
  As we have seen, an ontology, can be used as a specification of the relevant conceptualisation to aid human beings involved in the IES generation process. An ontology can also be used at run-time, whereas several components of the IES can access the ontology to utilise its content for their tasks.

To enable the smooth integration of ontologies within IESs several requirements have to be reconciled. The components of the input ontology should contain few additional properties, which are essential for the Rule Generation Module (RGM) to produce accurate rules to be used to extraction information from input data.

- *Quality Properties:* We mentioned before (see Section 5.1.1) that it is important for ISs to have knowledge about the confidence level of the components in the ontology (property: confidence_level). In the case of IESs this becomes even more important, because the levels are needed to compute the confidence levels of the rules themselves. These computed levels are used to choose between rules, when more than one rule can be applied on a certain part of data.

- *Value Constraint Properties:* Value constraints are used to restrict property values such as the data type or cardinality. It is already possible to state this kind of knowledge in ontology representation languages such as OWL. This additional information will be used in an IES to state the rule conditions under which the rule can be applied.

- *Temporal Properties:* In many settings the components of an ontology have to be marked with temporal values such as the transaction time (property: *transaction_time)*, or valid time (property: *valid_time_begin* and *valid_time_end*) of the component. These properties are especially useful in connection with changing ontologies where out-of-date components are not deleted from the ontology but marked as such. Because, in a common scenario where the IES has to be able to extract information from new and relatively old data alike, a completely up-to-date ontology would not serve the purpose.

In existing work about ontology-driven IE approaches, we can see this trend of enriching ontological components with additional semantic knowledge in order to perform the extraction task more accurately.

Embley [2004] presents an approach for extracting and structuring information from data-rich unstructured documents using extraction ontologies. He proposes the use of the Object-oriented Systems Model (OSM) [Embley *et al.*, 1992] to represent extraction ontologies, because it allows regular expressions as descriptors for constants and context keywords. Both, the generation of the ontology and the generation of the regular expressions are being done manually. The ontology is then

parsed to build a database schema and to generate extraction rules for matching constants and keywords. After that, recognisers are invoked which use the extraction rules to identify potential constant data values and context keywords. Finally, the generated database is populated using heuristics to determine which constants populate which records in the database. For the extraction of relevant information from car advertisements, the presented approach achieved recall ratios in the range of 90% and precision ratios near 98%. For domains with more complex content and where the relevant records (e.g., car advertisements) are not clearly separated from one another, the performance decreases, though.

However, our aim is to provide an unsupervised ontology-driven IES, that is able to exploit the content of its underlying ontology on its own for extracting relevant information from natural language texts.

Chapter 6

ontoX - An ontology-driven IES

Then I found my head one day when I wasn't even trying
And here I have to say, 'cause there is no use in lying, lying

Yes the answer lies within, so why not take a look now?
Kick out the devil's sin, pick up, pick up a good book now

On The Road To Find Out - Cat Stevens

In this section, we will explain our method for IE from natural language text, which tries to utilise the knowledge in an ontology. The input ontology is being used as a knowledge bearing artifact that represents the conceptualisation of a domain of interest and the task specification for the extraction task. Our aim by developing this method is to provide a means for 'common' people to perform IE in a way that requires neither skills in particular rule representation languages, nor any other resource but the ontology, such as lexicons, etc.

Although our method can be applied to any other representation language, our implementation supports ontologies formulated in the Web Ontology Language (OWL) version 1.0 [Horrocks *et al.*, 2003] [Grigoris and van Harmelen, 2004]. We decided ontoX to process OWL ontologies because of several reasons:

- *Well-defined semantics:* OWL has been provided with a well-defined semantics. This semantics assures the unambiguous specification of a conceptualisation of interest, which is essential for IESs.
- *Predefined data types:* Several built-in XML Schema data types can be used in OWL, which makes it possible to formulate extraction rules that can capture values of these data types.
- *Popularity:* OWL became a W3C (World Wide Web Consortium) Recommendation in 2004 and enjoys a large-scaled popularity among the Semantic Web research community. This popularity led to the development of several ontology generation tools that support OWL and also to many OWL ontologies about different domains.

- *Tool support:* As mentioned before, many tools for the generation of ontologies have been developed, allowing the more-or-less user friendly development of OWL ontologies. This makes it easier for ontoX users to develop their own ontologies representing their own domains of interest.

The main idea behind our approach is that ontology representation languages are in general provided with pre-defined semantics and that this semantics can be exploited to 'understand' the conceptualisation they convey, whereas the conceptualisation represents the task and domain knowledge that is needed by an IES to perform actual extraction on texts.

The main architecture of our IES, ontoX, is depicted in Figure 6.1. Here, the user of the system has to provide the IES with an ontology that represents her domain of interest and at the same time contains the task specification as well. The Ontology Management Module (OMM) takes this ontology and tries to exploit the knowledge in it to determine what exactly has to be extracted from the input data. The Rule Generation Module (RGM) uses the output of the OMM and performs several steps to formulate rules (in our case regular expressions) to locate candidate values that are relevant according to the input ontology. The Extraction Module (EM) takes these rules and determines candidate values in the input texts and applies several heuristics to choose the most accurate values from them. This module finally returns the extracted values and also suggestions to the user regarding possible changes in the ontology.

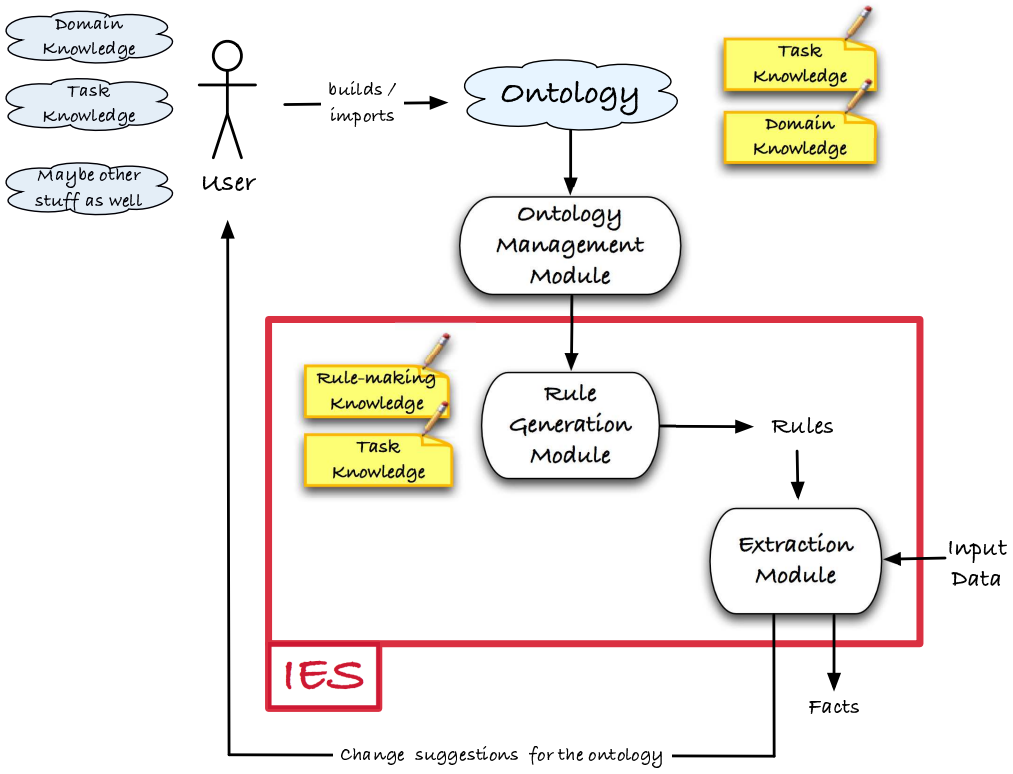

Figure 6.1: Main architecture of ontoX

To support the understandability of our extraction method and its implementation ontoX, we are going to explain them on an example. For that purpose we have chosen the domain of digital cameras, because the domain is known to many people and its characteristics are suitable for extraction.

Figure 6.2 shows a possible conceptualisation of the domain containing only relevant aspects of the domain we are interested in, that is the model name, the number of megapixels, the optical zoom factor, the digital zoom factor, and the screen size. In this sense, we can think of the ontology as the task specification where the properties of concepts represent the properties for which the extraction method has to determine appropriate values from input data.

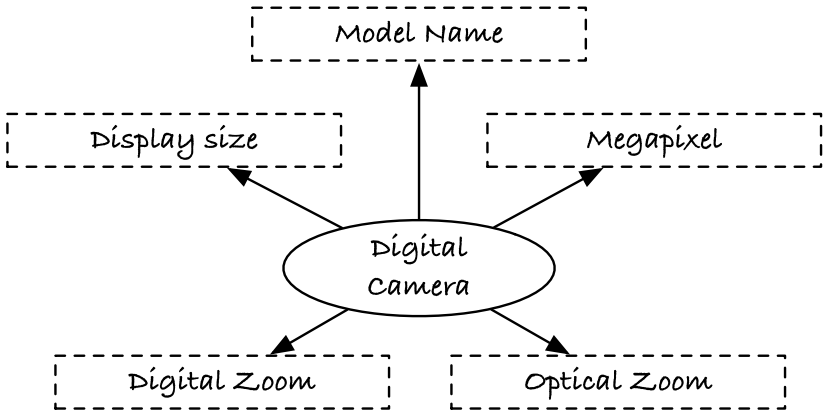

Figure 6.2: Basic digital camera ontology

To evaluate our extraction method, we also collected a set of digital camera reviews from the Web[1]. Below, you can see how a typical review looks like. The depicted review contains all relevant data we are interested in and is therefore a good example to explain our method with. The digital camera ”D-595 Zoom” apparently is a 5.0 megapixel camera with an 3x optical zoom lens, supports 4x digital zoom, and has a 1.8-inch LCD screen.

Although our method can process any valid ontology that is conform with OWL 1.0, several requirements have to be reconciled by the ontology to get acceptable results (compare Section 5.2.1). In the following we will take a look at those requirements and their implementation in a concrete ontology for the digital camera domain.

6.1 Input Ontology of ontoX

In the previous chapter we defined different requirements that have to be reconciled by ISs to provide accurate services based on ontologies. We also stated several additional requirements on ontologies when they are going to be used in IESs. In this

[1] http://www.steves-digicams.com/

```
The 5.0-megapixel D-595 Zoom combines the power of manual control
with the convenience of an easy-to-use design for incredible
results. Featuring manual control of shutter speed and aperture
options, a 3x optical zoom and 19 shooting modes, the D-595 Zoom
combines manual settings with the ease of a point-and-shoot,
packing high-end advanced features into an affordable, easy-to-
use compact camera.

The 3x optical zoom lens (38-114mm equivalent in 35mm photography
f2.8 - f4.9) combines with a 4x digital zoom to deliver a total
12x zoom so virtually no photo opportunity is out of reach. And
with the super macro mode it is possible to capture amazingly
small details, as tiny as the date on the back of a penny, from
as close as 0.8 inches.

TruePic TURBO is a super-charged image processor that
significantly enhances image quality and processing speed. Users
will experience rapid startup, shutter release and playback, as
well as capture sharper, more realistic images. One of the
greatest benefits of digital photography is the ability to
compose and review images on the LCD, as it allows users to
accurately frame their photographs and make on-the-spot decisions
about saving or deleting them. To ensure that users can easily
view their subject and frame the best-shot possible, the D-595
Zoom has a, 85,000 pixel 1.8-inch Semi-Transmissive TFT LCD - for
easy viewing even in bright sunlight.
```

Figure 6.3: Sample input data representing a digital camera review

section, we will see how these additional requirements can be reconciled by extending ontologies with additional properties. All these properties can be attached to ontological constructs in OWL using the `owl:AnnotationProperty` element.

