
The University Of Sheffield.

M. Yankova-Doseva

# TERMS:

# Text Extraction from Redundant and Multiple Sources

Submitted for the degree of Doctor of Philosophy

2010

# TERMS:

# Text Extraction from Redundant and Multiple Sources

**Milena Yankova-Doseva**

Submitted for the degree of Doctor of Philosophy

Department of Computer Science
The University of Sheffield

May 2010

# Contents

# List of Figures

# Abstract

In this work we present our approach to the identity resolution problem: discovering references to one and the same object that come from different sources. Solving this problem is important for a number of different communities (e.g. Database, NLP and Semantic Web) that process heterogeneous data where variations of the same objects are referenced in different formats (e.g. textual documents, web pages, database records, ontologies etc.). Identity resolution aims at creating a single view into the data where different facts are interlinked and incompleteness is remedied.

We propose a four-step approach that starts with *schema alignment* of incoming data sources. As a second step - *candidate selection*- we discard those entities that are totally different from those that they are compared to. Next the main evidence for identity of two entities comes from applying *similarity measures* comparing their attribute values. The last step in the identity resolution process is *data fusion* or merging entities found to be identical into a single object.

The principal novel contribution of our solution is the use of a rich semantic knowledge representation that allows for flexible and unified interpretation during the resolution process. Thus we are not restricted in the type of information that can be processed (although we have focussed our work on problems relating to information extracted from text).

We report the implementation of these four steps in an IDentity Resolution Framework (IDRF) and their application to two use-cases. We propose a rule based approach for customisation in each step and introduce logical operators and their interpretation during the process. Our final evaluation shows that this approach facilitates high accuracy in resolving identity.

# Acknowledgements

It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

I owe my deepest gratitude to my supervisor, Prof. Hamish Cunnigham; this thesis would not have been possible without his irrepealable help, support and patience. I would like also to thank my co-supervisors Horacio Saggion and Galia Angelova and my research panel consisting of Dr. Louise Guthrie, Dr. Mark Sanderson and Prof. Phil Green for their constant guidance during my postgraduate research period.

I am indebted to many of my colleagues at Sheffield University as well as at Ontotext AD. and especially to Kalina Boncheva, Borislav Popov and Krasimir Angelov for their feedback in the initial discussions on implementation and application of the result of my research.

I would like to thank the management of Ontotext AD for their encouragement and financial support. This work was also partially supported by the EU-funded projects MUSING (IST-2004-027097), MediaCampaign (027413) and NoTube (FP7-ICT-231761).

This thesis is a small tribute to my noble grandfather who has always motivated me with his example and has given me the confidence that education makes a difference.

# Chapter 1

# Introduction

## 1.1 Motivation

In this work we focus on extracting information from redundant and multiple sources. By redundancy we mean the fact that the same information or different versions of one and the same information can be obtained from various independent sources. It can be authored by different individuals or organisations e.g. different news agencies reporting one and the same event, or originated by a single source and distributed over different media (e.g. internet pages, personal local copies, etc.).

Regardless of their origin, there are several characteristics of a redundant dataset that increase the complexity of further data processing:

- *Enlarged volume* - the redundant data set contains items than can be removed without losing any information. Extending the definition above, a redundant fact is a bit of information that is presented more than once in a dataset. Thus removing a duplicated fact will decrease the size of the dataset without loss because the same information will still be presented by another fact in the resulting dataset. For example several portals report on currency exchange rates. When all redundant facts are removed the resulting dataset will represent the same information as the redundant one, whereas the size of the original set will be larger.

- *Data errors and variations* - if the dataset is collected from different sources one can expect contradictions in matters of detail if not substance. Some sources may report different details because of an error introduced during data entry, or the details change over time and more recent information becomes available. For example number of employees of a company may vary between the previous annual report and the present one. Another possibility is that different sources express different perspectives on the same facts and so differ in details, e.g. the expected benefit from applying a social care reform as estimated by different organisations. Cleaning errors and out-dated information in a redundant dataset may significantly increase the quality of further data analysis.

- *Information spread over different sources* - duplicated facts may not be totally overlapping (i.e. redundant) but possess some unique details that are not present in other sources, i.e. part of the information is given by one source while another part is provided by another source. For example a company profile provided by a particular source may contain details about company share prices while another may focus on its recruitment behaviour. Consolidating different elements in a single description and removing duplicated information may significantly reduce the complexity of subsequent data analysis.

Our main motivation to remove redundancy in datasets collected from multiple sources is to reduce the complexity of two of the most time consuming information related tasks: analysis and search. A representative study [Feldman *et al.* 05] on the major information-related tasks which use technologies (e.g. authoring tools, content management and retrieval software) place search and data analysis among the top five most time consuming tasks. According to the study[1] *"Analyze information"* takes *9.6 hours per week* holding the third position in the survey. *"Search"* is placed in fourth position taking *9.5 hours per week*. How important it is to automate information access is clear from the position of the world leader in searching - Google (Top 100 World Brands for 2007 statistics [Optimor 07]).

[1]First and second place are given to *"Email:read and answer" (14.5 h/week)* and *"Create documents" (13.3 h/week)*

Technically, searching can be seen as retrieving either unstructured information (e.g. text or media) or already extracted and structured data. The automation of data analysis relies on access to structured data. In order to support data cleaning process for both tasks we focus on obtaining and maintaining structured data. It can be collected either automatically or manually and then the identity resolution aims at cleaning it from redundancy in order to limit the effect of the three characteristics of a redundant dataset described above.

This work is focused on providing a general solution to the identity problem and recognises different mentions of one and the same fact coming from different sources, in order to filter out redundancy. Our main hypothesis is that variations of one and the same fact are filterable and can increase the correctness of fact extraction. Fact variations, presented in different ways, will improve the ability of a system to recognise at least one mention of the fact. Once aggregated, the information can be analysed and searched more easily, enabling the retrieval of more accurate and relevant results.

## 1.2 Problem in Focus

The identity resolution process (also known as record linkage or cross-document co-reference resolution) aims at identifying newly presented facts and linking them to their previous mentions. Unlike classical information extraction [Grishman 97, Cunningham 05] where the extracted facts are classified as belonging to a pre-defined type, fact identification creates links to previously obtained knowledge. The result of iterative identity resolution applied successfully to each new fact is a clean dataset. Our approach uses a rich semantic data representation formalism for the resulting dataset. This allows for modelling background knowledge as well as presenting facts as objects and relations between them. Ontologies provide flexible and extendable schemata for modelling facts (which can be general as well as domain specific).

### Ontologies as a Specification Mechanism

A body of formally represented knowledge is based on a conceptualisation: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them [Genesereth & Nilsson 87]. A conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or intelligent agent is committed to some conceptualisation, explicitly or implicitly.

Following [Gruber 95], we say that an ontology is an explicit specification of a conceptualisation. This term is borrowed from philosophy, where an ontology is used for systematic account of existence. For Artificial Intelligence (AI) systems what exists is that which can be represented. The vocabulary that a knowledge-based program uses to represent this knowledge is a set of objects and the describable relationships among them. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms that associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects). Gruber defines Ontologies as "including computer-usable definitions of basic concepts in the domain and the relationships among them. They encode knowledge in a domain as well as knowledge that spans domains. In this way, they make that knowledge reusable."

The word ontology has been used to describe artefacts with different degrees of structure. These range from simple taxonomies (such as the Yahoo! hierarchy), to metadata schemes (such as the Dublin Core), to logical theories. Ontologies are also not limited to definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world [Enderton 72]. The Semantic Web initiative requires ontologies with a significant degree of structure. They have to specify descriptions for the following kinds of concepts: classes (general things) in many domains of interest; relationships that can exist among things; properties (or attributes) those things may have.

According to the World Wide Web Consortium (W3C)[2] ontologies are usu-

[2] "The W3C is an international industry consortium jointly run by the MIT Computer

ally expressed in a logic-based language, so that detailed, accurate, consistent, sound, and meaningful distinctions can be made among the classes, properties, and relations. Formally, an ontology can be seen as a statement of a logical theory [Gruber 95]. As we are using ontology languages, ontology may have a very specific meaning that is determined by the particular ontology language in use. As Gruber states: "Ontology is what the ontology language allows us to describe as long as one needs to state axioms that do constrain the possible interpretations for the defined terms to specify a conceptualization."

To show the possible expressiveness, here we present the leading languages standardised by the W3C to support ontology development. Their descriptions strictly follow their specifications in the W3C recommendations.

RDF [Klyne & Carroll 04, Beckett 04] – the Resource Description Framework – is a standard way for simple descriptions to be made. What XML is for syntax, RDF is for semantics - a clear set of rules for providing simple descriptive information. RDF Schema [Brickley & Guha 04] then provides a way for those descriptions to be combined into a single vocabulary. RDF is integrated into a variety of applications from library catalogues and worldwide directories to syndication and aggregation of wide range of sources (e.g., news, software and personal collections of music, photos, and events) using XML as an interchange syntax. The RDF specifications also provide a lightweight ontology system to support the exchange of knowledge on the Web.

Although XML DTDs [Bray *et al.* 00] and XML Schemas [Biron & Malhotra 01] are sufficient for exchanging data between parties who have agreed to definitions beforehand, their lack of semantics prevent machines from reliably performing this task given new XML vocabularies. The same term may be used with (sometimes subtly) different meaning in different contexts, and different terms may be used for items that have the same meaning. RDF and RDF Schema approach this problem by allowing simple semantics to be associated with identifiers. With RDF Schema, one can define classes that

Science and Artificial Intelligence Laboratory in the USA, the European Research Consortium for Informatics and Mathematics (ERCIM) headquartered in France and Keio University in Japan leading the Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability" `http://www.w3.org/`

may have multiple subclasses and super classes, and can define properties, which may have sub-properties, domains, and ranges. In this sense, RDF Schema is a simple ontology language. However, in order to achieve interoperation between numerous, autonomously developed and managed schemas, richer semantics is needed. For example, RDF Schema cannot specify that Person and Car classes are disjoint, or that a string quartet has exactly four musicians as members.

OWL - the Web Ontology Language - provides a language for defining structured, Web-based ontologies which delivers richer integration and interoperability of data among descriptive communities. While earlier languages have been used to develop tools and ontologies for specific user communities (particularly in the sciences and in company-specific e-commerce applications), they were not defined to be compatible with the architecture of the World Wide Web in general, and the Semantic Web in particular.

Both RDF and OWL are Semantic Web standards that provide a framework for asset management, enterprise integration and the sharing and reuse of data on the Web. OWL builds on RDF and RDF Schema and adds more vocabulary for describing properties and classes: among others, relations between classes (e.g., disjointness), cardinality (e.g., exactly one), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes. It uses URIs for naming in the description framework for the Web.

### The Fact as a Main Extraction Item

According to WordNet 2.0[3] a **fact** is *a piece of information about circumstances that exist or events that have occurred*. Relying on this definition we will use the term **fact** to point to the results of an IE system.

In this work we will use an ontology as the main specification mechanism, thus the extraction results – facts – will be presented in terms of instances of concepts and relations among them as defined in the ontology. The information about these instances and their relations will be stored in a **knowledge base**.

[3] `http://www.cogsci.princeton.edu/cgi-bin/webwn?stage=1&word=fact`

For example a very simple fact is that *Sofia is the capital of Bulgaria.* Here we have two concepts, *Sofia* – a city and *Bulgaria* – a country, and the relation between them *is_capital_of.* Another example of a fact is that *James Bond* is a *covert agent* of *SAS*, created by *Ian Fleming* in *1952* in *Jamaica* (this is formally described later in the thesis).