6.1.1 Keywords

The most important requirement to enable ontology-based IE is to enrich the ontological constructs with keywords (i.e., trigger words) that indicate the presence of relevant information in the input text.

Although there is no explicit ontology construct to formulate this kind of knowledge in OWL 1.0, our implementation utilises comma-separated words in the 'comment' section (`rdfs:comment`) of ontological constructs as keywords. If the user does not provide a property with corresponding keywords, our system tries to extract appropriate values considering frequent terms in the neighbourhood of other properties' keyword occurrences. Note, that this can only return feasible results when enough properties co-exist and not more than one such unknown property has the same data type.

6.1.2 Constraining Properties

Constraining properties are required to narrow the search range of possible values of properties that have to be find. In our method, this is possible as far as the OWL 1.0 specification allows it. This means, that our extraction system only takes the XML Schema data types into account whose usage is legal in OWL 1.0[2].

To increase the performance of the system, the user should state the data type with the minimal data range. For example, for a property *megapixel*, she should not state the data type to be merely a `xsd:string`, but should state that it is a `xsd:float`. In that way the system can rule out many candidates, resulting in more accurate values.

It would be much better to have means for stating more complex constraints on the values of properties, such as minimal or maximal values, etc. In fact the next version of OWL (OWL 1.1) will provide means to state user specified data types.

6.1.3 Quality Properties

Previously we mentioned that a property stating the confidence of the ontology engineer in a particular ontological construct (*confidence_level*) could be useful for several reasons and in context with different purposes. In the context of IE this property represents the confidence level of the ontology engineer w.r.t. the correctness of the ontological component. The difficulty with such a *confidence_level* for a whole ontological construct is that it is not clear to which aspect of the construct it refers. Is it now the level of confidence of the ontology engineer that this concept is really relevant, or does it reflect the certainty of the ontology engineer about the details of the concept (e.g., place in ontology in terms of hierarchical order, etc.)?

In our particular method we think of this property in its former sense and allow the user to enrich the constructs in the ontology by the property *confidence_level*, which can take values from $[0, 1]$. This property helps our extraction method to make decisions when the same value is tried to be assigned to two different properties. In such a case, the property with the higher confidence level would be the winner.

Another property that can be attached to ontological constructs is the *relevance* property, which can take two values $\{$true, false$\}$. By marking a construct with this property the user can tell the system that she is not interested in the construct as far as the task specification is concerned, but rather that the construct is part of the domain of interest. An ontology that consists of the main concepts of a domain, although not all of them are relevant for extraction, can help the system to 'understand' the context of the task specification better. For such constructs, the system is going to find appropriate values just as if they were relevant, decreasing the risk that their values could be assigned to other constructs. The only difference to other constructs in the ontology is, that assigned values will not be presented to the user. If the ontological construct whose *relevance* property is said to be false is a property, its value will not be presented to the user; whereas if it is a concept, none of the values assigned to its properties will be presented to the user.

[2]For a full list of OWL 1.0 data types see Appendix B

6.1.4 Temporal Properties

Temporal properties can be useful in the context of IESs to enable two kinds of services. The first one is to enable temporal extraction and the second is change management. With the first one, a user can state that she wants her input data to be extracted using ontological components that were valid at a certain point of time. With the second one, the user can be provided with suggestions regarding out-of-date concepts because they did not appear in the input texts anymore, so that she could adopt the ontology if necessary.

To enable both kinds of services, we suggest the use of *valid_time_begin* and *valid_time_end* properties that can be stated for every construct in the ontology. By doing this, the input data will be analysed using only ontological constructs that are valid at the given point of time, which must be provided by the user of the system in a way that is conform with the `xsd:Date` data type. If no date is supplied all constructs in the ontology will be used for extraction.

To enable change management we also provide the user with the property *value_change_frequency* which can have two values: stable and frequent. The user can mark ontological components with this property in order to indicate that the component will appear in the input text consequently (i.e., stable) or not (i.e., frequent). Using this value, an IES can compute accuracy levels for each component after each extraction looking at whether the component had appeared in the input or not.

6.2 Ontology Management Module of ontoX

An ontology that can be used to extract information from digital camera reviews and that reconciles the requirements we just mentioned, can be seen in Figure 6.4. This ontology includes data types of the concept properties and additionally trigger words that might indicate the occurrence of appropriate values of the properties in the input text. The OWL document that represents this ontology can be found in Appendix A.

The Ontology Management Module (OMM) in our system is responsible for processing the input ontology to determine attribute-value pairs that constitute the actual extraction task. For that purpose, the OMM, loads the ontology into its intern ontology model and tries to exploit the pre-defined semantics of the underlying representation language (OWL 1.0). The used ontology model is the one provided by the Jena Semantic Web Framework[3], which provides a programmatic environment for RDF, RDFS, and OWL.

Because the ontology is going to be used for extraction purposes, not all modeling primitives of OWL are relevant for us. In this section, we will take a look at the relevant modeling primitives of OWL and how they are going to be interpreted by ontoX. For a more detailed description of OWL's modeling primitives, you are referred to the excellent book by Antoniou and van Harmelen [2004].

[3] http://jena.sourceforge.net/

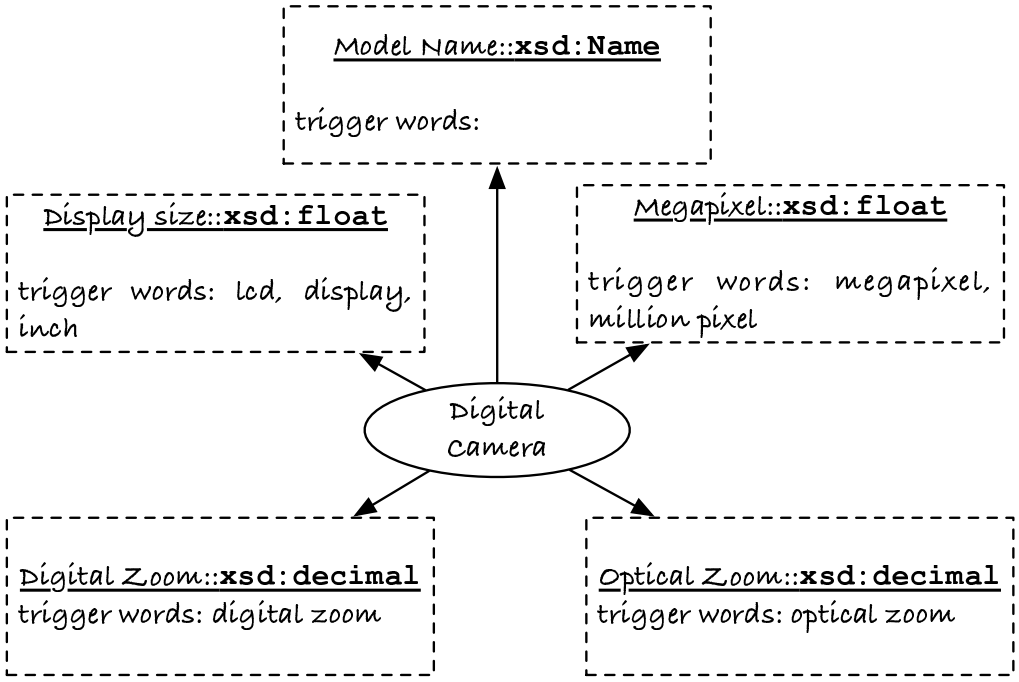

Figure 6.4: Graphical digital camera ontology used for extraction

6.2.1 Class Elements

Class elements are the basic ontological components that represent relevant concepts in a domain of interest. The OMM takes each defined class in the ontology as a particular concept and tries to identify attribute-value pairs to cover the task specification by determining all properties that are defined for this class. There are different ways to define a class in OWL. Some of these ways enable the definition of anonymous classes and are neglected by our OMM, because they apparently cannot have properties.

Direct Definition

A class can be defined directly using the `owl:Class` element and by stating its identification using the `rdf:ID` element.

```
<owl:Class rdf:ID = "digital_camera">
 <rdfs:comment> concept of a digital camera </rdfs:comment>
</owl:Class>
```

Subclass Declaration

It can be stated that a class is a subclass of an already defined class with the `rdfs:subClassOf` element. In this case, the OMM generates the same attribute-value pairs for this class as for its superclass.

```
<owl:Class rdf:ID = "digital_camera">
 <rdfs:subClassOf rdf:resource="#camera" />
</owl:Class>
```

Equivalence Declaration

It can also be stated that a class is the equivalent of an already defined class. Again, the OMM generates the same attribute-value pairs for this class as for its equivalent class.

```
<owl:Class rdf:ID = "Digicam">
 <owl:equivalentClass rdf:resource="#DigitalCamera" />
</owl:Class>
```

Enumeration Boolean Combinations

In OWL, a class can further be defined using the `owl:oneOf` element to enumerate all its elements or by boolean combinations of already existing classes. For example, it is possible to state that an instance of the class 'MovieFormat' can be one of the mentioned formats. Having such an enumeration of elements, our system simply has to look whether the input text contains any of these terms.

```
<owl:Class rdf:ID = "MovieFormat">
 <owl:oneOf rdf:parseType = "Collection">
   <owl:Thing rdf:about = "#QuickTime" />
   <owl:Thing rdf:about = "#AVI" />
   <owl:Thing rdf:about = "#MPEG" />
 </owl:oneOf>
</owl:Class>
```

6.2.2 Property Elements

Property elements are components to define the characteristics of class elements. Therefore, they are the main constructs of interest of our OMM. In OWL, two kinds of properties can be defined: object properties to relate objects with each other, and data type properties to relate objects with data types.

Object properties can be defined using `owl:ObjectProperty` elements. The domain and range of both kind of properties can be defined using `rdfs:domain` and `rdfs:range` elements. For each object property, our OMM collects defined instances of the class that is stated as the properties' range (i.e., `rdfs:range`) and use them as search strings in the input text.

```
<owl:ObjectProperty rdf:ID = "movie_format">
 <rdfs:domain rdf:resource="#DigitalCamera" />
 <rdfs:range rdf:resource="#MovieFormat" />
</owl:Class>
```

Data type properties can be defined using `owl:DatatypeProperty` elements. The range of such properties have to be one of the allowed XML Schema data types

(see Appendix B) in OWL 1.0. Our OMM generates for each data type property an attribute-value pair as part of the task specification where the attribute is the name of the property and the value is the allowed data type for that property (e.g., <megapixel, xsd:float>).

```
<owl:DatatypeProperty rdf:ID = "megapixel">
 <rdfs:domain rdf:resource="#DigitalCamera" />
 <rdfs:range rdf:resource="http://www.w3.org/2001/
      XMLSchema/#float" />
</owl:Class>
```

Property Restrictions

Property restrictions can be used to define subclasses of classes that satisfy certain conditions regarding some of their properties, using `owl:Restriction` elements. These restriction are interpreted by ontoX to assign correct values to properties.
The `owl:hasValue` element can be used to state that a property has to have a certain value.

```
<owl:Class rdf:ID = "SonyDigicam">
 <rdfs:comment>
   SonyDigicam is a Digicam that has a Memory Stick as
   storage medium.
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#Digicam" />
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#storage_medium" />
     <owl:hasValue rdf:resource = "#MemoryStick" />
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The `owl:allValuesFrom` element can be used to state that all values of a property must be from the specified class.