On the other hand, since facts are expressed by the knowledge base items, we can also think about a fact as a specific segment of the knowledge base that represents a piece of information.

Following the above definition of a fact expressed by the knowledge base items, we can also think about it as a specific segment of the knowledge base presenting a piece of the encoded information.

Figure 1.1 shows an open domain ontology and some objects associated with it:

- members of class "Person" are "Tony Blair", "George Bush" and "Jeb Bush";
- members of class "Organisation" are "UK Government", "US Government" and "US Senate".

Further to this knowledge there are several explicit relations between people and organisations:

- "Tony Blair" *is member of* "UK Government"
- "George Bush"' *is member of* "US Government"
- "Jeb Bush" *is member of* "US Senate"

All these facts are present in the dataset because they are obtained from previously processed sources (shown as documents) or because they represent the system's background knowledge e.g. the political and government system, geography etc.

Identity resolution is needed when new facts are recognised by the system. In Figure 1.1 they are illustrated as two sets of person names annotated (highlighted) in a news article: one referred to as "Prime Minister Tony

Blair" and the other as "Bush". Recognising names does not by itself link the people discovered in the text into the system's knowledge base. In this case one may suppose that the referant of mentions of "Tony Blair" can be easily identified, but that instances of the entities "George Bush" and "Jeb Bush" are ambiguous, since both of them are referred to as "Bush". Our goal is to disambiguate the entity reference so to resolve the identity of these mentions.

The identity resolution process identifies "Prime Minister Tony Blair" as "Tony Blair" who *is member of*[4] "UK Government" and instead of adding redundant facts to the dataset will create a link from the document to pre-existing knowledge. In this example there are two[5] politicians whose name is "Bush" ("George Bush" and "Jeb Bush"), thus the process will disambiguat them in order to identify the correct object in the dataset to link to.

The non-dotted arrows on the figure illustrate how the mentions of the three politicians should be identified. Each of the entities is shown as a single reference point for all its mentions. Therefore the entities in the knowledge base can also aggregate information and relations to the other entities (e.g. "Tony Blair" *is_a_prime_minister* in the "UK government").

### Known Difficulties

As is clearly shown by the example above, simple comparison of the entity labels on a syntactic level (e.g. words, stems or syntactic structure matching) is not sufficient for their correct identity resolution. Providing semantics as detailed entity definitions enriches the criteria of their comparison. The expected benefit from using a semantic representation is the opportunity to associate text entities with concrete instances in a knowledge base. Then we will be able to recognise not only the type of the entities, or the concepts they belong to, but also the individual objects in the knowledge base they refer to. Such a linkage of the recognised entities to a rich knowledge base promises that they will be more easily and more correctly compared. This

[4] Or, at the present time, *was* a member of.

[5] To make the example simpler we ignore the fact that in the real world there are many people called "Bush".

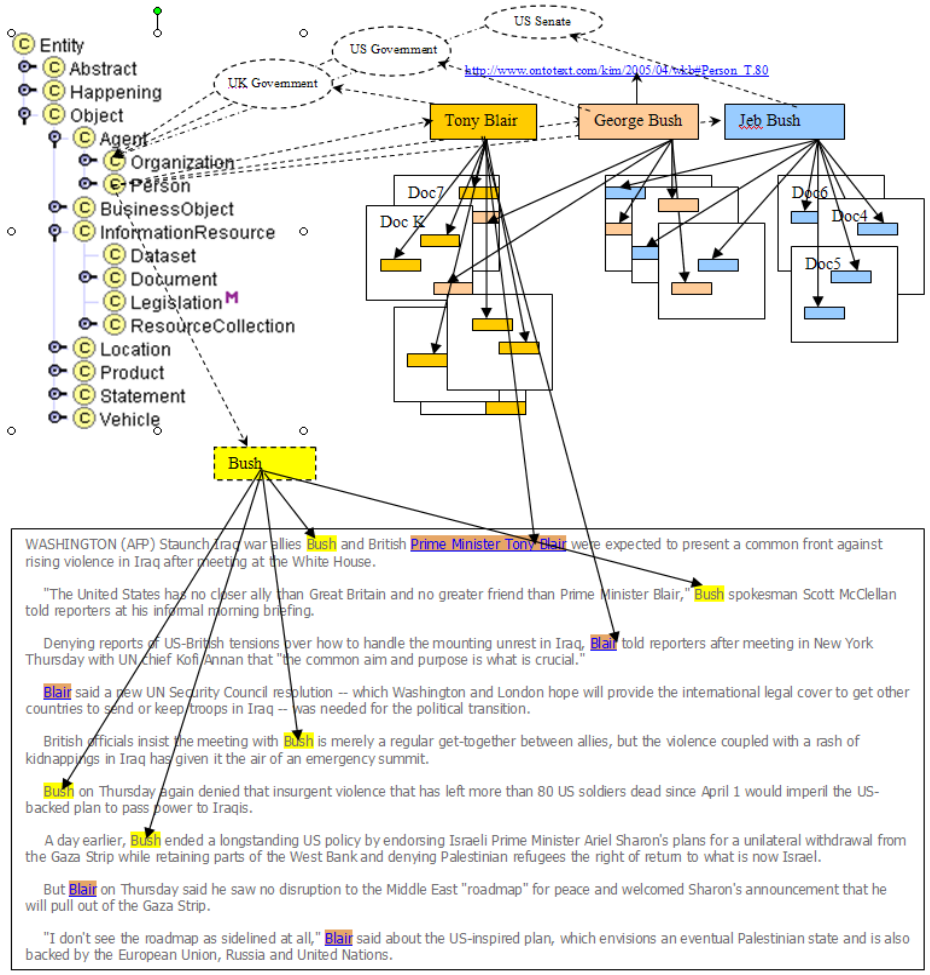


Figure 1.1: Identity resolution of Person name in a text document

is possible because of their detailed description, as opposed to a simple syntactic representation.

Tasks similar to identity resolution have previously been addressed by several different communities (see Chapter 2). This research has identified two groups of problems. These difficulties appear whenever we must deal with facts that are described in natural language using lexical expressions, proper names etc., and they usually come from natural language sources (e.g. text, video, database records etc.). Since the encoded meaning is designed to be interpreted by humans the following difficulties arise:

- The natural language labels used within facts are ambiguous and this makes them very hard to compare and merge. A common example of an ambiguous label is "Paris", which may refer to at least two completely different places: (i) the capital of France; (ii) the city in Texas, US. Another obvious example is "President Bush" referring to both the George H.W. Bush and his son George W. Bush.

- The second major group of automatic merging difficulties comes from the absence of background knowledge. Since the extracted facts are originally designed to be interpreted by people not by machines the authors address the readers' natural intelligence and ability to understand facts in context. Another phenomenon in this group is synonymy - usage of different words with identical or at least similar meanings. Identification of the same fact presented in several different ways is a challenge.

**Expected benefits**

*Improving the completeness of extracted knowledge*
One and the same information often is used for slightly different purposes by different sources of publication. Different sources also give prominence to different details; therefore individual facts can be reported in fragmentary forms. Thus identifying details spread over multiple sources and combining them into a single representation gives us a more complete knowledge about the fact, compared to the one given by a single source.

*Avoiding the extraction of incorrect information*
Regardless of the chosen technology, automatic information extraction is very sensitive to background knowledge. This is especially true for the symbolic approaches that are strongly dependant on the the collection and encoding of supporting information. Therefore enriching the background knowledge with newly obtained data fragments helps refining the extraction criteria. Although more detailed data may not improve extraction of new facts, it can significantly improve the correctness of merging new mentions of those that have already been found.

*Adding a degree of trust to the extracted facts*

Since it can happen that different sources contradict each other, one can use this as an indicator of uncertainty. It appears that it is more likely for a given fact to be true if several sources agree about it and this can determine the level of confidence for the extracted facts.

*Tracking changes over the time*
Multiple sources provide evidence for another interesting phenomenon, namely for change in information over time. Automation of the tracking of this process is very important for decision making, especially in areas where analysis of huge amounts of information is crucial. Certain details of an extracted fact might be strictly time dependent (e.g. the age of a person). Other fact details may be changed due to external circumstances (e.g. the budget of an organization, effects of medical treatment). Although some of the changes can be easily calculated (e.g. age), information about others needs to be explicitly provided or extracted from a source document.

For example an earthquake is a fact for which details may change very fast. The first news about it usually comes right after it happened at which point the details are only roughly estimated. Then the following investigations give more precise information and are usually revised further e.g. the number of victims can vary from the first hour of an accident, during the rescue actions, to the final reports. Although certain details are time dependent, their values cannot be automatically calculated but can only be obtained directly from the sources, subsequent to the original one. Hence, identification of a fact reported by multiple sources can be used to investigate how its details are changed over a certain time period.

Although there are far more opportunities to use consolidated data than those mentioned above, one important but non-trivial step should be taken. One should identify different variations or mentions of one and the same fact in order to profit from having complete information. Therefore, the main task considered in this work is the introduction of a mechanism and criteria for identity resolution of facts extracted from different sources.

**This work is organized as follows**

- Chapter 2 describes the state of the art in this area, including relevant approaches taken in several related fields such as Text Summarization, Databases and the Semantic Web. We outline systems that address the identity resolution problem as well as the recent Linked Data initiative.

- Our approach to the identity problem is presented in Chapter 3. It outlines four major stages of the process and gives details about choosing knowledge representation formalism in Section 3.1, and data preparation presented in Section 3.2,

- Chapter 4 discuss in details our approach to similarity measures - in Section 4.1 and final data fusion in Section 4.2. Each of the sections in this chapter provides theoretical background and outlines the newly proposed as well as the already known techniques that are applicable to the corresponding stage.

  We present two use-cases as walk through examples of the identity resolution process. These examples serve as initial implementation and case study of the proposed solution in two domains: recruitment and company profiling. The incoming data in both cases is gathered from the Web, thus we focus on the information extraction mechanism and methods for combining and merging this data as well as using external structured sources.

- We implemented the proposed identity resolution approach as a framework, and this work is described in Chapter 5. We outline its architecture in Section 5.1 and then provide more details about the two major components: the class model definition (see Section 5.2) and the semantic description compatibility engine (see Section 5.3) which together form the backbone of our solution.

- Chapter 6 is focused on the evaluation of the work. We discuss appropriate evaluation approaches and metrics and measure the accuracy, efficiency and maintainability of the approach and its implementation (based on the two use-cases introduced in Chapter 3).

- Chapter 7 provides a summary of our work and gives further directions for extending the approach.

# Chapter 2

# Related Work

## 2.1 Application Domain for Identity Resolution

Previous experiments in extracting information from multi-source data have been mainly done in the areas of Summarization, Multimedia Indexing and Co-reference Analysis. The task of matching entity names has also been addressed by a number of other communities, including statistics and databases. In statistics a long line of research has been conducted in probabilistic Record Linkage. The entity matching problem was formulated as a classification problem, where the basic goal is to classify entity pairs as matching or non-matching. In databases, performing Record Linkage or de-duplication creates a clean set of data that can be accurately mined. This chapter will survey the relevant work within these different communities.

### 2.1.1 IE and Semantics

Information Extraction (IE) systems extract pieces of information by mapping natural language texts into predefined structured representation - linguistic patterns, usually sets of attribute-value pairs. Some of the attribute-value pairs are to be filled in by results from morphological analysis, named entities recognition, and (partial) syntactic analysis. These processes are relatively well studied and most of the IE systems report high precision and recall. However, the semantic analysis - including recognition of references

and template filling - is a complicated process, which is still far from its ultimate solution.

Following the terminology established by the Message Understanding Conferences (MUCs), we shall call the specification of the particular events or relations to be extracted *scenario* and we shall refer to the final, tabular output format of information extraction as *template*. The actual structure of the templates used has varied from a flat record structure at MUC-4 [Lehnert *et al.* 92] to a more complex object oriented definition which was used for Tipster and MUC-5 [ARPA 93], MUC-6 [Grishman & Sundheim ] and MUC-7 [Chinchor 98].

Once filled, templates represent an extract of key information from the text [Wilks 97, Grishman 97]. Extracted information can be stored in databases for various purposes such as text indexing, information highlighting, data mining, natural language summarisation, etc. More recent approach to the IE task [Bontcheva *et al.* 09] suggests ontologies instead of flat templates.

Different systems provide different approaches for solving semantic problems in IE. The CRYSTAL system [Soderland 97], for example, is based on machine-learning covering algorithm for building expected rules for template filling. Large hand-marked training corpus is needed. But the domain is quite static - weather forecast - with explicitly fully expressed information. The system creates a formal representation of the text that is equivalent to related database entries. Another Information Extraction system is SMES [Neumann *et al.* 97], which does not have semantic analysis implemented in it. Fragments extracted by a lexically driven parser are attached to anchors - lexical entries (mainly verbs). If successful, the set of found fragments together with the anchor build up an instantiated template.

### Ontology-Based Information Extraction

There are several research tracks on Ontology-Based Information Extraction (OBIE) from the web. KnowItAll[Etzioni *et al.* 04], for example, uses syntactical patterns to extract new instances or relations, based on pre-defined examples in their ontology. However, it is limited to the sentence level, which is relatively easy to process. Another attempt for integrating ontolo-

gies as a knowledge representation mechanism into IE systems is presented in [Maedche *et al.* 02]. However most of the systems aim to extract or populate their ontologies using IE techniques [Sintek *et al.* 01].

Semantic tagging systems, for example SemTag [Dill *et al.* 03], perform a task that is complementary to that of other systems that extract facts from the web (e.g., KnowItAll). SemTag starts with TAP, a knowledge base consisting of basic lexical and taxonomic information about some popular objects [Guha & McCool ], and computes semantic tags for the chosen web-pages. The main challenge in the tagging is to choose correctly the corresponding tags and handling the knowledge base ambiguity, while in systems like KnowItAll the task is to automatically extract the knowledge that SemTag and the other tagging systems take as input. h-TechSight [Maynard *et al.* 05] presents IE algorithms that have been specifically created for particular domains and therefore they can offer the extended functionality in contract to the large-scale domain-independent approaches as in SemTag and KIM [Popov *et al.* 04]. There the authors show that manual adding of new instances would be more beneficial for domain-specific applications which can afford to be semi-automatic, unlike the large-scale systems that need to do automatic annotation.

Magpie [Domingue *et al.* 04] is another tool for semantic text annotation, with instances from a known ontology, providing a very specific and personalised viewpoint to webpages. These annotations are intended to be used as a confidence measure for carrying out some services, and to provide the users with the appropriate information supplying their different knowledge and/or familiarity needs to browse. The semantic annotation is used also in the S-CREAM project, presented in [Handschuh *et al.* 02]. The approach there is interesting with the heavy involvement of machine learning techniques for automatic extraction of relations between the entities that are annotated. A similar approach is also taken within the MnM project [Vargas-Vera *et al.* 02], where semantic annotations can be placed inline in the document content and refer to an ontology.

Another related approach is taken in OOF [Collier *et al.* 03], where ontology engineering and automatic semantic annotation are assisted by the PIA-Core IE system [Collier & Takeuchi 02]. However, this system requires an

ontology structure that is restricted to a simple hierarchy, where the classes could not be subclasses of more than one class, and instances may belong only to a single class.

Further context of using semantic annotation has been pioneered in the course of the On-To-Knowledge project [Iosif & Ygge 01]. An approach for extending an information retrieval engine with the usage of semantic metadata has been demonstrated by the QuizRDF module [Davies & Weeks 04]. QuizRDF is responsible for indexing as well as for retrieving textual information with respect to an ontology extracted from OntoBuilder [Gal *et al.* 04].

A notable aspect of using semantics for matching knowledge representation structures is presented by [Giunchiglia *et al.* 04]. The authors define *Match* as an operator that takes two graph-line structures and produces mappings among the nodes that correspond semantically to each other. However, the processing is based mainly on the node labels, even if their comparison is based on the WordNet [Miller 95] and the graph structure is restricted to trees.

[Welty & Murdock 06] present the integration issue from the reasoning prospective. Their system called KITE extracts formal knowledge from text combining information extraction techniques (built in UIMA [Götz & Suhre 04]), ontological knowledge representation and reasoning. The problem that authors focus on, is the need of integration of the extracted data, in order to perform powerful reasoning. The solution essentially uses co-reference analysis, though it is not discussed in details. Finally, the authors do not claim to give a complete solution, but rather to raise awareness of the real problem - knowledge integration - also the main topic of our research.

### Ontology Population

Ontology population is a part of knowledge base construction, where new instances are inserted in a given domain ontology. There are different automatic approaches for ontology population. [Giuliano & Gliozzo 08] use lexical substitution to extract entities from the Web. Others [Valarakos *et al.* 04], [Castano *et al.* 08], [Maynard *et al.* 08] suggest two

step process consisting of (i) information extraction; and (ii) identity resolution. The first step is generally performed by means of some kind of OBIE identifying the key terms in the text, while the second step relates them to concepts in the ontology. In this thesis we focus on identity resolution subtask only, exploiting the results from previously performed information extraction.

This task has been defined also as a precondition of Semantic Web[1] that uses the results of the semantic annotation and fuses them [Nikolov 06]. It aims at integration of all the recognised pieces of knowledge from heterogeneous sources and one of its subtasks, among mapping of different data structures and conflicts resolutions, is "instance identification" or finding the correspondence between entities in the source and target knowledge base. The authors discuss various problems e.g. identifying instances by known key fields, etc., but details description of proposed algorithms and their evaluation is missing. Another approach for identifying different data for various ontologies is given in [Jaffri *et al.* 07].

### 2.1.2 Multi-document Summarisation and Indexing

Multi-document Summarisation faces similar problems to combine information from different sources and to avoid redundancy. A good example of a multi-document summarization system is SUMMONS [McKoewn & Radev 95]. The main starting point for SUMMONS is identification of that part of the information, within the relevant documents, that needs to be included into the summary. To achieve this, the early system versions attempt to choose a set of relevant templates produced by an Information Extraction system and given to SUMMON as an input. The requirement for the templates to be within a same set is to contain a large number of similar fields as relevance evidence. Once the set is identified, the system tries to combine the templates into a simple structure, keeping the common features and marking the distinct ones. The merging process is defined in several heuristically derived summary operators. At

[1] The semantic web is an evolving extension of the World Wide Web in which web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily (source: Wikipedia `http://en.wikipedia.org/wiki/Semantic_web`)

each step, one operator is selected, based on the existing similarities between identified templates, and then applied to produce a new combined template. The operators are chosen to be independent and several operators can be applied in succession.

The more recent work is based directly on document sentences, not on extracted templates, and proposes a cross-document structure model [Radev 00]. Another method presented by the authors is cluster-based sentence utility [Radev *et al.* 00], including cross-sentence information subsumption, which reflects that a certain sentence repeats some of the information presented in other sentences and may, therefore, be omitted during summarization. Sentences subsuming each other are said to belong to the same equivalence class. An equivalence class may contain more than two sentences within the same or different documents.

Another system working on multiple sources is MUMIS [Declerck *et al.* 03, Saggion *et al.* 03], presenting an integrated solution to the problem of Multimedia Indexing and Searching. Their approach results in the generation of a conceptual index of the content which may then be searched via semantic categories instead of keywords. The system consists of using information extraction from different sources (structured, semi-structured, free, etc.), modalities (text, speech) and languages all describing the same event. Single-document, single-language information extraction is carried out by independent systems that share a semantic model. The results of all the information extraction systems are then merged by a process of alignment and rule-based reasoning that also uses the semantic model. The merging component of the MUMIS project aims to fill in missing aspects of events with information gathered from other documents. First it divides related documents into paragraphs called scenes, according to their predefined structure, and then aligns corresponding paragraphs. Due to the specifics of the chosen domain, namely football, time-stamps play an important role in defining paragraph frames for alignment. In order to combine the partial information from already aligned scenes the system uses several kinds of rules that express different aspects of domain knowledge. Shortly, MUMIS provides a methodology that uses robust named entity alignment from multiple sources together with domain specific semantic rules.

### 2.1.3 Co-reference Analysis

Co-reference Analysis is closely related to extraction from multiple sources. It refers to the process of determining whether or not multiple mentions of entities refer to the same object and enables the extraction of relations among entities as well as complex propositions. Cross-document Co-reference Analysis pushes the task into considering whether mentions of a name in different documents are the same. There is little published work on cross-document co-reference analysis, and it has generally been evaluated on a small corpus of documents [Gooi & Allan 04].

Significant work in this field is presented by [Bagga & Baldwin 98, Bagga & Biermann 00]. Their system takes as input the co-reference chain for all entities in each document outputted by CAMP (University of Pennsylvania's Information Extraction system). It then passes these documents through the SentenceExtractor module which extracts, for each document, all the sentences relevant to a particular entity of interest. In other words, the SentenceExtractor module produces a summary of the article with respect to the entity of interest. Then the VSM-Disambiguate module that uses a Vector Space Model (VSM), computes similarities between the summaries (sentences extracted) for each pair of documents. Summaries having similarity above a certain threshold are considered to be regarding the same entity.

An improvement of this approach, that yields better results [Gooi & Allan 04], is the usage of an agglomerative vector space clustering algorithm. The VSM disadvantage, that the authors claim to overcome, is the consecutive chain construction. The discussed problem appears when an entity is wrongly attached to the chain, as it further drags wrong but highly similar to entities and populates the co-reference chain with entities that do not belong there. The proposed agglomerative approach, used to solve this problem, builds up clusters in a way that is order independent. First it creates a single chain for each entity and then iteratively merges the chains with highest similarity above certain threshold. Finally, the authors note that this technique requires more comparisons and takes more time than the standard VSM.

Another work exploring clustering algorithm for cross-document co-reference

is [Saggion 07], however the system automatically summarise the documents content before clustering them.

### 2.1.4 Entity Identification

In contrast to co-reference resolution which identifies different names that correspond to one and the same object, entity identification aims at disambiguation of identical names referring to different objects. A popular example of this problem is the name "Paris" which often refers to the capital of France, but also may point to a small town in Texas. Entity identification is often addressed as author's name disambiguation in context of bibliographical records. It is preferred domain for experiments mainly because of the semi structured nature of the text and well defined entities behaviour and relations. [Aswani *et al.* 06] base their approach on web searches while looking for the author home pages, as well as on papers titles and abstracts.

Identification of names in news articles is another popular domain. [Fernandez *et al.* 06] base their approach on the categories of the news articles and "news trends" - news phenomena where one and the same entity appears in several consequent articles, usually connected with daily news. They use ontologies for internal representation and adopt their page ranking algorithm for this task. Other approaches based on web searching hits, author names distances, etc., are given in [Yang *et al.* 06]. [Kousha & Thelwall 07] also explore different web sources e.g. Google Scholar for tracking citations.

Citeseer scientific papers repository runs a project that aims for recognising identical citation in the reference section of the articles. The problem is not trivial because of the different formats used in the original string presentation of the bibliography. [Lawrence *et al.* 99] suggest several steps for normalisation and comparison of different parts of the text. The normalisation step includes removal of prefixes, punctuation signs, etc. Further comparison is based on either the entire record or different parts it consists of. The main limitation of this approach is that it does not proceed with full parsing of the citation and sophisticated comparison of its building parts, but relies on the string representation.