```
<owl:Class rdf:ID = "LithiumIonCamera">
 <rdfs:comment>
   LithiumIonCameras are cameras that support Lithium Ion accus.
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#Camera" />
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#hasBattery" />
     <owl:allValuesFrom rdf:resource="#LithiumIonAccu" />
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The `owl:someValuesFrom` element can be used to state that the values of a property can be from a specified class.

```
<owl:Class rdf:ID = "VideoDigicam">
 <rdfs:comment>
   VideoDigicams are Digicams that support some movie format.
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource = "#Digicam" />
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#movie_format" />
     <owl:someValuesFrom rdf:resource = "#MovieFormat" />
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

In OWL 1.0 it is also possible to state constraints on the cardinality of values of properties using the `owl:minCardinality` element, the `owl:maxCardinality` element and the `owl:cardinality` element. For example, we can state that a digital camera has to support at least on storage medium.

```
<owl:Class rdf:about = "DigitalCamera">
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#hasStorageMedium" />
     <owl:minCardinality rdf:datatype=http://www.w3.org/
        2001/XMLSchema/#nonNegativeInteger">
      1
     </owl:minCardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Special Properties

In OWL 1.0 properties can also have properties. For instance, we can state that a property is transitive using the `owl:TransitiveProperty` element, or the `owl:SymmetricProperty` element to state that a property is symmetric.

An important property for ontoX is the one to state that a property is functional, that is that it has at most one value for each object. Such properties are specified using the `owl:FunctionalProperty` element. It can also be stated that two objects cannot have the same value for a property using the `owl:InverseFunctional Property` element.

```
<owl:FunctionalProperty rdf:about = "megapixel" />

<owl:InverseFunctionalProperty rdf:about = "model" />
```

6.3 The Rule Generation Module of ontoX

The Rule Generation Module (RGM) of ontoX is responsible for the generation of extraction rules that can be used to identify candidate values for the attribute-value pairs generated by the OMM. Because, OWL 1.0 supports only a pre-defined set of data types, at the present this module merely has to look up appropriate regular expressions for the stated data types.

However, for other ontology representation languages the RGM would have to do a lot more. For the oncoming standard OWL 1.1, which will support user-defined data types using a set of pre-defined data types, for example, the RGM would have to parse the data type definitions in order to generate corresponding extraction rules.

6.4 The Extraction Module of ontoX

The Extraction Module (EM) of ontoX is responsible for applying the rules generated by the RGM and using them to identify candidate values for properties in the ontology. The steps that are taken by the EM can be considered in two main parts: preprocessing step and extraction step. In the following we will explain the particular tasks that are being performed by our method during these two main steps.

6.4.1 Preprocessing

As in any other text processing task, several steps have to be taken to eliminate noisy data from the input, which could affect the performance of the algorithm. Further, preprocessing is needed to transform the input data into a format that can be processed more easily by successive modules. The preprocessing phase of our extraction method comprises the following particular steps.

Removing stop words

Natural language texts are usually loaded with words, called stop-words, which do not convey relevant information but occur frequently in the text, because they are needed to build grammatically correct sentences. Because of their high frequency, they often cause outliers in term frequency histograms and have to be eliminated from the input text before starting the actual extraction phase.

As we evaluate our method with texts in English we determined the stop words commonly used in English. Our stop word list contain the following: ’a’, ’an’, ’and’, ’are’, ’as’, ’at’, ’be’, ’but’, ’by’, ’for’, ’if’, ’in’, ’into’, ’is’, ’it’, ’no’, ’not’, ’of’, ’on’, ’or’, ’s’, ’such’, ’t’, ’that’, ’the’, ’their’, ’then’, ’there’, ’these’, ’they’, ’this’, ’to’, ’was’, ’will’, ’with’.

Locating Data Type Occurrences

We stated earlier that data type properties represent our main interest w.r.t. extraction, for they define the actual values of interest to be extracted. Therefore, our whole method is centered around these ontological components. However, the

XML Schema data types have different value spaces, which ranges from integers, bytes, floats to more general data types such as strings. It is clear, that the more constraints stated about a property the easier will it be to identify a particular text as a value of that property. In order to rule out cases where the data types with wider value spaces overlap data types with narrower value spaces, we begin with locating occurrences of the narrower ones. For example, we first identify `xsd:float`'s and then `xsd:decimal`'s in order to avoid cases where parts of a token '5.0' could be identified as `xsd:decimal` values (i.e., '5' and '0'), whereas it should be identified as a whole to be a `xsd:float` value (i.e., '5.0').

Values of data types derived from xsd:string are much more harder to identify then numbers and other data types that have fixed formats. Therefore, simple pattern matching methods cannot be applied on them. For these data types, especially for the data type xsd:Name, we use a heuristic that computes the *named_entity_probability* of a string. To compute this value, we consider the anomalies in the string that could indicate that the string at hand is not a proper word, like as numbers and characters appearing in the same string, or mixed lower and upper case usage of characters.

Algorithm 1 Compute Named Entity Probability

Input: a set $I = \{s_1, ...s_n\}$ of strings that matches the definition of the data type `xsd:Name`

Output: sorted set $I' = \{s'_1, ...s'_2\}$

```
1:  for each string s in I do
2:      p := 0
3:      for each character c in string s do
4:          if c is an upper case character then
5:              p := p + 1
6:          else if c is a digit then
7:              p := p + 2
8:          end if
9:      end for
10: end for
11: I' ← sort strings in I according to their p value
12: return I'
```

In the next section we will see, how our algorithms and heuristics approach the task of extracting the kind of information as it was formulated in the input ontology.

6.4.2 Extraction

The extraction phase is the actual phase where the located data type values are going to be assigned to corresponding properties in the ontology. Figure 6.5 depicts the sample input data we used (compare Figure 6.3) after the preprocessing phase. As you can see, all the stop words have been removed and all occurrences of data type values that are allowed in OWL 1.0 are underlined. We took a step further and

highlighted also the keywords that were present in the text by underlining them with two lines (i.e., 'megapixel', 'optical zoom', 'digital zoom', 'inch', 'LCD').

5.0-megapixel D-595 Zoom combines power manual control convenience easy-to-use design incredible results. Featuring manual control shutter speed aperture options, 3x optical zoom 19 shooting modes, D-595 Zoom combines manual settings ease of point-and-shoot, packing high-end advanced features affordable, easy-to-use compact camera.

3x optical zoom lens (38-114mm equivalent 35mm photography f2.8 - f4.9) combines 4x digital zoom deliver total 12x zoom so virtually photo opportunity out of reach. super macro mode possible capture amazingly small details, tiny date back penny, from close 0.8 inches.

TruePic TURBO super-charged image processor significantly enhances image quality processing speed. Users experience rapid startup, shutter release playback, capture sharper, more realistic images. One greatest benefits digital photography ability compose review images LCD, allows users accurately frame photographs make on-the-spot decisions about saving deleting them. ensure users can easily view subject frame best-shot possible, D-595 Zoom has, 85,000 pixel 1.8-inch Semi-Transmissive TFT LCD - easy viewing even bright sunlight.

Figure 6.5: Sample input data after preprocessing

Now we have to assign appropriate values to the properties in the ontology that the identified keywords belong to. For that purpose, we are looking for values that are conform with the predefined data type of the property. In fact, we are looking for the first such value at the left side and the right side of a keyword occurrence, because it is more likely that the values are located near the keywords. For the keyword 'megapixel' that must have a value conform with xsd:float, we have the value '5.0' at its left and no appropriate value at its right, because there is a sentence boundary between the keyword and the next valid value ('3'). So we can add the value '5.0' into our list of candidate values for the property 'megapixel' in our input ontology. The heuristic used to choose appropriate candidate values for each keyword occurrence is formulated in Algorithm 2.

A property may have been provided with more than one keyword, in which case every occurrence of each keyword would be encountered to collect candidate values in the input text. While collecting candidate values, our method marks them with a level of *evidence* that is computed as

$$evidence = \begin{cases} \frac{1}{d} & \text{if } d > 0 \\ 1 & \text{if } d = 0 \end{cases} \tag{6.1}$$

whereas d is the distance of the candidate value from the keyword occurrence in terms of the words that lie between them. This inverse distance function is used to favour data type values that are near to the keyword over values that are more far

Algorithm 2 Select Candidate Values

$L_k \rightarrow$ List of all keyword occurrences in $Input$
$L_d \rightarrow$ List of all data type occurrences in $Input$

1: **for each** keyword k in L_k **do**
2: $r \leftarrow$ k.datatype ▷ look up the data type of keyword k, e.g., xsd:float
3: $dp \leftarrow$ k.property ▷ look up property dp to which keyword k belongs
4: **repeat**
5: $c_l \leftarrow$ next token at the left
6: **until** c_l is a valid value or a sentence boundary
7: **repeat**
8: $c_r \leftarrow$ next token at the right
9: **until** c_r is a valid value or sentence boundary
10: **if** c_l == sentence boundary and c_r is not **then**
11: candidate_value $= c_r$
12: **else if** c_r == sentence boundary and c_l is not **then**
13: candidate_value $= c_l$
14: **else** ▷ both, c_l and c_r are valid values
15: $c_l.evidence =$ compute evidence for c_l
16: $c_r.evidence =$ compute evidence for c_r
17: **if** $c_l.evidence > c_r.evidence$ **then**
18: candidate_value $= c_l$
19: **else if** $c_r.evidence > c_l.evidence$ **then**
20: candidate_value $= c_r$
21: **else** ▷ both values have the same evidence
22: **if** r == xsd:string or a derivative of xsd:string **then**
23: candidate_value $= c_r$
24: **else**
25: candidate_value $= c_l$
26: **end if**
27: **end if**
28: **end if**
29: add candidate_value to the list of candidate values of dp
30: **end for**

away. If a keyword and an appropriate value are part of the same token in the input text, the distance is 0 and the evidence would be 1. The token '5.0-megapixel' is an example for this case, as the appropriate value '5.0' and the keyword 'megapixel' are part of the same token. If a value with an evidence of 1 has been found, the algorithm would not look any further and will assign the value to the property in the ontology to which the keyword belonged.

If the same data type value appears more than one as a candidate for a certain property, the level of evidence of this value is being changed to the maximum between them:

$$evidence = max(evidence_{old}, evidence_{new}) \tag{6.2}$$

After having identified all candidate values for a certain property in the ontology, we have to choose the final result from these. We already mentioned that OWL allows the definition of functional properties (i.e., `owl:FunctionalProperty`) that is, properties that can have only one value. For these kinds of properties our method chooses the candidate value with the highest computed evidence. For other properties it presents all candidate values whose evidence are above a user defined threshold.