Another innovative approach to entity identification is given in [Hassell *et al.* 06]. Although this woks is focused only on entity disambiguation (assuming that all entities are already encoded in the system knowledge base), the authors present a novel algorithm for entity comparison. They use ontologies as internal knowledge representation formalism, enabling entities as well as relation storage. The presented system disambiguates researchers' names appearing in a collection of DBWorld posts. The system is based on a huge pre-collected set of researches, organisations, etc. obtained from DBLP site. Although the text processing is restricted only to entity look up, individual documents are used as a context where the entities appear. The comparison algorithm explores the background knowledge (e.g. the number of publications per author), as well as the relations among the entities. The way the entities appear in one and the same document (e.g. how close they are, measured in number of characters, etc.) serves as a clue for their identification. Hence the explicit relations stored in the system knowledge base are compared to the implicit ones found in the text.

Although the entity identification approaches does not differ dramatically from the philosophy of our work, their goal slightly differs from ours. They are topically limited to a close domain, while they are either based on exhaustive pre-collected background knowledge or presume on the textual clues looked up over a big corpus. Therefore the identification of mentions cannot be applied directly to the more general task - identity resolution of entities from various source types - the goal of this work.

### 2.1.5 Record Linkage

Traditional work in de-duplication for databases or reference-matching for citations analysis measures the distance between two records according to some metric, and then collapses all records at a distance below a threshold. This task is most frequently solved by examining individual pairwise distance measures between mentions independently of each other and setting a distance threshold below which records are merged [Monge & Elkan 97]. This method does not take into account the records inter-dependent relations thus reducing the merging accuracy.

Most recent efforts in the area are focused on learning the distance met-

ric [Bilenko & Mooney 03]. Bilenko and Mooney present a framework for duplicate detection using trainable measure of textual similarity (a learnable text distance function). Learned distance metrics are used to calculate distance for each field of each pair of potential duplicate records, creating distance feature vectors for the classifiers. The binary classifiers categorize the resulting feature vectors for each candidate pair as belonging to the class of duplicates or non-duplicates. Pairs are sorted by increasing confidence.

[Leit *et al.* 07] propose a framework for matching XML entities based on a Bayesian network (BN) model to explicitly represent the dependencies among object attributes. The objects are derived from the structure of the XML entities. The approach calculates and combines the similarity for direct attributes as well as for descendant values of two XML entities, and the probability threshold above which entities are considered matches is manually set.

More sophisticated work in this direction has been done by [McCallum & Wellner 03]. They present relational models which do not assume that pairwise co-reference decisions should be made independently from each other. Unlike other relational models of co-reference that are generative, the conditional model can incorporate a variety of features, without having to be concerned about their dependencies. They use a graph representation of objects and relations and claim that their solution has a relational nature, because the assignment of a node to a partition (or, mention to an entity) depends not just on a single low distance measurement to one other node, but on its low distance measure to all other nodes in this partition (and furthermore on its high distance measurements to all nodes of all other partitions).

A similar approach using text-edit distance metrics and record field structure has been described in [Cohen *et al.* 03]. There record pairs are presented as feature vectors - the distances between corresponding fields. Again, the binary classifier trained on these features is used to score the confidence in the match class.

## 2.2 Systems Addressing the Identity Resolution Problem

There is a wide range of systems, platforms and frameworks that face the identity resolution problem. For some of the systems this is not the main problem, therefore they solve it only partially. Others implement a complete identity resolution approach either for a particular domain, or generally. [Köpcke & Rahm 10] present a detailed comparison on the features of some of the systems discussed below.

### 2.2.1 Nilesh Dalvi et al.

[Dalvi *et al.* 09] propose a general method for matching reviews to objects. They explore a dataset of 24,910 Yelp reviews covering 6,010 out of about 700K restaurant records in Yahoo! Local database. The main goal in this work is to identify the object of the review matching unstructured text to a list of structured object. The algorithm starts from a list of structured objects (restaurants/cameras/movies) and given a text review, it identifies the object from the list that is the topic of the review. For this purpose the authors use a language model for generating reviews.

The authors claim that apart from the description of the restaurant, the model contains all the words used in the review. The intuition behind the language model is that the reviews are written about an object and each word in the review is drawn either from a description of the object or from a generic review language that is independent of the object. Combination of both description and the language model gives a principled method to find the object most likely to be the topic of the review. Based on the evaluation in this work, using language model-based method vastly outperforms traditional tf.idf-based methods. Further the authors advocate that the proposed method is light-weight and scalable and can substitute highly expensive information extraction.

The result of this work is a link between a textual document and an object and does not enrich the original object description. It may be seen as a competitive approach to standard information retrieval techniques, while

the identity resolution is only partially covered and the last step of fusing information from multiple sources is omitted.

### 2.2.2 Active Atlas

The Active Atlas system proposed by [Tejada *et al.* 01], [Tejada *et al.* 02] suggests using machine learning techniques for two identity resolution stages. The authors emphasize on the cost of the manual configuration of data transformation rules for pre-processing new data sources and normalising their values, and on retrieving object identification rules. They propose an approach to tailor a general set of transformations to a specific domain application. The main goal of this transformation is to resolve format inconsistencies in different sources.

The system uses a training method to get a combination of several decision tree learners to define attribute based similarity rules. The intuition behind it is that some of the attributes are more important in the identification then others. Finally the single attributes are matches using a variation of TFIDF extended with additional information (e.g., stemming, abbreviations) to support shallow match of two attribute values.

The system also supports an initial blocking strategy based on hashing that selects potential matching candidates. In contrast to our approach, Active Atlas is fixed to database records representation and does not allow usage of any knowledge about the nature of attribute values and the relations among them. It also does not provide a mechanism for fusing identified objects. However the main contribution of the author is the novel method to combine both forms of learning to create a robust object identification system.

### 2.2.3 WHIRL

[Cohen 00] address the problem of identity resolution over a database records as integrating data from sources that lack common object identifiers. Their solution is designed as a "soft" database management system which supports "similarity joins". The criteria for similarity are robust and general-purpose

metrics over the textual description. The solution is designed to process records that contain natural language labels as attribute values.

After defining similarity over a single attribute, the authors define WHIRL as a query language that allows conjunctions of various attributes. In this way one can define a similarity function over entire database record and retrieve similar objects. The goal of the language is to provide one step access to not exactly equivalent information usually collected from different sources.

WHIRL based system however lacks data normalisation process that unifies format variations with relatively low computational cost. A pre-selection step is also missing which leads to processing the entire dataset while calculating the similarity. Finally the system returns all found records without any further aggregation. Its main strength however is the declarative language and its robustness.

### 2.2.4 Flamingo Project

The Flamingo Project[2] provides a framework for similarity string matching. The authors reduce the record-linkage problem to linking similar strings in two string collections based on a given similarity metric. Their approach is based on mapping similarity spaces into similarity/distance-preserving multidimensional Euclidean spaces. They propose two-step solution.

In the first step, they combine the two sets of records and map them into a high-dimensional Euclidean space using a specially developed algorithm called StringMap. It has a linear complexity and it is independent of the distance metric; other mapping techniques can also be used instead of this one. Initially the distances between all records are calculated, and then they are placed in high-dimensional space.

In the second step, the algorithm finds similar object pairs in the Euclidean space whose distance is within a threshold. The threshold can be recalculated to reflect the specificity of each pair. In case of using multiple attributes for presenting a single record the merging rules are expressed as logical formula over similarity predicates based on individual attributes. The authors

[2] `http://flamingo.ics.uci.edu/`

suggest that such rules can be generated by a classifier such as a decision tree and result in an overall similarity function between records using a labelled training set. Based on the merging rule the algorithm chooses a set of attributes over which the similarity join is performed, such that similar pairs can be identified with minimal cost.

Finally the result of the record process returns pairs of identical object, and their processing and fusion is placed outside of the scope of the project. The intuition about using logical formulas for combining different attributeswith the same content is the same that is used in our approach; however we use similar mechanism in two steps - for restriction of the candidates' pool and later in the actual comparison. Decision on object similarity in Flamingo is incorporated in the similarity representation process, while we suggest a separate step for identification followed by data fusion. Some other common feature of Flamingo and the approach proposed in this work is independence from particular implementation of similarity measures.

A more recent research around the project goes in the direction of efficient fuzzy string search in database record [Vernica & Li 09] and parallelizing the similarity join algorithm using Map-Reduce [Vernica *et al.* 10].

### 2.2.5 Febrl

FEBRL (Freely Extensible Biomedical Record Linkage) [Christen 08] is a recently developed open source framework for record linkage. It was originally developed for entity matching in the biomedical domain, it is written in Python[3] and provides GUI for data manipulation. It supports four basic de-duplication steps: (i) Cleaning and standardisation; (ii) Blocking; (iii) Field comparison and (iv) Weight classification.

The main task of data cleaning and standardisation is the conversion of the raw input data into well defined, consistent forms. It is then passed to the blocking step, which output is a set of candidate record pairs. They are gathered by applying a restriction over a single record field or a combination of fields called a "blocking key" that splits the databases into blocks. Based on the key the data is indexed and the framework supports disjoint overlapping

[3] `http://www.python.org`

blocking methods based on several indexing algorithms e.g. BlockingIndex [Baxter *et al.* ], the sorted neighbourhood approach [Hernandez *et al.* 95].

The Field comparison is base on similarity functions applied to the field values of the record pairs. The user has to select one of the various measures as well as the fields that will be compared. Usually the comparison is performed on fields with the same content, and using different fields is also visible. A large selection of 26 different similarity measures such as Jaro, edit distance, and q-gram distance is available for attribute value matching. Some of them are implementation of phonetic encoding e.g. Soundex, NYSIIS, and Double Metaphone that allow for detecting similar names.

The weight classification is needed to decide their final linkage status as matches, non-matches, and possible matches. It is based on a weight vector formed for each compared record pair and it contains all the matching weights calculated by the different comparison functions. These weight vectors are then used to classify record pairs. Febrl supports several classification algorithms that are based either on manually tagged data or apply unsupervised techniques.

Examples of the supervised techniques supported by the frameworks are Decision trees, which authors [Goiser & Christen 06] have chosen as a baseline to compare other unsupervised classifiers. Another classifier that is implemented and requires training set is Support Vector Machine. Besides manual training selection two unsupervised algorithms are supported : K-means and Farthest-first.

Finally one of the three possible categories “match”, “non-match”, and “possible match” is assigned to each candidate pair. If the decision is either “match” or “unmatch” this will be the final result and the last step in the process. However, the class of possible matches requires manual/clerical review. It is assumed that the person undertaking this clerical review has access to additional data (or may be able to seek it out) which enables her or him to resolve the linkage status.

In contrast to Febrl approach we propose rule based definition of blocking constrains and decision on the identity of the pairs candidates. The main driving force behind our choice is to allow for real time resolution where the entire pool of entities for blocking is not available beforehand. The decision

in our approach is totally automatically taken, and the process does not end at this stage. We also propose a data fusion step which resolves possible conflicts in attribute values of the two identical entities. In spite of all the differences mentioned here and the fact that Febrl works only for database record like entities, compared to the rest of the systems presented in this work, it is the one that is the most similar to our approach.

### 2.2.6 MOMA

MOMA (Mapping-based Object MAtching) [Thor & Rahm 07] is a framework for mapping-based object matching. The design of MOMA is inspired by our previous work on schema matching - COMA approach [haiDo & Rahm 02] - which allows the combined use of multiple match algorithms for a given schema match problem. Thus it provides extensible library of matchers, both attribute value and context matchers. Its architecture allows for specification of a workflow of several steps each of which combines several existing mappings or matcher executions. Each workflow step consists of two parts: matcher execution and mapping combination.

According to its authors, the key feature of MOMA is that it uses the notion of instance mappings. The algorithm either calls selected matchers or only combines existing or previously computed mappings from a mapping repository. The results are accommodated in the mapping repository used for materialisation of same-mappings between objects.

It further allows for combining several mappings as specific mapping combiner. The input of a mapping combiner is a list of independent mappings the output is a same-mapping. A combiner is specified by a mapping operator followed by an optional selection. The output of a mapping step is represented as a so-called same-mapping (set of corresponding entity pairs) indicating which input objects are considered semantically equivalent.

MOMA also provides self-tuning capabilities to automatically find optimal configurations for a mapping task. It automatically selects matchers and mappings to find the optimal configuration parameters. Initially the focus is on optimizing individual matchers and combination schemes. The parameters are choice of attributes to match, the similarity function that will be

calculated, as well as the similarity threshold to be applied. For example a so-called neighbourhood matcher implements a context-based match approach and utilizes semantic relationships between different entities, such as publications of authors or publications of a conference.

MOMA supports compose and merge operators to combine different mappings resulting in a mapped / not mapped linked between the objects, which goes also on the attribute level. However data fusion and conflict resolution strategies are not supported. It does not explicitly offer blocking methods. However, a blocking-like reduction of the search space could be implemented by a liberal attribute matching within a first workflow step whose result is then refined by further match steps.

### 2.2.7 SERF

SERF (Stanford Entity Resolution Framework) [Benjelloun *et al.* 09] is a generic entity resolution infrastructure. The main focus is not on improving the efficiency of simple matchers (similarity functions) but on reusing existing techniques viewing them as "black boxes". The main contribution in this work are several algorithms that aim at minimizing the number of invocations to computationally expensive similarity calculations by exploring previously compared values thus avoiding redundant comparisons. The architecture of SERF is based on two black-box functions provided as input to the ER computation: match and merge.

Match function takes two records and returns boolean output whether they represent the same entity - true, or they are different - false. The framework does not support confidence score as a numeric value. Although the similarity measures may return such values, they are transformed to a yes-no decision. The main argument of avoiding confidence is the processing complexity it may introduce to the system.

Matching works pairwise. It calculates similarity of corresponding record attributes in the pair and the comparing algorithms depend solely on the data in these records. Thus matching decisions are made locally, based on the two records being compared. An advanced option is usage of multiple matchers that can be combined by a disjunction of manually defined simple

match rules. However possible relations between records are not taken into account if the records are different objects. Only values that can be attached directly to the objects are considered.

A merge function takes two records as input and returns a single record. It fuses the values of the objects in those pairs that are defined as matching records, i.e., records known to represent the same entity. The output in this step is a record that consolidates the information from the two records and represents a single entity.

The framework implements three scenarios:

- G-Swoosh - the most general algorithm that makes no assumptions about the match and merge functions. It is very similar to the "brute force" algorithm that compared each record to all the others; however it caches already processed pairs. In this way it avoids making the same calculation, but retrieves the result for a storage.
- R-Swoosh - is the algorithm that is optimal in terms of record comparisons, it may still perform redundant comparisons of the underlying values. It reduces the number of invocations to the match and merge functions relying on two sets: input records and already processed non-matching records. R-Swoosh compares each record in the input set to all records in the processed set and performs a merge as soon as a pair of matching records is found. Finally the current entity is removed from the input set and if it is a non-matching record, it is added to the processed set. Performing merges and deletions as early as possible, it avoids unnecessary match comparisons.
- F-Swoosh - is an improved R-Swoosh version that is further optimised on the attribute comparison level. The intuition behind it is that while different records may share common values same attribute comparisons may be performed redundantly. It keeps track of encountered values and the result of their comparison, thus it avoids repeated feature comparisons and can be significantly more efficient than R-Swoosh.

As all the presented systems so far, SERF uses flat database model for data representation therefore it supports limited similarity functions e.g.,

not exploiting relations. However it addresses the scalability issue and the authors present detailed scale evaluation of all the three approaches. This is a good reference point for us to compare to, since our approach considers more complex techniques as using confidence and rich data model. As it is shown in Section 6.3 our approach outperforms SERF in a domain of comparable complexity.

### 2.2.8 TAILOR

TAILOR [Elfeky *et al.* 02] is a toolbox for record linkage with a GUI that allows users to build their own record linkage models by tuning system parameters and by plugging in in-house developed and public domain tools. Finally it provides means for reporting statistics, such as estimated accuracy and completeness, which can help the users understand better the quality of a given duplicate detection execution over a new data set.

TAILOR follows a layered design isolating the comparison functions from the duplicate detection logic. Apart from the GUI and the database management system itself that provides actual access to the data, it contains four functional layers: Searching Methods, Comparison Functions, Decision Models and Measurement Tools.

The Searching Method is used to reduce the size of the comparison space. The authors point to the fact that it is very expensive to consider all possible record pairs for comparison. Therefore they proposed two approaches for data pre-selection: blocking and sorted neighbourhood. Thus both disjoint as well as overlapping selection methods are supported. The Comparison Functions calculate the similarity between attributes. They are used for comparison of attribute values and include hamming distance, edit distance, Jaros algorithm, q-grams and Soundex.

The Decision models are addressed by adopting a machine learning approach. Three probabilistic models are proposed: an induction model, a clustering model and a hybrid model. The induction model is an example of possible usage of supervised learning techniques. The two examples of such classifiers implemented in TAILOR are decision trees and instance-based learning. In order to use them the user has to provide training data.

Under Clustering models the authors mention unsupervised learning techniques. They use k-means clustering to cluster the compared pairs into three clusters, one for each possible matching status, matched, unmatched, and possibly matched. After applying the clustering algorithm to the set of comparison vectors, the issue is to determine which cluster represents which matching status. It is based on the assumption that if two records agree on all the attribute values, their comparison vectors have zero distance. Thus the nearest cluster to the origin is the one that contains matched pairs.

The Hybrid record linkage model combines supervised and unsupervised techniques in two steps. In the first step, clustering is applied to predict the matching status of a small set of record pairs while in the second step, a classifier is used for building a classification model following the pattern of the induction record linkage model.

Finally the results are passed as an input to various Measurement Tools that aim at helping the user to adjust the system parameters in each step and to measure the success of the newly built application. In this way TAILOR supports the full life cycle of a record linkage application. The missing element needed to make it an identity resolution supporting tool is data fusion and enrichment.

### 2.2.9 Other tools

WizSame[4] by WizSoft is a commercial product that applies simple string matching algorithms for discovering duplicate records in a database. The matching algorithm is very similar to SoftTF.IDF - two records match if they contain a significant fraction of identical or similar words, where similarity is defined as words that are within an edit distance of one.

## 2.3 Linked Data Initiative

The Linked Data initiative addresses the Web of Data concept. It is an alternative view to the World Wide Web (the Web of hypertext) which consists of interlinked documents (web pages). The new concept interprets

[4] `http://www.wizsoft.com/default.asp?win=9&winsub=45`

the HTML links as possibly pointing to a structured data representation instead of another page. Thus Linked Data refers to data instead of content published on the Web. The main benefit of using structured data is that it is in a machine-readable from. Moreover it can be referred to from any content page of data sources as external data sets. In this way one data set can link to another data set that can link to a third data set and so on, which results in a web of data.

The formal definition of Linked Data is given by [Berners-Lee 06] as a set of rules for publishing data on the Web. The main principle is that the links represent relations between objects described in RDF and alike the web pages each object has got a Unique Resource Identifier (URI) [Berners-Lee *et al.* 09] in the global data space. The four fundamental rules are:

- Use URIs as names for things - It follows closely the notion of the Web where each resource is synonymously identifiable with its name.
- Use HTTP URIs so that people can look up those names - The URIs should use *http://* scheme, so they can be simply looked up over the HTTP protocol. The benefit of using such URIs comes from the fact that the HTTP protocol provides a simple and universal mechanism for retrieving resources as a stream of bytes. It can be applied to RDF resources in a way similar to HTML content.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL) - This refers to the common practice to publish ontologies (the schema of the data), but not to provide direct access to the actual data that is described using this schema. The data entries are often disposed as a single archived package or trough a custom access API. This rule gives directions in both ways: how to formally represent the data; and how to provide direct independent access to each entity. The RDF model encodes the data in a form of triples ( subject, predicate, object) where the subject is a URI and when used as a link to other resources the object has to be URI as well (further discussion on semantic knowledge representation standards is given in Section 3.1).

SPARQL is W3C standardised query language for RDF and can be used to express queries across diverse data sources. Since data described in RDF forms a direct, labelled graph, it is capable of querying required and optional graph patterns along with their conjunctions and disjunctions. The results of SPARQL queries are subgraphs of the queried RDF graphs.

- Include links to other URIs, so that they can discover more things - This rule follows the etiquette widely adopted in the hypertext web sites to refer to external materials, thus to point to the value of the presented information as well as to promote the inherent value from other sources. This practice has been proved successful in WWW so far, therefore the authors of the Linked Data rules encourage it.

According to [Bizer *et al.* 09] Linked Data directly builds on the general architecture of the Web employing HTTP URIs. Thus the authors suggest that the Web of Data can be seen as an additional layer to the classic document Web, which shares the same properties:

- The Web of Data is generic and can contain any type of data.
- Anyone can publish data to the Web of Data.
- Data publishers are not constrained in choice of vocabularies with which to represent data.
- Entities are connected by RDF links, creating a global data graph that spans data sources and enables the discovery of new data sources.

In summary any resource that is published following the Linked Data rules can be easily made part of the Web. This provides unlimited potential for enriching the information space with structures identical resources. Hence the links between different mentions of the same entity over various datasets can be seen as a result of manual or automatic identity resolution.

### 2.3.1 Linked Open Data Project

Linked Open Data (LOD) project[5] is a home for resources that apply Linked Data principles. This effort is initially funded and supported by W3C Semantic Web Education and Outreach Group[6] and aims at collecting and publishing datasets, tools and research activities for adopting Linked Data rules. The project is widely supported including a number of large organisations as BBC, Thomson Reuters and the Library of Congress and already collects over 70 different interconnected data sets.

The most important requirement for each data set in order to be included in the LOD cloud is to be connected to some of the other already present datasets. So far the cloud contains datasets of several different categories shown in different colours on Figure 2.1:

- bio-informatics in pink
- publications in green
- social network data in orange
- geographic locations in yellow
- entertainment in violet
- open domain knowledge in blue

As one can notice certain datasets serve as linking hubs in the cloud. For example DBpedia[7] is a dataset that is regularly updated and extracts information from Wikipedia info-boxes as RDF statements. As of April 2010[update], the DBpedia dataset describes more than 3.4 million things, out of which 1.5 million are classified in a consistent Ontology, including 312,000 persons, 413,000 places, 94,000 music albums, 49,000 films, 15,000 video games, 140,000 organizations, 146,000 species and 4,600 diseases. It consists of over 1 billion pieces of information (RDF triples), thus it is one of the largest publicly available datasets and naturally plays the role of a meeting point for other datasets.