For our sample input text in Figure 6.5 we would have the following attribute-value pairs and their candidate values with decreasing order of their computed evidence.

<Model Name, xsd:Name>

Candidates: D-595 Zoom, Semi-Transmissive TFT, TruePic TURBO

Final Decision: D-595 Zoom

<Megapixel, xsd:float>

Candidates: 5.0, 1.8

Final Decision: 5.0

<Display size, xsd:float>

Candidates: 1.8, 0.8

Final Decision: 1.8

<Optical Zoom, xsd:decimal>

Candidates: 3, 19, 4

Final Decision: 3

<Digital Zoom, xsd:decimal>

Candidates: 4, 12

Final Decision: 4

6.4.3 Change Detection within ontoX

In the previous section we described our method to extract appropriate values for our properties in the ontology. But as many application fields and domains have a dynamic nature, it is possible that parts of the ontology will become out-of-date after a certain amount of time. We think that it is essential to provide ontology-driven systems with means, which are able to detect such changes in the conceptualisation. Therefore, we stated earlier (see Section 5.1.1) that ontologies have to be enriched with several additional components, like as *source_link_components*, *change_components*, and *value_change_frequency*.

In the context of our ontology-driven IESs we do not perform ontology learning, therefore we do not need *source_link_components* that represent links between ontological structures and input files in a data corpus from which they had been extracted. Such link components are useful in settings where the used ontology is being generated (semi-) automatically, because changes in the file corpus, like as the removal of files, can be used to change the ontology in a way that it reflects the information in the updated file corpus.

In many application fields, it may be required to adopt the ontology to changes in the environment by looking at the input files provided by the user. Our aim by developing our extraction method, however, was not to provide sophisticated change management support that covers, for example, undo/redo functionality for changes. We think that such a functionality is not suitable with the nature of IE, because the ontology itself represents besides the conceptualisation of the domain of interest also the task specification. Although, it is possible to change parts of the ontology that do not contribute to the task specification (i.e., ontological constructs whose *relevance* property is stated to be $false$) automatically we think that it is more appropriate to leave this task to the ontology engineer, because the effects of an automatic change in the ontology on the task specification is not foreseeable. There would be always the danger, that the system would tailor the ontology to a certain amount of input data at hand, leading to a bad overall performance of the system for unseen input data.

Therefore, we think that in our setting it would be sufficient to generate a log-file that shows which ontological constructs became out-of-date over time to suggest to the user to change the ontology because it apparently contains constructs that do not occur in the input files anymore.

Our heuristic to detect out-of-date constructs works in a way that incorporates the property *value_change_frequency* and the amount of time over which a property in the ontology did not occur in the input files. There are two values that the property

value_change_frequency can have: *frequent* and *stable*. Properties that are marked as *frequent* are properties that may or may not occur in input files. It would be incorrect to suggest that such a property became out-of-date because it did not appear in the input files over a short amount of time, because the ontology engineer told the system in advance that such a thing could happen.

However, if a property is marked as *stable* it conveys the information that values of this property will most likely appear in input files consequently. So it would be correct to suggest that such a property became out-of-date if it did not appear in input files over a short amount of time. If an ontological construct is not given a value for its' *value_change_frequency* property, our system assumes per default that it is *stable*.

The first idea was to decrease the value that indicates the probability that a certain construct in the ontology is still accurate (property *accurate*) for let's say 20% every time a construct did not appear in the input data. It is clear that such a course of action would not lead us to satisfactory results, because it does not encounter the predicted value change frequency of the construct and also not the confidence level of the construct, which indicated the level of confidence the ontology had for this construct at the first place. Therefore, we adjusted our idea and defined our function to determine the value of accuracy as follows:

$$accuracy_{new} = accuracy_{old} - \frac{(accuracy_{old}/5) * value\_change\_frequency}{confidence\_level} \quad (6.3)$$

This function ensures that the value of accuracy decreases at a faster pace for constructs that are said to be 'stable' than for constructs that are predicted to be 'frequent' anyway. The function also ensures that the value decreases at a slower pace for constructs of which the ontology engineer was more certain.

Having the accuracy computed for each construct after every extraction, the system generates a log file where it lists the constructs in the ontology according to their accuracy value in increasing order, so that the ontology engineer may take a look and decide whether to change the ontology or not.

6.5 Limitations

Our focus in this work was on providing an extraction method that can extract information from natural language text without using any knowledge resource but an input ontology. We thought that such a system would be useful for people with light-weight extraction demands and who are not familiar with generating all kinds of knowledge resources (e.g., gazetteer lists, extraction rules, etc.) or do not have access to linguistic processing resources (e.g., part-of-speech tagger, etc.) some other state-of-the-art IESs require to perform feasible IE.

Because, of these decision at the beginning of our work, we have to live with some limitations, which can be overcome if several requirements on the input ontologies are reconciled.

- The provided keywords have to be chosen carefully, because they are the trigger words using which our method locates candidate values. If the user provides the properties in the ontology with inappropriate keywords, the system can not extract correct values.

- The ontology should contain as much information as possible regarding the constraints on property values that could decrease their possible value space. For example, if it is known that values of a particular property can have only values greater than 1, then its datatype should be `xsd:positiveInteger` instead of just `xsd:integer` or other data types with a wider value space.

So we can say, that the performance of our proposed method highly depends on the quality of the input ontology. If the user can provide the system with an ontology that reconciles the requirements stated above, the results of the system will be as outlined in the next section of our experimental results.

Chapter 7

Experimental Results

You can't bargain with the truth
'Cause whether you're right or you're wrong
We're going to know what you've done
We're going to see where you belong -
in the end
...
And good's going high,
And evil's going down - in the end

In The End - Yusuf Islam

In this chapter, the evaluation results of our proposed method to extract information from natural language text using ontologies formulated in OWL 1.0 (Web Ontology Language) are presented. Fortunately, the IE community can look back on a relatively long history of evaluation research (see Section 4.1), which resulted in a variety of evaluation measures that became a standard over the years, such as precision and recall (see Section 4.4). So, we will use these standards and will present the extraction results of the developed system in terms of its recall and precision values.

We stated earlier (see Section 5.2) that the use of ontologies in conjunction with IESs will increase not only the performance of the systems, but also their scalability and portability. To prove that the system is resistant to changes in the task specification (i.e., the ontology), we change the ontology we used for the first part of our evaluation to extract also additional information and present the results.

Another aspect of our proposed method is that it suggests that constructs are either not relevant anymore or that they are not provided with appropriate keywords, when they do not occur in the input data for a certain amount of time. These suggestions are then presented to the user and she can decide to perform changes on the ontology or not. In this chapter we also state some results we encountered during the evaluation of performance regarding our change detection functionality.

7.1 Evaluation of Performance

In the first part of our conducted evaluation we will focus on the performance of our proposed extraction method that utilises only a relatively small ontology to extract relevant facts from natural language text documents.

We have chosen the domain of digital cameras, because of its popularity nowadays and the fact that its nature can be captured using ontologies. We collected a set of 137 digital camera reviews from the Web[1] in natural language text, with over 57,000 words. The ontology that we have generated for our evaluation and which represents our task specification is depicted in Figure 7.1[2] and states that we are interested in the model name, the number of megapixel, the optical and digital zoom factor, and the display size of a digital camera.

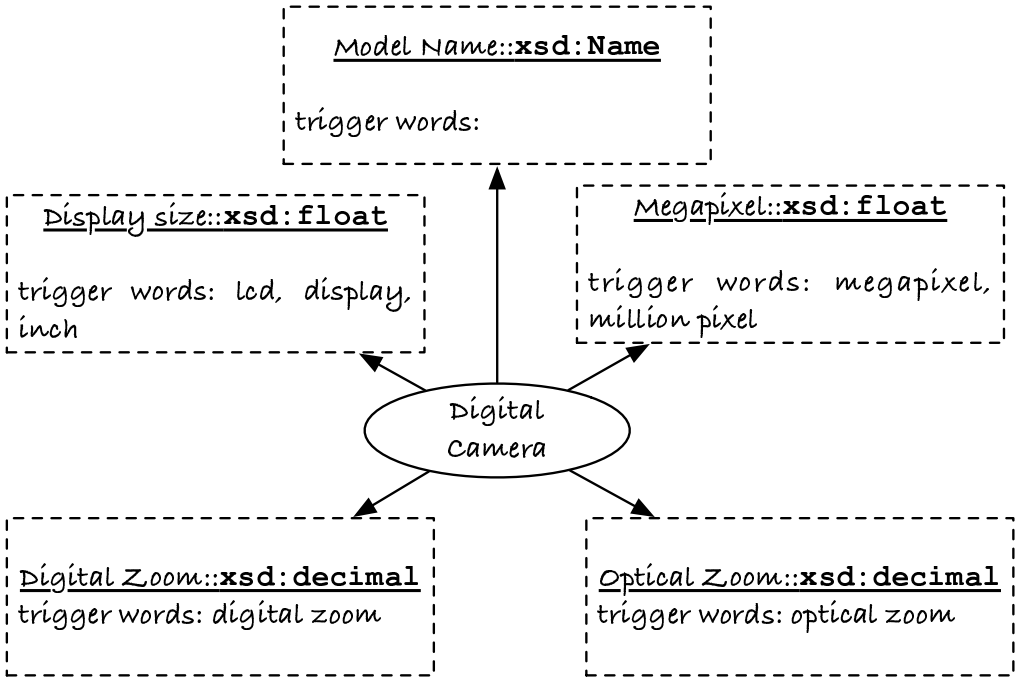

Figure 7.1: Graphical representation of our digital camera ontology

Our aim with this evaluation is to see whether it is sufficient for an IES to know a few keywords and the correct data type of a property to be able to extract appropriate values for them from natural language text documents. Therefore, our ontology contains two properties with the data type `xsd:float` (i.e., 'megapixel' and 'display size'), two properties with the data type `xsd:decimal`, and finally one property with the datatype `xsd:Name`. Whereas, `xsd:float` and `xsd:decimal` can have only predefined values that lie in the value space of their definitions, things are different for `xsd:string` derived data types as `xsd:Name`. Strings can have any value, because they are defined as sequels of characters. In our extraction method

[1] http://www.steves-digicams.com/

[2] The corresponding OWL 1.0 document for this ontology can be found in Appendix A.

we used a heuristic to determine a value that might indicate how likely a string is actually a named entity.

Another aspect we want to evaluate with this first phase, is the usage of ontological relations for the extraction process. For that reason, we did not provide the property 'Model' with any keywords that would indicate the presence of an appropriate value in the text. In such a case, our method had to look in the neighbourhood of occurrences of other properties' keywords for appropriate values.

Table 7.1 depicts the results of our extraction method for our collected data corpus and the ontology as in Figure 7.2 in terms of the standard evaluation metrics Recall and Precision:

$$Recall = \frac{C}{N} \tag{7.1}$$

$$Precision = \frac{C}{C + I} \tag{7.2}$$

, whereas *N* is the number of how many times a property was actually present in the document, *C* is the number of correctly extracted values, and *I* is the number of incorrectly extracted values.

	Number of Facts	Correctly Identified Facts	Incorrectly Identified Facts	Recall	Precision
Model	137	110	28	0,79	0,79
Megapixel	137	70	63	0,51	0,52
Optical zoom	124	105	22	0,84	0,82
Digital zoom	13	6	6	0,46	0,46
Display size	113	93	23	0,82	0,80

Table 7.1: Evaluation results for the digital camera ontology in Figure 7.1

Analysing the results, we must admit that we had expect better results for the property 'Megapixel' because the keywords for that property are relatively clear and occur consistently in input documents and its value space is also narrow. However, we figured that the cause for these results was that some reviews contained the megapixel information as decimals and sometimes as floats (e.g., '5-megapixel' vs. '5.0-megapixel') and sometimes even in letters (e.g., 'five'). Therefore, we think that it is essential for the generation of extraction ontologies to have more means to define data types of properties.

On the other hand, the results for the property 'Model' were a pleasant surprise. We did not expect our method to locate appropriate values for this property that well, which was not even provided with any keywords. The reason for this, must be that the model name of a digital camera appears relatively often in the text and therefore falls more often in the neighbourhood of other keywords occurrences.

The number of incorrectly assigned values for the 'Digital Zoom' property is due to data type occurrences that could not be assigned to their original properties.

For example, often when the megapixel of a camera was given as '5', it was not assigned to the property 'Megapixel' but to the property 'Digital Zoom' if its keyword was near this value occurrence, because the data type of 'Digital Zoom' is `owl:decimal`. Again, better means to state data types can decrease the number of incorrect values, leading to better precision results.

We stated earlier that the use of ontologies in conjunction with IESs can increase not only the performance of IESs, but also their scalability and portability. To evaluate this statement we changed our ontology in Figure 7.2 to represent the task specification where we are interested in three more properties of the same concept.

7.2 Evaluation of Scalability and Portability

To evaluate how our method reacts to changes in the task specification, we changed our task specification that we used for the evaluation of the systems' performance in the previous section. The new specification states, that the system has to extract also the type of storage medium a digital camera supports (e.g., xD-Picture Card, etc.), the kind of power supply it has (e.g., Lithium Ion Battery, etc.), and also information about supported video formats (e.g., MPEG format, etc.). The ontology that covers this specification is depicted in Figure 7.2 [3].

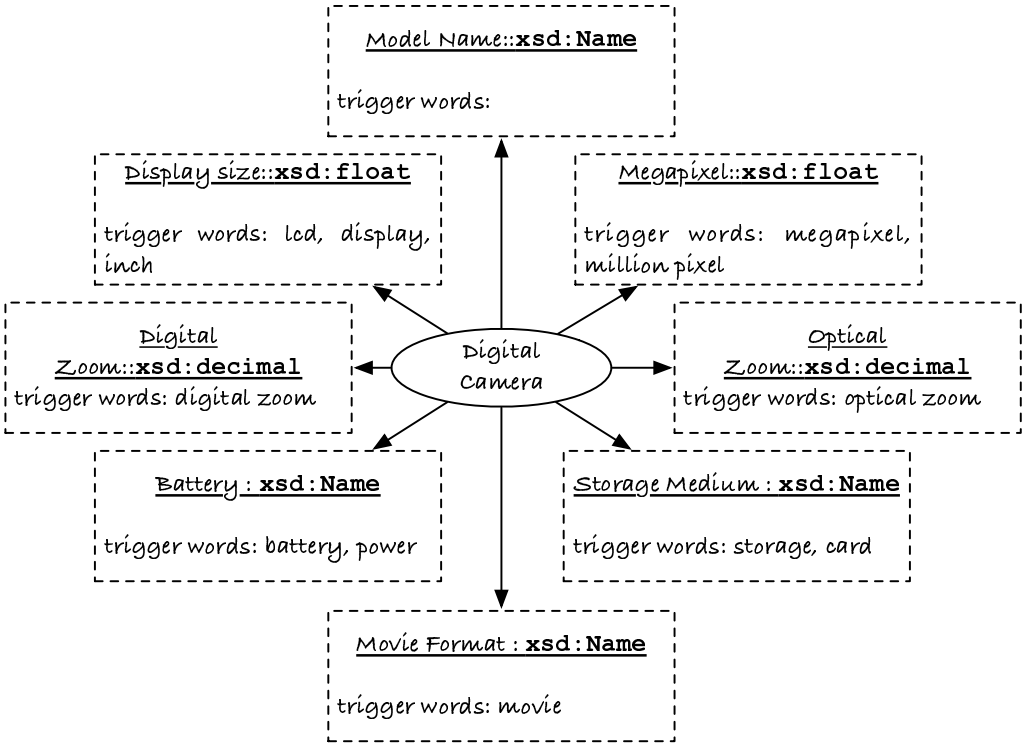

Figure 7.2: Graphical representation of our extended digital camera ontology

[3] The corresponding OWL 1.0 document for this ontology can be found in Appendix A.

Table 7.2 contains the results of our extraction method for the same data corpus and the changed ontology as in Figure 7.3 in terms of the standard evaluation metrics Recall and Precision:

	Number of Facts	Correctly Identified Facts	Incorrectly Identified Facts	Recall	Precision
Model	137	110	28	0,79	0,79
Megapixel	137	70	63	0,51	0,52
Optical zoom	124	105	22	0,84	0,82
Digital zoom	13	6	6	0,46	0,46
Display size	113	93	23	0,82	0,80
Storage	61	15	56	0,25	0,22
Movie Format	56	41	59	0,73	0,41
Power Source	60	26	64	0,43	0,28

Table 7.2: Evaluation results for the extended digital camera ontology in Figure 7.2

As you can see from the first five rows in Table 7.2 that the values for the first five properties mainly have not changed during this second part of our evaluation. We think that this would have been different, if the properties with which we extended our first ontology would have contained properties with the same data types as the already existing ones. In such a case, the number of incorrectly assigned values could have decreased, because our method would have to face less confusion regarding all the appropriate values at hand.

The relatively bad results for the property 'Storage' shows how the choice of inappropriate keywords for properties can affect the performance of the system. On the other hand, we had not that much options to choose from. We figured that the keyword 'storage' is used apparently in another context as well, leading to a lot of incorrectly identified values, which was fostered further by the rather large value space that the data type `xsd:Name` was allowing. The same can be said for the large number of incorrectly assigned values for the properties 'Movie Format' and 'Power Source'. The fact that all of them allow values from the `xsd:Name` value space, lead to a lot of candidate values, with some of them apparently very close to the keywords.

However, the aim with this phase in our evaluation was to look how difficult it is to make the system extract information for a different task specification. As such a change only requires the change of the input ontology and not the IES itself, we conclude that the scalability and portability of ontology-driven IESs are indeed much better than of regular IESs.

As a concluding remark regarding the extraction performance of our method, we can say that the performance highly depends on the quality of the input ontology. With quality we mean the right choice of keywords for the properties and the right choice of data type value. Obviously it is helpful to know a little bit about how relevant properties appear in the input text in order to generate an ontology that represents the right level of generality.

To make it clear how changes in the ontology can affect the resulted output, we changed our input ontology as in Figure 7.3 where we stated that a movie format can be one of 'Quicktime', 'AVI', or 'MPEG' by defining 'MovieFormat' as a class and the three mentioned movie formats as its instances. Further, we stated that storage medium can be one of 'SD card', 'xD card', or 'MemoryStick' by defining 'StorageMedium' as a class and the three mentioned memory cards as its instances. Even with these minor changes we can see a significant improvement in the performance of our IESs for these properties as can be seen in Table 7.3.

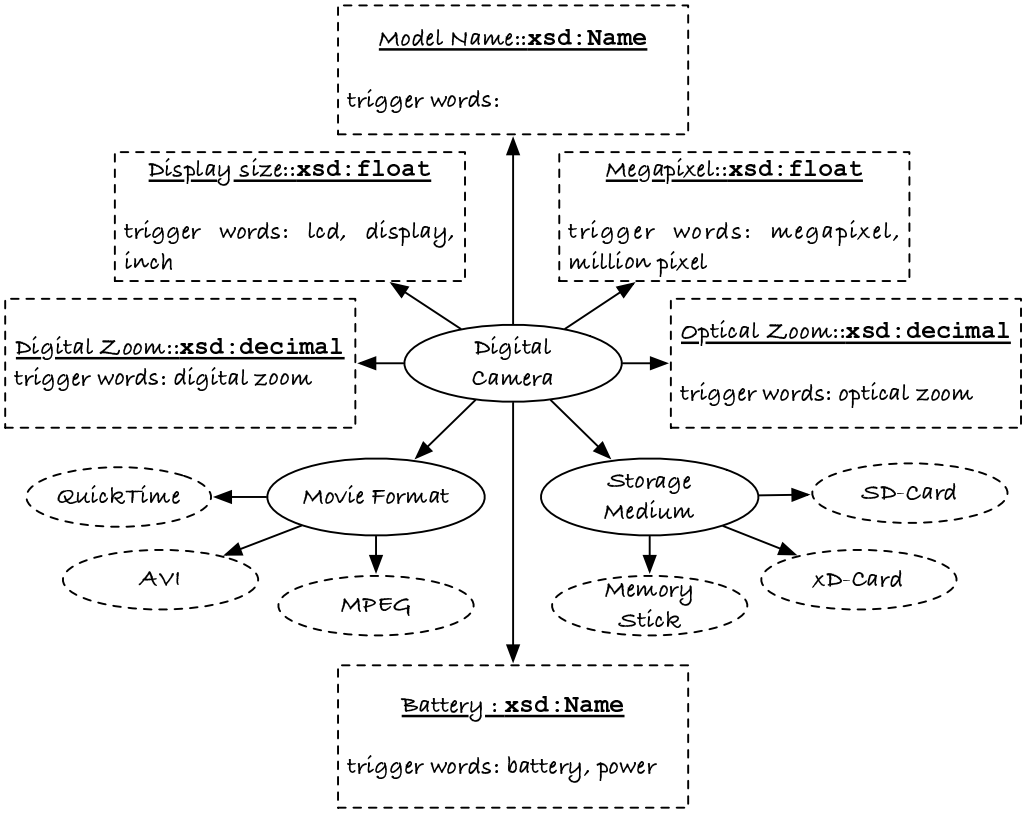

Figure 7.3: Graphical representation of our digital camera ontology

	Number of Facts	Correctly Identified Facts	Incorrectly Identified Facts	Recall	Precision
Storage	61	51	6	0,83	0,89
Movie Format	56	56	0	1	1
Power Source	60	26	64	0,43	0,28

Table 7.3: Evaluation results for the corrected digital camera ontology in Figure 7.3

7.3 Evaluation of Change Detection

Another aspect of our method was the ability of basic change detection in the extraction interests of the user. We stated the heuristic our method applies to determine the probability of accuracy of properties to identify properties or concepts that do not appear in the input text over a certain amount of time. After every extraction, the user will be given a list of all ontological constructs in increasing order of their accuracy level, so that she can change the ontology if necessary.