[5] `http://linkeddata.org`
[6] `http://www.w3.org/2001/sw/sweo/`
[7] `http://dbpedia.org`

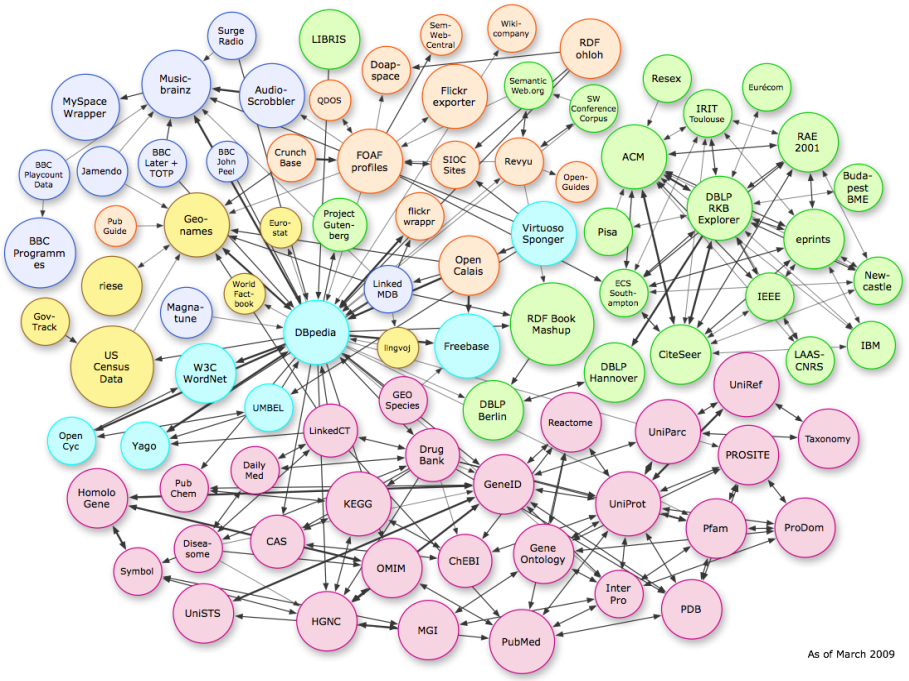


Figure 2.1: Datasets in LOD cloud as of March 2009

### 2.3.2 Link Discovery Tools

The emerging usability of Web of Data concept provoked recent research in identity resolution of resources described in RDF. It is common practice to use automated or semi-automated methods to generate RDF links and the resulting tools aim at supporting data publishers in providing links to the pre-existing information about the entities they refer to. In this way the contribution of a given source to the public is made explicit and reusable.

In certain domains e.g. publications there are generally accepted identifiers of objects e.g. ISBN and ISSN numbers, EAN and EPC product codes. Therefore they are often used as identity evidence in discovering implicit relationship between entities in the datasets, and generating explicit as RDF links. However if shared naming schemas does not exist, then one can apply identity resolution process computing the similarity of entities based on a combination of different similarity metrics over an entity attributes. An example of such system is Silk.

**Silk**

Silk Linking Framework[Volz *et al.* 09] is a toolkit for discovering and maintaining data links between Web data sources. It works in line the Linked Data initiative and employs the RDF data model to publish structured data on the Web. however it is mainly devoted to creating explicit data links between entities within different data sources and providing means for maintaining them when the data in different sources change. Silk consists of three components:

- A link discovery engine, which computes links between data sources based on a declarative specification of the conditions that entities must fulfil in order to be interlinked;
- A tool for evaluating the generated data links in order to fine-tune the linking specification;
- A protocol for maintaining data links between continuously changing data sources.

Link discovery process starts with pre-selection of those entities that are potentially matching. All the entities in the target dataset are pre-indexed using the values of one or more attributes (often labels). The lookup in this index uses the BM258 weighting scheme for ranking of the results and additionally supports spelling corrections of individual words of a query. The pre-selection results are then normalized before being passed for actual comparison. Silk provides twelve predefined data transformation functions that can be used for processing attribute values.

Attribute similarity is calculated by configuration of several different metrics. For this purpose the framework includes a specification of a declarative language called "Silk - Link Specification Language" (Silk-LSL). It describes the type of RDF links that should be applied as well as how different similarity measures can be combined in order to calculate a total similarity value for a pair of entities. The supported similarity metrics in Silk are about ten - include string, numeric, date, URI, and set comparison methods as well as a taxonomic matcher that calculates the semantic distance between two concepts within a concept hierarchy.

These similarity metrics may be combined using the following aggregation functions:

- AVG - weighted average
- MAX - choose the highest value
- MIN - choose the lowest value
- EUCLID - Euclidian distance metric
- PRODUCT - weighted product

The final result of the comparison will be single similarity score that reflects how similar two resources are. However the final decision about introducing a link between them is taken based on a threshold comparison. The resource pairs with a similarity score above 0.9 are to be interlinked and pairs between 0.7 and 0.9 require manual confirmation made by an expert. The tool for evaluating generated data is a web interface allows the user to quickly evaluate and confirm or reject the links.

The purpose of the "Web of Data - Link Maintenance Protocol" (WOD-LMP) is to keep the created links between the source and the target datasets in line. This includes removing dead links to nonexistent recourses i.e. removed from the target dataset, and generating new links for entities added to the target dataset. The protocol automates the communication between two cooperating Web data sources. The protocol covers the following three use cases:

- Link Transfer to Target - aims at keeping the target set notified about subsequent updates in the source set (i.e. additions and deletions). Here the source set is proactive in informing the target set that may respond with creating back-links.
- Request of Target Change List - In this use case, the source data source requests a list of all changes that have occurred to RDF resources in a target dataset within a specific time period.

- Subscription of Target Changes - the active role is given to the target set that notifies the source about all changes by sending a Change Notification message. The source may then use this information to recompute affected links.

With its three components Silk covers the entire lifecycle of connecting date resources from its discovery and introduction to their update and removal. Although it covers the identity resolution problem, the authors do not discuss the schema alignment task.

In the context of Linked Data, the identity resolution process that we propose can assist building links between the two datasets. It starts from mapping their data representation schemas and proceeds with discovering identical objects in both datasets. Although our approach suggests additional step of fusing the identical objects into a single accommodative entity, it goes beyond the concept of Linked Data. Therefore the it should be finished before this step with only introducing a link between corresponding objects.

# Chapter 3

# Identity Resolution Architecture and Data Preparation

In this chapter we discuss our general approach to the Identity Resolution (IdR) problems seeing it as discovering references to one and the same object that come from different sources. Solving this problem is important for a number of different communities (e.g., Database, NLP and Semantic Web) that process heterogeneous data. The desired effect of identity resolution is removing redundancy and consolidating the incoming data that is used for further data analysis. This problem can also be seen as part of the data integration task, where many instances of identical or complementary data represented in different data sets are collected together.

Another aspect of the heterogeneity is that the variations of the same objects are obtained in different formats e.g. textual documents, web pages, database records, ontologies, etc., and this fact introduces another layer of complexity. Therefore identity resolution aims at creating a single view to the data where different facts are interlinked, the noise of redundancy is removed and incompleteness is dissolved. As a result of this process one can easily query and use enlarged data set for various tasks. This problem is also known as "object consolidation" or "record linkage" in the Database community, or "cross document co-reference resolution" in NLP community,

"ontology population" in Semantic Web community, etc.

There are many examples of applications (e.g. [Jaffri *et al.* 07]) that face the identity resolution problem. Systems that retrieve information from various ontologies usually rewrite their queries to retrieve as much data as they have access to. Finally the received data is gathered from different sources and contains many repetitions, so to obtain consistent results the query rewriting applications need to resolve the identity of the same objects across different ontologies (e.g., is "J. Davis" from ontology O1 the same person as another "J.Davis" in Ontology O2).

Another scenario for IdR is information extraction from different sources. The main goal of such applications is to extract new information from a steam of documents. Thus, the decision about which information is new and which has been already extracted is crucial for the application success. The identity resolution helps with identifying the newly extracted facts with those already seen and stored in the knowledge repositories. It also enables further aggregation of details about the known objects, so each of the identified mentions may provide additional information to the initial object description.

In this work we propose four step process for identity resolution: *(i)schema alignment; (ii)candidate selection; (iii)similarity measure; (iv)data fusion* (see Figure 3.1).

### Step 1 - Schema Alignment

At the first step we propose aligning all possibly different formats of the original sources. This comes from the fact that the incoming data can be described against different database schemas, xml schemas, object-oriented data models, ontologies , etc., and in order to identify similar objects they should be presented in comparable formats. This leads us to a need of a common data representation that can be descriptive enough to hold all object attributes even if they are present in one data source and missing in the other. Formally this new representation should be:

- rich enough to contain all attributes that are likely to distinguish duplicates,

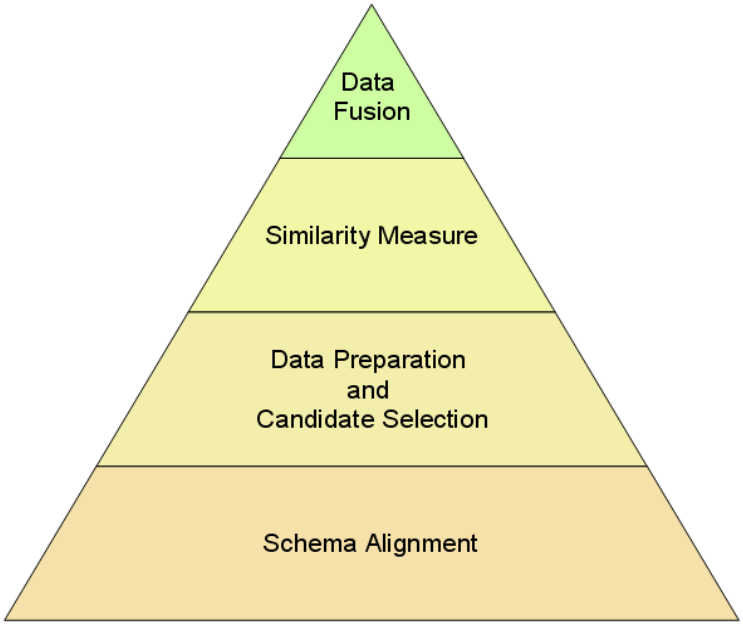


Figure 3.1: Four main stages in identity resolution process

- flexible enough to allow further extensions of the data model once a new data source is added to the system,
- keeping track of the semantics of each attribute and its relations to the other attributes and objects.

In the following section 3.1 we advocate that ontology based knowledge representation satisfies our requirements and is easy to be mapped to other data representation formalisms.

Once we have a single access or view of the data - referred later as source set $S$ - we are ready to proceed with the actual objects comparison. We propose examining the incoming objects one by one matching them to the set of already processed data. The comparison can start from an empty set or initialised set of non-duplicated objects that are previously processed or created manually. The target data set (noted below as $T$) will grow naturally to the size of all unique objects. In each iteration of the identity process over the objects in $S$, i.e. $c_{cur} \in S$, they will be either:

- merged with an existing record in the target data set if $\exists o_j \in T\,(o_{cur} \approx o_j)$,
- or added to the data set as a new object $T = T \cup o_{cur}$.

Following the above definition, finally the target data set $T \subseteq S$ will consist of all unique objects from the source data sets.