Low accuracy levels can mean either that the interests of the user has changed with regard to the relevant data that she wants to be extracted from the text, or that the ontological construct is not appropriately defined in the ontology, so that the extraction method cannot make use of it to identify the intended kind of information in the text. One example for the latter case could be that a property is provided with keywords that are not appropriate to locate relevant values for it.

During the course of our evaluation phase we looked at the accuracy levels of our ontological constructs at certain points of time. As can be seen in Table 7.1 and Table 7.2, values for the 'Digital zoom' property appear rather infrequently in the input texts. So we assume that this property will be at the top of the level at most of the times we look at the suggestions of our method.

Normally the accuracy of an ontological construct is computed using its old accuracy, its value change frequency, and its confidence level. The value change frequency and the confidence level are both properties of an ontological construct that have to be stated by the ontology engineer during the generation process of the extraction ontology. For the case where the ontology engineer does not state values for these properties, default values are used instead. The default value change frequency is 'stable', assuming that the construct really represents relevant information and therefore will be present in the input texts. The default value for the confidence level is 1, assuming that the ontology engineer is sure about the presence of this construct in the ontology. These default values are being used for our extraction ontology as well, because we choose not to state values for those properties of our ontological constructs.

Starting to use an extraction ontology for the first time, the constructs in the ontology are being marked with an accuracy of 1. After that, each extraction will initiate the re-computation of this value, using the before mentioned properties. Every time a property does not occur in the input text, its accuracy will decrease with a pace related to its value change frequency and confidence level.

After the first ten input texts had been extracted, our method returned the following accuracy list:

$0,129$ accuracy of 'Digital zoom'

$0,36$ accuracy of 'Optical zoom'

$0,6$ accuracy of 'Megapixel'

1 accuracy of 'Model'

1 accuracy of 'Display size'

Looking at such a list, the user can see that the 'Digital Zoom' property is not well represented in the input texts, indicating either that the property lost its relevance over time (i.e., the domain of interest had changed over time) or that the keywords or the data type of this property are not suitable to extract the desired information. Both cases require the user to look at the input texts and the ontology to figure out which one of these is actually true.

We think that this approach to change detection is necessary and sufficient for the use within IESs. More sophisticated approaches, for example to change the ontology automatically, is not an issue within IESs. This is mainly because the input ontology represents the interests of the user, and changing it would be like imposing the systems' views on the user. Such automatic adaptations would also yield to an ontology that is tailored just for the input texts at hand, whereas it could have been already at the correct level of generality to extract as much information as possible and a change would only worsen the overall performance of the system.

Chapter 8

Summary and Future Work

Maybe there's a world that I'm still to find
Open up o world and let me in, then there'll be
a new life to begin

Maybe There's A World - Yusuf Islam

In this work, we presented our approach towards ontology-driven IES that takes an ontology as input and utilises its pre-defined semantics to exploit as much information as possible of the underlying conceptualisation of the ontology. Using this information it is able to extract information from natural language text.

The system as it is, can be seen as a scalable and portable IES, because to adopt the system to a changed specification or a new domain, only the ontology has to be changed. This task is easier for an ontology-driven approach then for the knowledge engineering approach or the semi-automatic training approach to IE. Because in the case of the former, a human knowledge engineer has to adopt the extraction rules given in a rule representation language that is known to him. And in the case of the latter, a human annotator has to annotate the whole data corpus from scratch or has to go through all the documents in the corpus to change her annotations.

Some domains require better performing IESs and therefore it is likely that some IES developers will tailor their systems for only a particular domain at hand. But although this would cause a decrease in the portability of the system, it still would be easier to scale, compared to the other approaches to IE. Further, many domains have similar characteristics. In such cases the portability should also be not a real problem (e.g., digital cameras vs. digital video cameras).

It should also be stated that generating ontologies today is much more easier then generating extraction rules or making annotations on large data corpora. Whereas, the generation of extraction rules requires a knowledge engineer who is familiar with the particular rule representation language, ontologies can be generated using ontology editing tools which require no knowledge about the underlying syntax of the ontology representation languages. As such, ontologies can be generated by a larger community of people, widening the application field of ontology-driven IESs.

8.1 Summary

In the introduction of this thesis we stated our research questions to be addressed in the course of our work. In this section we will summarise what we were able to discover in our quest for satisfying answers.

Research questions regarding effects of ontologies on IESs

How can ontologies be utilised to address main challenges of IESs, such as performance, portability, and scalability?

Ontologies can be used to represent the context in which the relevant kind of information is naturally embedded in. So, they can serve as both, a specification of relevant information the system has to look for and a conceptualisation of the domain of interest. By providing the IES with such context knowledge, the system is able to rule out incorrect answers and thus increase its performance. Further, by putting the specification and domain knowledge in an ontology, which can be seen as an external and independent component from the system, it is achieved that application-specific knowledge became explicit. Changes in the specification require only changes in the ontology and not the system itself, yielding to better scalable IESs. The same fact can also yield better portability, because to provide extraction for a different domain, the ontology can be changed so that it represents the new specification and the new domain.

Is it possible to develop an unsupervised and automatic IE method, that utilises no other resource but an input ontology?

Given the well defined semantics of existing ontology representation languages it is indeed possible to utilise ontologies to develop automatic IESs. Our experimental results (see Chapter 7) showed that an IES can yield feasible results even if it uses only an input ontology and no other knowledge resource that ordinary IESs use, such as lexicons, parsers, etc. The only thing that has to be provided to the system is apparently the input ontology. When this is generated by a human, it will be more likely at the right level of generality. However, this cannot be compared to other approaches to IESs where the human intervention takes the form of generating extraction rules using a particular rule representation language, or of annotating large amounts of data. Rather, the user has to develop a small ontology representing her interests with respect to a certain domain and can thereby use well-known ontology development frameworks. In that way, the application field of our method is widened, because it enables the 'common' user to state extraction specifications that goes beyond simple keyword search.

Are there certain requirements to the content of ontologies when they are going to be used in conjunction with IESs? If yes, what are they and how can they be reconciled?

In the course of our work we encountered that for many application fields, researchers propose the use of additional knowledge that help the system to perform its task more accurately. The field of IE turns out to be one of these application fields, because of its complexity and because of the complexity of the ontology lifecycle itself. In our case we had to provide the system, first of all, with trigger words (i.e., keywords) for the sought after components in the ontology. The second important thing, was to specify value constraints so that the system could narrow its search range from almost everything to, for example, integer numbers.

We also encountered that it can be very useful to mark ontological components with confidence levels. For example, when the same value is assignable to two components at the same time, the component will be chosen of which the ontology engineer was more certain. The system uses these confidence levels also to predict whether the extraction interests of the user has changed over time or not. Our system makes use of a method that decreases the accuracy of an ontological component when it does not appear in the input files according to a mathematical function that uses the pre-defined confidence level and information about the components' predicted behaviour over time.

Research questions regarding the maintenance of ontologies within IESs

How to detect out-of-date ontological components in a domain of interest?

As mentioned before, the detection of out-of-date ontological components can take several forms depending on the type of the ISs at hand. In our case, where we have an IES, the most straight forward way to detect that an ontological component is not relevant anymore is to monitor its appearance in the input files over time. However, during our work we encountered that pre-defined knowledge about the predicted behaviour of a component can be very helpful to decide whether a change in the appearance frequency is really an indicator that the component became out-of-date or not. Therefore, we suggest that the user of our system should mark the components in the ontology as 'stable' or 'frequent' indicating that the component will likely occur in the input files consequently (i.e., stable) or that its appearance in the input files are rather unpredictable (i.e., frequent). Considering this value change frequency and the confidence level of an ontological construct, our system computes the *accuracy* of each component after a extraction is being performed on an input text. If the *accuracy* of a component reaches a given threshold the system can suggest that it became out-of-date.

How to query/monitor the changes themselves?

As mentioned in Section 3.3.2 there are several ways to represent ontology changes. In our context, we decided that it is not desirable to change the ontology automatically. Rather, we present the user suggestions of changes that could be performed on the ontology, because either the conceptualisation have changed or the extraction interest of the user has changed.

How to apply changes to the ontology? Automatically or manually?

In cases where the ontology is being generated using a corpus of domain relevant documents, changes can be detected by monitoring changes in the corpus itself and applied automatically. In our case, however, we use already existing ontologies to extract information form input documents and do not perform any automatic ontology learning. So, our only source that represents the domain of interest are the input files the user provides us to extract information from. One might think that the ontology could be changed when certain components are not present in the input files over a certain time anymore. However, we think that this is not that good an idea in the context of IESs, because it would adopt the ontology to the input files causing over-fitting. Therefore, we decided that the ontology, mainly because it represents the task specification, should not be changed by the system, but only by the user.

The only changes in the ontology performed automatically by our IESs is to compute new values for properties such as *accuracy*. These changes only affect the displayed results of the extraction process and do not change the conceptualisation represented by the ontology. The system then lists the components of the ontology by their accuracy in increasing order and suggests that components with a low accuracy should be changed in the ontology. However, the end decision is going to be made by the user of the system.

8.2 Future Work

In this section, we want to point out directions for future work in the field of ontology-based IE that resulted from the research we have done in the course of this thesis, because we think that a lot of work has to be done in this field to convince users that it is indeed beneficial to use ontologies in conjunction with IESs.

Developing an extension to OWL for IE

In this thesis we proposed several properties that should be used to enrich the existing components of OWL to represent conceptualisations. However, we did not developed an extension to OWL that covered these properties. The reason for that was mainly because this work aims to give insights in the usability and benefits of ontologies when used in conjunction with IESs. To develop an extension for a particular ontology representation language would therefore be out of the scope of our work.

However, we think that the more OWL becomes the quasi standard for representing ontologies among the AI community, such an extension could be useful for researchers interested in IE and thus worth the effort. The difficulty of such an attempt should not be underestimated, because the development of an extension would require the development of appropriate reasoners that can handle the new components. Further, it would also require the extension of existing APIs or the development of new APIs to make the integration of ontologies in IESs possible.

An extension to OWL for IE should contain means for the following:

- User-defined data types
- Quality properties
- Temporal properties
- Linguistic properties

Incorporating linguistics

Another possible direction for future work would be the incorporation of linguistics into the extraction process. For that, the ontological components could have to be enriched with linguistic information, for example the part-of-speech tag for possible values of a certain property. Of course, the IESs would have to be developed in a way that it can process this kind of knowledge. Further, it would be necessary to pre-process the input files of the system linguistically, to at least assign part-of-speech tags to the words in the input files.

Such an attempt would especially be useful for domains where the data types of the relevant information are mainly strings.

Utilising intentional knowledge for better extraction results

In our proposed method we focused only on the extensional knowledge present in ontologies but merely neglected the intensional knowledge (i.e., instances) if they were not directly related to properties of interest (i.e., `owl:ObjectProperty`). However, intentional knowledge could turn out to be useful as well. An IES could compare its identified candidate values with existing values of instances and could make decisions based on some similarity measurement.