### Step 2 - Candidate Selection

Once an object from the source dataset is selected, it is compared to all entities in the target to discover its identity. While the target data set grows, one to one comparison of each new coming object to each target object becomes more computationally expensive. Therefore we introduce the next step - **candidate selection** - that aims at reducing the number of required pairwise calculations. During this step, we use strict criteria that need to be present in order to consider two objects as identical. This criteria are general enough to classify objects as non-identical or possibly identical and to form a set of comparison candidates $T'$. For example, we may want to compare objects of the same type only, e.g., an organisation record should not be compared to a vehicle description. Other general or domain specific restrictions can also be defined here and reduce the pool of candidates. Ideally the criteria should form a $filter()$ function that satisfy the following requirements:

- selects only candidates that are similar to the currently processed object - increasing the precision of the $filter()$ function:

  For a threshold $\theta$ and a similarity function $sim()$

  $$sim() : S, T' \rightarrow R' = \{a : a \in R, 0 \geq a \geq 1\}$$

  $$filter(o_{cur} \in S) \rightarrow T' :$$

  $$\forall o_i \in T'\,(sim(o_{cur}, o_j) \geq \theta)$$

- not to leave a possibly similar candidate outside the selected candidate set - this is the characteristic that is considered to be more important than the previous one and we formally define optimal recall as:

$$\forall o_i \in T \setminus T' \, (sim(o_{cur}, o_j) < \theta)$$

The balance between precision and recall should be considered while selecting concrete restriction criteria. Then one needs to decide on the trade-off between calculation complexity (number of similarity calculations performed in the next step) and the accuracy (finding the best identity match). It might be different for each domain and application, therefore a universal solution does not exist (further discussion on how to select candidates and set a $filter()$ function is presented in 3.2).

### Step 3 - Similarity Measure

Once selected an object $o_{cur} \in S$ is compared to each of the candidates $o_j \in T'$. We define it as a function $sim(o_1, o_2)$ - pairwise **similarity measure** - that reflects the likeliness of two object to be identical. At this stage we still do not decide on the identity but only measure how similar the current object is to each of the filtered candidates.

Following the ontology paradigm in Step 1, the objects are described as sets of attributes and their values can be of different types (strings, numbers, or other objects). Therefore, the similarity function takes into account the similarity between different attributes that could be processed by very different comparison algorithms. In order to combine the results from individual comparison of attributes, we define a total similarity function of two objects as a first order probabilistic logic formula resulting in a real number between 0 and 1. A value of 0 means that the given objects are totally different and value 1 means that they are equivalent.

Each formula - a similarity function - is manually composed by combining predicates with the usual logical connectives "$\vee$", "$\wedge$", "not" and "$\Rightarrow$" where the building elements of the formula are atomic measures - predicates - that process one or more attributes of the compared objects.

For example, the comparison of objects of type Person can be formulated as

$$sim(p_1, p_2) = same(p_1.name, p_2.name) \wedge same(p_1.age, p_2.age),$$

$$same(o_1, o_2) = \begin{cases} 0, o_1 \neq o_2 \\ 1, o_1 = o_2 \end{cases}$$

There are different measures that can be applied to each of the object attributes, e.g., various string similarity measures, numbers comparison. Since the main goal of this work is not to advocate one criterion among the others, but to introduce widely applicable solution of the identity resolution problem, we present some of the widely used measures as well as a suggestion for their combination and customisation. Furthermore, we introduce a mechanism for new criteria to be added, thus the proposed solution gives huge flexibility for building application and domain specific *sim()* functions. The similarity measurement step is discussed in details in Section 4.1.

**Step 4 - Data Fusion**

This is the step where we decide how to interpret the results of comparing a current object to the target set. It is a two step process which starts with choosing the best candidate from the target set and continues with fusing the current object and the candidate into a single updated record in the target set. One of the simplest approaches would be to select the candidate with highest similarity score above a certain threshold and then work with it.

$$sim(o_{cur}, o_j) > \theta \Rightarrow o_{cur} \approx o_j$$

However, more complex strategies could also be used, e.g., select a candidate from the set of top ten highly scored ones that is described with more attributes, etc. The exact algorithm for selecting candidate pair depends on the application domain and the entire context of performing the identity resolution task.

One can stop with simply pointing to the fact that two objects are identical and this is useful in many situations where one wants to keep the original records. The link can then be used later for data integration or Extract, Transform, Load (ETL) [1] processes while the original records stay unchanged

[1] `http://en.wikipedia.org/wiki/Extract,_transform,_load`

and used by third party applications. Another scenario would be to merge the two original records into a single one that represents both. This process is also known as data fusion [2]. Formally, the fusion function $fusion()$ results in a new object $o_3$ such that:

if $o_1 \approx o_2$ , $fusion(o_1, o_2) = o_3$
then $o_1 \approx o_3$ and $o_2 \approx o_3$.

In our data model, where each object is described by a set of attributes, fusing objects is reduced to combining their attributes. For each attribute type $a$ we merge the corresponding values for the two objects - $o_1.a$ and $o_2.a$ into a new attribute value $o_3.a$. We consider three possible scenarios:

- if one of the source attributes is not specified, then we chose the one that is present
  $o_3.a = o_1.a$, if $o_2.a$ is null or $o_3.a = o_2.a$,if $o_1.a$ is null
- if the two source attributes have the same value, then we chose this value
  $o_3.a = o_1.a = o_2.a$
- if the two source attributes are contradictory, then one can define various strategies for resolving the conflict, e.g., choosing more descriptive values, etc.

More details on data fusion and various strategies for attribute combination including those that consider the semantics of different values (e.g., ontology class hierarchy) are given in Section 4.2.

So far we have outlined the four basic steps of the identity resolution process, starting from the raw data and resulting in a consistent integrated data set where all the duplicates and conflicts are cleaned. Each of these steps, however, is a challenging task by itself, therefore in the following sections we aim at clarifying possible constrains as well as our achievements in solving them. Then we implement our approach as a framework for identity resolution of various types of objects in different domains. It uses ontologies as internal knowledge representation formalism, the benefits of which are described in the following section.

[2] `http://en.wikipedia.org/wiki/Data_fusion`

## 3.1 Knowledge Representation

The knowledge representation used in the proposed solution is based on ontologies. It has been chosen because of its detailed entity description formalism that is complemented with semantic information. Ontologies are widespread and they are already successfully used for IE (see Section 2.1.1) as well as in other areas (e.g., in biology and chemistry). Several ontology description languages have already been standardised and we will presented them briefly below.

The ontologies are already used for approaching the identity resolution problem. [Funk *et al.* 07] present the advantages of semantically enhanced annotation for resolving co-references from different sources. Another example of using ontologies in this domain is the innovative work of [Klein *et al.* 07] for extending standardised ontology description languages to unable approximation of instances. The authors introduce new "Rough Description Language" to represent and reason about similarity of instances.

We consider Ontologies as the most appropriate knowledge representational formalism mainly because of the expected benefit from using semantic representation and its opportunity to associate general type objects with concrete instances of ontology classes. In this way we will be able to recognise not only the type of the objects, or the class they belong to, but also the individual instances they refer to. Such a semantic linkup of the identified objects guaranties more detailed description as opposed to a simple syntactic representation. In this way it provides more details, which serving as evidence can improve the accuracy of object comparison. The ontological representation also provides a standard mechanism for introducing relations between concepts that can be used for further comparison enrichment (e.g., the most popular relation *sub class of* gives us the class hierarchy).

### 3.1.1 Entity Description

Entity Description is a formal representation of a fact that corresponds to a concrete ontology. The entity has an unique identifier (URI) and belongs to a certain ontology class and its features are presented though a set of predefined properties and relations valid for the particular class.

For example let us create a simple ontology for describing people, organisations and places. The following script represents OWL definition of the classes and a few simple properties and relations. Initially we start with defining the ontology parameters e.g., ontology language, version, namespaces, etc.

```
 <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY psys "http://proton.semanticweb.org/2006/05/protons#">
  <!ENTITY ptop "http://proton.semanticweb.org/2006/05/protont#">
  <!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#">
]>
<rdf:RDF xmlns:owl="&owl;"
         xmlns:psys="&psys;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
 xmlns = "&ptop;"
         >

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""
                rdfs:comment="PROTON (Proton Ontology), Top module"
                owl:versionInfo="0.2">
        <owl:imports rdf:resource="protons.owl"/>
  </owl:Ontology>
```

Then we continue with describing the main classes and their subclasses. In this way one actually builds the class hierarchy shown on Figure 3.2, where the top class is Object. Then Person is subclass of Agent, which is subclass of Object. Location is also subclass of Object; and Organisation which is subclass of Group is also subclass of Object. JobPosition is a class that is defined as being subclass of Situation, and to make the example simple [3], lets make Situation also a subclass of Object.

[3]This is a derivation from the original definition of PROTON, where more complex hierarchy is suggested.

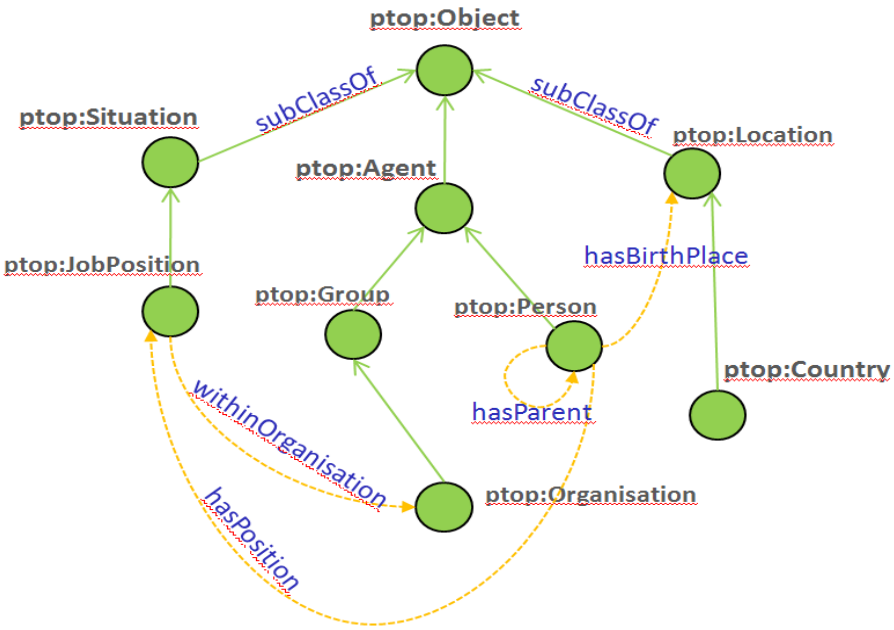


Figure 3.2: Ontology graph

```
<!-- Classes -->

  <owl:Class rdf:about="&ptop;Person"
             rdfs:label="Person">
    <rdfs:comment>A Person is an agent ..., which is an individual
     who is a human being (i.e. any living or not alive member
     of the family Hominidae). Wordnet 2.0.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&ptop;Agent"/>
  </owl:Class>

  <owl:Class rdf:about="&ptop;Agent"
             rdfs:label="Agent">
    <rdfs:comment>An Agent is something, which can show (carry out)
      an independent action, whether consciously or not. Most animals
      are considered agents; so are most organizations...
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="&ptop;Object"/>
  </owl:Class>
```

```
 <owl:Class rdf:about="&ptop;Location"
            rdfs:label="Location">
   <rdfs:comment>Usually a geographic location on the earth,
    however any sort of 3D regions also fit here. The classification
    is based  on the ADL Feature Type Thesaurus version 070203...
   </rdfs:comment>
   <rdfs:subClassOf rdf:resource="&ptop;Object"/>
 </owl:Class>


<owl:Class rdf:about="&ptop;Organization"
            rdfs:label="Organization">
   <rdfs:comment>Organization is a group, which is established in
     such a way that certain known relationships and obligations
      exist between the members, or organization and its members...
   </rdfs:comment>
   <rdfs:subClassOf rdf:resource="&ptop;Group"/>
 </owl:Class>

 <owl:Class rdf:about="&ptop;Group"
            rdfs:label="Group">
   <rdfs:comment>A group of agents that is not organized in any way.
    This could be the group of people in a bus or the shareholders
    of a company.  </rdfs:comment>
   <rdfs:subClassOf rdf:resource="&ptop;Agent"/>
 </owl:Class>

 <owl:Class rdf:about="&ptop;Object"
            rdfs:label="Object">
   <rdfs:comment>Objects are entities that could be claimed to exist
    - in some sense of existence...</rdfs:comment>
   <rdfs:subClassOf rdf:resource="&psys;Entity"/>
 </owl:Class>


<owl:Class rdf:about="&ptop;JobPosition"
            rdfs:label="Job Position">
```

```
    <rdfs:comment>The situation of a person, holding a job position
     within an organization. </rdfs:comment>
    <rdfs:subClassOf rdf:resource="&ptop;Situation"/>
  </owl:Class>

   <owl:Class rdf:about="&ptop;Situation"
              rdfs:label="Situation">
    <rdfs:comment>A static event or situation, like
      "sitting on a chair" or "holding position"...
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="&ptop;Object"/>
  </owl:Class>
```