Appendix A

Example Ontologies in OWL

OWL Representation of the Basic Digital Camera Ontology

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   ]>

<rdf:RDF
  xmlns     ="&owl;"
  xmlns:owl ="&owl;"
  xml:base  ="http://www.w3.org/2002/07/owl"
  xmlns:rdf ="&rdf;"
  xmlns:rdfs="&rdfs;"
>


<owl:Ontology rdf:about="">
  <rdfs:comment> An ontology representing my interests with regard to
    digital cameras.
  </rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID = "digital_camera">
  <rdfs:comment> digital camera </rdfs:comment>
</owl:Class>

<owl:DatatypeProperty rdf:ID="model">
  <rdfs:comment></rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;Name" />
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="display">
  <rdfs:comment>lcd, LCD, display, screen, inch, liquid, crystal
  </rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;float" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="megapixel">
  <rdfs:comment>megapixel, Megapixel, MegaPixel million pixel</rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;float" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="optical_zoom">
  <rdfs:comment>optical zoom, Optical, optical, zoom</rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;decimal" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="digital_zoom">
  <rdfs:comment>digital zoom</rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;decimal" />
</owl:DatatypeProperty>

</rdf:RDF>
```

OWL Representation of the Extended Digital Camera Ontology

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   ]>

<rdf:RDF
  xmlns     ="&owl;"
  xmlns:owl ="&owl;"
  xml:base  ="http://www.w3.org/2002/07/owl"
  xmlns:rdf ="&rdf;"
  xmlns:rdfs="&rdfs;"
>


<owl:Ontology rdf:about="">
  <rdfs:comment> An ontology representing my interests with regard to
     digital cameras.
  </rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID = "digital_camera">
  <rdfs:comment> digital camera </rdfs:comment>
</owl:Class>

<owl:DatatypeProperty rdf:ID="model">
  <rdfs:comment></rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;Name" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="display">
  <rdfs:comment>lcd, LCD, display, screen, inch, liquid, crystal
  </rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;float" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="megapixel">
  <rdfs:comment>megapixel, Megapixel, MegaPixel, million, pixel
  </rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;float" />
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="optical_zoom">
  <rdfs:comment>optical zoom, optical, Optical, zoom</rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;decimal" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="digital_zoom">
  <rdfs:comment>digital zoom</rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;decimal" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="battery">
  <rdfs:comment>battery, power, Battery</rdfs:comment>
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="&xsd;Name" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="storage_type">
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="#StorageMedium" />
</owl:ObjectProperty>

<owl:Class rdf:ID ="StorageMedium"/>

<StorageMedium rdf:ID ="SD"/>

<StorageMedium rdf:ID ="CF"/>

<StorageMedium rdf:ID ="xD"/>

<StorageMedium rdf:ID ="MemoryStick"/>

<owl:ObjectProperty rdf:ID="movie_format">
  <rdfs:domain rdf:resource = "#digital_camera" />
  <rdfs:range rdf:resource ="#MovieFormat" />
</owl:ObjectProperty>

<owl:Class rdf:ID ="MovieFormat"/>

<MovieFormat rdf:ID ="QuickTime"/>

<MovieFormat rdf:ID ="AVI"/>

<MovieFormat rdf:ID ="MPEG"/>

</rdf:RDF>
```

Appendix B

OWL 1.0 Data Types

Allowed XML Schema data types in OWL 1.0 as defined in XML Schema Part 2: Datatypes[1]

`xsd:string` represents character strings in XML. Set of finite-length sequences of characters.

`xsd:boolean` has the value space required to support the mathematical concept of binary-valued logic: {true, false}.

`xsd:decimal` represents arbitrary precision decimal numbers. The value space of `decimal` is the set of the values i x 10^{-n}, where i and n are integers such that n $>= 0$.

`xsd:float` corresponds to the IEEE single-precision 32-bit floating point type [IEEE 754-1985]. The basic value space of `float` consists of the values m x 2^e where m is an integer whose absolute value is less than 2^{24}, and e is an integer between -149 and 104, inclusive. In addition to the basic value space described above, the value space of `float` also contains the following special values: positive and negative zero, positive and negative infinity and not-a-number. The order-relation on float is: x $<$ y iff y - x is positive. Positive zero is greater than negative zero. Not-a number equals itself and is greater than all float values including positive infinity.

`xsd:double` corresponds to IEEE double-precision 64-bit floating point type [IEEE 754-1985]. The basic value space of `double` consists of the values m x 2^e, where m is an integer whose absolute value is less than 2^{53}, and e is an integer between -1075 and 970, inclusive. In addition to the basic value space described above, the value space of `double` also contains the following special values: positive and negative zero, positive and negative infinity and not-a-number. The order-relation on double is: x $<$ y iff y - x is positive. Positive zero is greater than negative zero. Not-a-number equals itself and is greater than all double values including positive infinity.

[1] http://www.w3.org/TR/2003/WD-owl-semantics-20030203/syntax.html

`xsd:dateTime` represents a specific instant of time. The value space of `dateTime` is the space of combinations of date and time of day values as defined in 5.4 of [ISO 8601][2].

A single lexical representation, which is a subset of the lexical representations allowed by [ISO 8601], is allowed for `dateTime`. This lexical representation is the [ISO 8601] extended format CCYY-MM-DDThh:mm:ss where "CC" represents the century, "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading "-" sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired i.e the format ss.ss... with any number of digits after the decimal point is supported. The fractional seconds part is optional; other parts of the lexical form are not optional. To accommodate year values greater than 9999 additional digits can be added to the left of this representation. Leading zeros are required if the year value would otherwise have fewer than four digits; otherwise they are forbidden. The year 0000 is prohibited.

This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or -, followed by the difference from UTC represented as hh:mm (note: the minutes part is required).

For example, to indicate 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: 1999-05-31T13:20:00-05:00.

`xsd:time` represents an instant of time that recurs every day. The value space of `time` is the space of time of day values as defined in 5.3 of [ISO 8601]. Specifically, it is a set of zero-duration daily time instances.

The lexical representation for `time` is the left truncated lexical representation for dateTime: hh:mm:ss.sss with optional following time zone indicator. For example, to indicate 1:20 pm for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: 13:20:00-05:00.

`xsd:date` represents a calendar date. The value space of `date` is the set of Gregorian calendar dates as defined in 5.2.1 of [ISO 8601]. Specifically, it is a set of one-day long, non-periodic instances e.g. lexical 1999-10-26 to represent the calendar date 1999-10-26, independent of how many hours this day has.

The lexical representation for `date` is the reduced (right truncated) lexical representation for `dateTime`: CCYY-MM-DD. No left truncation is allowed. An optional following time zone qualifier is allowed as for `dateTime`. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed. For example, to indicate May the 31st, 1999, one would write: 1999-05-31.

[2] International Organization for Standardization (ISO), Dates and Times, 1988-06-15.

`xsd:gYearMonth` represents a specific gregorian month in a specific gregorian year. The value space of `gYearMonth` is the set of Gregorian calendar months as defined in 5.2.1 of [ISO 8601]. Specifically, it is a set of one-month long, non-periodic instances e.g. 1999-10 to represent the whole month of 1999-10, independent of how many days this month has.

The lexical representation for `gYearMonth` is the reduced (right truncated) lexical representation for `dateTime`: CCYY-MM. No left truncation is allowed. An optional following time zone qualifier is allowed. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed. For example, to indicate the month of May 1999, one would write: 1999-05.

`xsd:gYear` represents a gregorian calendar year. The value space of `gYear` is the set of Gregorian calendar years as defined in 5.2.1 of [ISO 8601]. Specifically, it is a set of one-year long, non-periodic instances e.g. lexical 1999 to represent the whole year 1999, independent of how many months and days this year has.

The lexical representation for `gYear` is the reduced (right truncated) lexical representation for dateTime: CCYY. No left truncation is allowed. An optional following time zone qualifier is allowed as for `dateTime`. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed. For example, to indicate 1999, one would write: 1999.

`xsd:gMonthDay` is a gregorian date that recurs, specifically a day of the year such as the third of May. Arbitrary recurring dates are not supported by this datatype. The value space of `gMonthDay` is the set of calendar dates, as defined in 3 of [ISO 8601]. Specifically, it is a set of one-day long, annually periodic instances.

The lexical representation for `gMonthDay` is the left truncated lexical representation for date: –MM-DD. An optional following time zone qualifier is allowed as for date. No preceding sign is allowed. No other formats are allowed.

This datatype can be used to represent a specific day in a month. To say, for example, that my birthday occurs on the 14th of September ever year.

`xsd:gDay` is a gregorian day that recurs, specifically a day of the month such as the 5th of the month. Arbitrary recurring days are not supported by this datatype. The value space of `gDay` is the space of a set of calendar dates as defined in 3 of [ISO 8601]. Specifically, it is a set of one-day long, monthly periodic instances.

The lexical representation for `gDay` is the left truncated lexical representation for date: —DD . An optional following time zone qualifier is allowed as for date. No preceding sign is allowed. No other formats are allowed.

`xsd:gMonth` is a gregorian month that recurs every year. The value space of `gMonth` is the space of a set of calendar months as defined in 3 of [ISO 8601]. Specifically, it is a set of one-month long, yearly periodic instances.

The lexical representation for `gMonth` is the left and right truncated lexical rep-

resentation for date: –MM–. An optional following time zone qualifier is allowed as for date. No preceding sign is allowed. No other formats are allowed.

`xsd:hexBinary` represents arbitrary hex-encoded binary data. The value space of `hexBinary` is the set of finite-length sequences of binary octets.

`hexBinary` has a lexical representation where each binary octet is encoded as a character tuple, consisting of two hexadecimal digits ([0-9a-fA-F]) representing the octet code. For example, "0FB7" is a hex encoding for the 16-bit integer 4023 (whose binary representation is 111110110111).

`xsd:base64Binary` represents Base64-encoded arbitrary binary data. The value space of `base64Binary` is the set of finite-length sequences of binary octets.

`xsd:anyURI` represents a Uniform Resource Identifier Reference (URI). An `anyURI` value can be absolute or relative, and may have an optional fragment identifier (i.e., it may be a URI Reference).

`xsd:normalizedString` represents white space normalized strings. The value space of `normalizedString` is the set of strings that do not contain the carriage return (#xD), line feed (#xA) nor tab (#x9) characters. The lexical space of `normalizedString` is the set of strings that do not contain the carriage return (#xD) nor tab (#x9) characters.

`xsd:token` represents tokenized strings. The value space of `token` is the set of strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces. The lexical space of `token` is the set of strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces.

`xsd:language` represents natural language identifiers as defined by [RFC 1766[3]]. The value space of language is the set of all strings that are valid language identifiers as defined in the language identification section of [XML 1.0 (Second Edition)[4]]. The lexical space of `language` is the set of all strings that are valid language identifiers as defined in the language identification section of [XML 1.0 (Second Edition)].

`xsd:NMTOKEN` represents the NMTOKEN attribute type from [XML 1.0 (Second Edition)]. The value space of `NMTOKEN` is the set of tokens that match the Nmtoken production in [XML 1.0 (Second Edition)]. The lexical space of `NMTOKEN` is the set of strings that match the Nmtoken production in [XML 1.0 (Second Edition)].

[3]H. Alvestrand, ed. RFC 1766: Tags for the Identification of Languages 1995.

[4]World Wide Web Consortium. Extensible Markup Language (XML) 1.0, Second Edition.

`xsd:Name` represents XML Names. The value space of `Name` is the set of all strings which match the Name production of [XML 1.0 (Second Edition)]. The lexical space of `Name` is the set of all strings which match the Name production of [XML 1.0 (Second Edition)].

`xsd:NCName` represents XML "non-colonized" Names. The value space of `NCName` is the set of all strings which match the NCName production of [Namespaces in XML[5]]. The lexical space of `NCName` is the set of all strings which match the NCName production of [Namespaces in XML].

`xsd:integer` is derived from `decimal` by fixing the value of fractionDigits to be 0. This results in the standard mathematical concept of the integer numbers. The value space of `integer` is the infinite set ...,-2,-1,0,1,2,....

`integer` has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39) with an optional leading sign. If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

`xsd:nonPositiveInteger` is derived from `integer` by setting the value of maxInclusive to be 0. This results in the standard mathematical concept of the non-positive integers. The value space of `nonPositiveInteger` is the infinite set ...,-2,-1,0.

`nonPositiveInteger` has a lexical representation consisting of a negative sign ("-") followed by a finite-length sequence of decimal digits (#x30-#x39). If the sequence of digits consists of all zeros then the sign is optional. For example: -1, 0, -12678967543233, -100000.

`xsd:negativeInteger` is derived from `nonPositiveInteger` by setting the value of maxInclusive to be -1. This results in the standard mathematical concept of the negative integers. The value space of `negativeInteger` is the infinite set ...,-2,-1.

`negativeInteger` has a lexical representation consisting of a negative sign ("-") followed by a finite-length sequence of decimal digits (#x30-#x39). For example: -1, -12678967543233, -100000.

`xsd:long` is derived from `integer` by setting the value of maxInclusive to be 9223372036854775807 and minInclusive to be -9223372036854775808.

`long` has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

`xsd:int` is derived from `long` by setting the value of maxInclusive to be 2147483647 and minInclusive to be -2147483648.

`int` has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is

[5]World Wide Web Consortium. Namespaces in XML.

assumed. For example: -1, 0, 126789675, +100000.

`xsd:short` is derived from `int` by setting the value of maxInclusive to be 32767 and minInclusive to be -32768.

`short` has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 12678, +10000.

`xsd:byte` is derived from `short` by setting the value of maxInclusive to be 127 and minInclusive to be -128.

`byte` has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 126, +100.

`xsd:nonNegativeInteger` is derived from `integer` by setting the value of minInclusive to be 0. This results in the standard mathematical concept of the non-negative integers. The value space of `nonNegativeInteger` is the infinite set 0,1,2,....

`nonNegativeInteger` has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: 1, 0, 12678967543233, +100000.

`xsd:unsignedLong` is derived from `nonNegativeInteger` by setting the value of maxInclusive to be 18446744073709551615.

`unsignedLong` has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 12678967543233, 100000.

`xsd:unsignedInt` is derived from `unsignedLong` by setting the value of maxInclusive to be 4294967295.

`unsignedInt` has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 1267896754, 100000.

`xsd:unsignedShort` is derived from `unsignedInt` by setting the value of maxInclusive to be 65535.

`unsignedShort` has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 12678, 10000.

`xsd:unsignedByte` is derived from `unsignedShort` by setting the value of maxInclusive to be 255.

`unsignedByte` has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 126, 100.

`xsd:positiveInteger` is derived from `nonNegativeInteger` by setting the value of minInclusive to be 1. This results in the standard mathematical concept of the positive integer numbers. The value space of `positiveInteger` is the infinite set 1,2,....

`positiveInteger` has a lexical representation consisting of an optional positive sign ("+") followed by a finite-length sequence of decimal digits (#x30-#x39). For example: 1, 12678967543233, +100000.

List of Figures

Bibliography

[Aitken, 2002] J.S. Aitken. Learning information extraction rules: An inductive logic programming approach. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, Amsterdam, 2002. IOS Press.

[Appelt and Israel, 1999] D.E. Appelt and D.J. Israel. Introduction to information extraction technology. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

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

[Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[Baehrecke *et al.*, 2004] E. Baehrecke, N. Dang, K. Babaria, and B. Shneidermane. Visualization and analysis of Microarray and Gene Ontology data with Treemaps. *BMC Bioinformatics*, 5(84), 2004.

[Berners-Lee, 1999] T. Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1999.

[Brachman and Levesque, 1984] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proceedings of the National Conference on Artificial Intelligence*, pages 34–37, Austin, Texas, 1984. William Kaufmann.

[Brewster *et al.*, 2004] Ch. Brewster, H. Alani, S. Dasmahapatra, and Y. Wilks. Data-driven ontology evaluation. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, Lisbon, 2004. European Language Resources Association.

[Buitelaar *et al.*, 2005] P. Buitelaar, Ph. Cimiano, and B. Magnini. *Ontology Learning from Text: An Overview*, volume 123 of *Frontiers in Artificial Intelligence and Applications*, pages 3–12. IOS Press, 2005.

[Chincor, 1998] N.A. Chincor. Overview of MUC-7/MET-2. In *In Proceedings of the Message Understanding Conference MUC-7*, 1998.

[Cimiano and J.Völker, 2005] Ph. Cimiano and J.Völker. Text2Onto - a framework for ontology learning and data-driven change discovery. In *In Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005)*, 2005.

[Cowie and Lehnert, 1996] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39:1:80–91, 1996.

[Donini *et al.*, 1996] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In *Principles of Knowledge Representation*, pages 191–236. Center for the Study of Language and Information, Stanford, CA, US, 1996.

[Embley *et al.*, 1992] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented systems analysis: a model-driven approac*. Prentice-Hall, 1992.

[Embley, 2004] D.W. Embley. Toward semantic understanding: An approach based on information extraction ontologies. In *Proceedings of the Fifteenth Conference on Australasian Database*, pages 3–12. Australian Computer Society, January 2004.

[Faure and Nedellec, 1999] D. Faure and C. Nedellec. Knowledge acquisition of predicate argument structures from technical texts using machine learning: The system ASIUM. In *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management*, 1999.

[Fensel *et al.*, 2001] D. Fensel, I. Horrocks, F. Harmelen, D. McGuinness, and D. Patel-Schneider. OIL: Ontology infrastructure to enable the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

[Fluit *et al.*, 2004] C. Fluit, M. Sabou, and F. van Harmelen. Supporting user tasks through visualisation of light-weight ontologies. In *Handbook on Ontologies*. Springer Verlag, 2004.

[Fruchterman and Reingold, 1991] T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Software - Practice Experience*, 21(11):1129–1164, 1991.

[Goméz-Pérez, 1995] A. Goméz-Pérez. Some ideas and examples to evaluate ontologies. In *Proceedings of the 11th Conference on Artificial Intelligence for Applications*, February 1995.

[Grigoris and van Harmelen, 2004] A. Grigoris and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts, London, England, 2004.

[Grishman and Sundheim, 1996] R. Grishman and B. Sundheim. Message understanding conference - 6: A brief history. In *In Proceedings of the International Conference on Computational Linguistics*, 1996.

[Grishman and Yangarber, 2000] R. Grishman and R. Yangarber. Issues in corpus-trained information extraction. In *In Proceedings of International Symposium: Toward the Realization of Spontaneous Speech Engineering*, pages 107–112, February 2000.

[Gruber, 1993] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, 1993.

[Guarino and Giaretta, 1995] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In Mars N, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32. IOS Press, Amsterdam, The Netherlands, 1995.

[Guarino, 1998] N. Guarino. Formal ontology and information systems. In Nicola Guarino, editor, *Proceedings of the First International Conference on Formal Ontologies in Information Systems (FOIS)*, pages 3–15, June 1998.

[Hearst, 1998] M. Hearst. Automated discovery of WordNet relations. In *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[Heflin and Hendler, 2000] J. Heflin and J.A. Hendler. Dynamic ontologies on the web. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[Horrocks *et al.*, 2003] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[Kifer *et al.*, 1995] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[Klein and Noy, 2003] M. Klein and N. Noy. A component-based framework for ontology evolution. Technical report, Department of Computer Science, Vrije Universiteit Amsterdam, 2003.

[Klein, 2002] M. Klein. Versioning of distributed ontologies. Technical report, Vrije Universiteit Amsterdam, 2002.

[Klein, 2004] M. Klein. *Change Management for Distributed Ontologies*. PhD thesis, Department of Computer Science, Vrije Universiteit Amsterdam, 2004.

[Lavelli *et al.*, 2004] A. Lavelli, M. Califf, F. Ciravegna, D. Freitag, C. Giuliano, N. Kushmerick, and L. Romano. IE evaluation: Criticisms and recommendations. In *Proceedings of the AAAI 2004 Workshop on Adaptive Text Extraction and Mining (ATEM 2004)*, 2004.

[Maedche, 2002] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, Boston, Dordrecht, London, 2002.

[Marsh and Perzanowski, 1998] E. Marsh and D. Perzanowski. MUC-7 evaluation of IE technology: Overview of results. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, April 1998.

[Menzies, 1999] T. Menzies. Cost benefits of ontologies. *Intelligence*, 10:26–32, 1999.

[Minsky, 1975] M. Minsky. A framework for representing knowledge. In *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, 1975.

[Mutton and Golbeck, 2003] P. Mutton and J. Golbeck. Visualization of semantic metadata and ontologies. In *Proceedings of the Seventh International Conference on Information Visualization*, pages 300–305, July 2003.

[Noy and Klein, 2004] N. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4):428–440, 2004.

[Noy and Musen, 2002] N. Noy and M. Musen. PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[Riloff, 1999] E. Riloff. Information extraction as a stepping stone toward story understanding. In *Understanding Language Understanding: Computational Models of Reading*. MIT Press, Cambridge, MA, USA, 1999.

[Schutz and Buitelaar, 2005] A. Schutz and P. Buitelaar. RelExt: A tool for relation extraction in ontology extension. In *Proceedings of the 4th International Semantic Web Conference*, 2005.

[Shamsfard and Barforoush, 2003] M. Shamsfard and A.A. Barforoush. The state of the art in ontology learning: a framework for comparison. *The Knowledge Engineering Review*, 18(4):293–316, 2003.

[Shneiderman, 1992] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[Sowa, 1987] J.F. Sowa. *Semantic Networks*. Encyclopedia of Artificial Intelligence. Wiley, 1987.

[Stojanovic and Motik, 2002] L. Stojanovic and B. Motik. Ontology evolution within ontology editors. In *Proceedings of the OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management*, pages 53–62, September 2002.

[Stojanovic *et al.*, 2002] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 285–300, October 2002.

[Tamma and Bench-Capon, 2002] V. Tamma and T. Bench-Capon. An ontology model to facilitate knowledge-sharing in multi-agent systems. *The Knowledge Engineering Review*, 17(1):41–60, 2002.

[Uschold and Grüninger, 1996] M. Uschold and M. Grüninger. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[Uschold and King, 1995] M. Uschold and M. King. Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conduction with IJCAI-95*, August 1995.

[Yildiz, 2004] B. Yildiz. Information extraction - utilizing table patterns. Master's thesis, Vienna University of Technology, 2004.

[Zúniga, 2001] G.L. Zúniga. Ontology: Its transformation from philosophy to information systems. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 187–197, October 2001.