The focus of our example will be on Person having the following:

- properties: *name*; *birth_date*
- and relations with entities of other classes:
  *hasBirthPlace:Location*; *hasParent:Person*; *hasPosition:JobPosition*

There is a standard property for providing names of the resources - rdfs:lable (see `http://www.w3.org/TR/rdf-schema/#ch_label`), therefore we will need to define only one property of this class, namely birth date. In contrast to the relations, its range is an atomic value, therefore it is not explicitly specified.

```
<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="&ptop;hasBirhtDate"
                        rdfs:label="has Birth Date">
    <rdfs:comment>Recommended best practice for encoding
     the date value is defined in a profile of ISO 8601 [W3CDTF]
     and includes (among others) dates of the form YYYY-MM-DD.
    </rdfs:comment>
    <rdfs:domain rdf:resource="&ptop;Person"/>
  </owl:DatatypeProperty>
```

Lastly, we can point to some relations between people, job positions, places and organisations. We say a Person can hold a Job Position and this Job Position can be defined within a Organisation. A Person can also be associated with a Location

as being his/her birth place. The relations can also be inherited; here we define the relations hasRelative between two members of class Person and then we say hasParent is a subproperty of hasRelative.

```
<!-- Object Properties -->
 <owl:ObjectProperty rdf:about="&ptop;hasPosition"
                     rdfs:label="has Position">
    <rdfs:domain rdf:resource="&ptop;Person"/>
    <rdfs:range rdf:resource="&ptop;JobPosition"/>
    <owl:inverseOf rdf:resource="&ptop;holder"/>
  </owl:ObjectProperty>


 <owl:ObjectProperty rdf:about="&ptop;hasBirthPlace"
                     rdfs:label="has BirthPlace">
    <rdfs:domain rdf:resource="&ptop;Person"/>
    <rdfs:range rdf:resource="&ptop;Location"/>
  </owl:ObjectProperty>


 <owl:ObjectProperty rdf:about="&ptop;hasParent"
                     rdfs:label="has Parent">
    <rdfs:subPropertyOf rdf:resource="&ptop;hasRelative"/>
    <owl:inverseOf rdf:resource="&ptop;hasChild"/>
  </owl:ObjectProperty>

 <owl:ObjectProperty rdf:about="&ptop;hasRelative"
                     rdfs:label="has Relative">
    <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    <rdfs:domain rdf:resource="&ptop;Person"/>
    <rdfs:range rdf:resource="&ptop;Person"/>
  </owl:ObjectProperty>


   <owl:ObjectProperty rdf:about="&ptop;withinOrganization"
                     rdfs:label="within Organization">
    <rdfs:comment>Determines in which organization is the position
    </rdfs:comment>
    <rdfs:domain rdf:resource="&ptop;JobPosition"/>
```

```
    <rdfs:range rdf:resource="&ptop;Organization"/>
  </owl:ObjectProperty>


</rdf:RDF>
```

Then a concrete instance of class Person will be associated with a certain identifier and described with values of all/some of its properties and relations.

For example the instance - **Person#007** is described with the following

- properties:
  - *name*: **"James Bond"**;
  - *birth_date*: **"January 1952"**
- and relations with entities of other classes:
  - *has_birth_place*: **Jamaica**;
  - *has_parent*: **Ian Fleming**;
  - *occupies_position*: **covert agent**
  - *position_in_organisation*: **SIS**

Formally this fact will be represented by a set of RDF triples using the ontology defined above. First we present definition of James Bond:

```
<example#Person.007> <rdf#type> <ptop#Person>.
<example#Person.007> <rdfs#label> ‘‘James Bond’’.
<example#Person.007> <ptop#hasBirthDate> ‘‘1952-01’’.
<example#Person.007> <ptop#hasPosition> <example#Job.41397>.
<example#Person.007> <ptop#hasBirthPlace> <example#Contry.356>.
<example#Person.007> <ptop#hasPerent> <example#Person.JF03>.
```

It refers to other entities in the knowledge base that we can briefly define as:

```
<example#Person.JF03> <rdf#type> <ptop#Person>.
<example#Person.JF03> <rdfs#label> ‘‘Ian Fleming’’.

<example#Contry.356> <rdf#type> <ptop#Location>
<example#Contry.356> <rdfs#label>‘‘Jamaica’’.

<example#Job.41397> <rdf#type> <ptop#JobPosition>.
```

```
<example#Job.41397> <rdfs#label> ‘‘Covert Agent’’ .
<example#Job.41397> <ptop#withinOrganisaiton>
                                <example#Organisation.MI6>.

<example#Organisation.MI6> <rdf#type> <ptop#Organisation>
<example#Job.41397> <rdfs#label> ‘‘SIS’’ .
```

In fact, one can define a data structure of arbitrary complexity extending the ontology, then our formal description of James Bond will still be valid and it can also be enriched with further details. Thus applications can use various subsets of the data ignoring details that are not useful for them. On the other hand, the data model can be extended with new attributes if needed for new applications without changing existing applications. This flexibility of ontology based semantic descriptions is one of the reasons for our preference for this representation formalism. Another preferred feature is its high degree of descriptiveness allowing any data structure to be represented as a set of triples. Some examples of how different data representations can be mapped to ontologies is given in the following section.

### 3.1.2 Schema Alignment of Different Data Types

While we claim that the proposed solution is general, it is very important to show that it supports various kinds of incoming data. Given that the data can be either unstructured or structured, we have taken examples of both: free text documents as the most common example of unstructured data; database records as the most popular type of structured data representation; and ontology knowledge base as a modern approach to data structuring.

#### Information Extraction from free text documents

Before merging any information, one should extract those pieces of text that contain relevant information. This brings us to recognising different entities and facts in the text before trying to identify them. The process might also be seen as classical Information Extraction (IE) template generation task, although the slot restrictions can be extended over an ontology class definition. Though, in order to assist further processing, we propose performing of ontology based IE instead of using classical IE techniques.

By extraction of facts with respect to an ontology, we mean recognising entities along with the classes they belong to. Possible relations are also recognised at this

stage and they are limited to those, predefined in our conceptualization model - the ontology.

In some very specific and predefined cases one can also identify instances at this step, e.g., in a specific business domain one can always link the string "IBM" to a company with id="Company.12028IBM" in the application knowledge base. However possible false positives here may introduce inconsistency propagating the errors in the extracted data, therefore such a practice is not generally advisable.

A more complex aspect of the extraction is template generation based on already recognised entities and relations. The definition of possible attributes, properties and relations for each ontology class can be seen as a template definition. Then recognising an entity of this class and filling its properties and relations, corresponds to template generation.

Possible technology choice for free text analysis is already available as combination of several systems:

- The text pre-processing presented in the IE pipeline consists of tokenisation, sentence splitting and part of speech tagging. All these steps are successfully performed by the corresponding GATE [Cunningham *et al.* 02] processing resources and are reused directly.
- Further looking up of known entities can be performed by KIM Gazetteer [4], since it also provides information about class and instance features of the look-ups.
- Named Entity(NE) recognition is an IE technique that proposes instance candidates and it is already implemented in KIM [Popov *et al.* 04], based on GATE, where the NE recognition is done with respect to an ontology. Thus once the entities are recognised, they bear unique identifiers that link them to the concepts (and possibly instances) in the ontology.

Once the entities and composite facts are extracted from the text they present different mentions of objects that we want to refer to from the already collected knowledge. The next step needed for successful linkage is their identification with the existing knowledge base.

### Transformation of Databases

Transformation of databases is used to solve the well known record linkage problem as general identity resolution during integration of data from several different

[4] part of Gate plugins

sources. Databases are well known and very efficient data storages based on tables where the data is already structured in rows and columns. The database schema is the data description that holds the meaning of the data, although the relations and the semantics of table elements are limited and often hard to interpret. Therefore, tracing identity of database records requires more powerful representation where the links between objects are explicitly shown. So, once the data is parsed and stored in a rich semantic representation, then we can apply the proposed approach for resolving identity issues. However, binging databases to other knowledge representational formalism e.g., ontologies requires deep understanding and domain expertise and is usually done manually, producing mapping between the particular database schema and a given ontology.

Once the mapping between the database schema and the ontology is produced, the records of interest are easily transformed into formal entity descriptions with respect to the ontology. The next step before inserting them in the ontology knowledge base is that each entity is passed to the identity resolution process that finds its place and inserts either only the new details or creates a new instance for it. The result from identity resolution is a knowledge base that serves as a single reference point for all different mentions of one and the same entity, throughout the entire source database. Finally, again using supplied database mapping, the identified entities in the knowledge base can be transformed back to their original representation as database records.

### Mapping of ontologies

Semantic integration or schema matching typically focuses on finding one-to-one correspondences (or mappings) among the concepts in ontologies [Noy & Musen 01, Milo & Zohar 98]. Ontology mapping or (semi)automatic discovery of mapping between the classes of two ontologies is a topic outside the scope of this work. However, for the purpose of the work, we assume that such a mapping is already produced either automatic or manually and it can be further used for data transformation. Thus the source data coming from an ontology are the entities and their entity descriptions fully or partially mapped to the identity resolution resulting ontology.

These entity descriptions are actually the native data representation for our approach to identity resolution. As described in Section 3.1 we have chosen ontologies for internal, as well as resulting knowledge representational formalism because they are designed to keep a single reference to different variations of one and the same data.

### 3.1.3 Schema Alignment - Case Studies

#### Job Offers Collection

The first experiment performed within this work is about combining information from the recruitment domain, where we are interested in two types of facts called, respectively, *vacancy* and *contact*. The project that provides the framework for this experiment aims to supply users with the most accurate recruitment information that extracts offers about vacant positions at UK companies. It uses the Internet as a source, namely the web-sites of companies, job-boards and recruitment agencies, where it is possible to rely on many sources to improve the extraction of the facts we search for. The proposed approach is to extract all available facts from any single document, and then to combine/merge them on several levels to retrieve the most accurate facts, while at the same time filtering out wrong and redundant information.

This effort is part of an ongoing project at Ontotext Lab[5]. The system aims at collecting vacant positions from corporate web sites automatically. The system searches the World Wide Web and gathers information from more than 150,000 corporate web-sites in the UK like the one shown in Figure 3.3. Each of the job positions announced on these web-sites will be automatically included in the database. One of the project's key strengths is its ability to combine information from multiple sources into single profiles. Even if a job is mentioned several times on the Internet, its corresponding profile appears just once, but becomes more detailed in the database. As a result, the system could be used as a provider of up-to-date, daily information about companies, and on demand it could report the information back to the users. Vacancies may be searched for by type, location, skills and/or industry sector.

This project integrates wide range of technologies in Natural Language Processing (NLP) and semantic indexing and retrieval.

- The KIM platform provides the infrastructure and services for automatic semantic annotation, indexing, and retrieval.
- The Information Extraction platform GATE provides a key natural language processing technology and assists in the extraction of information directly from the web.
- Sesame - the development of which is led by Aduna b.v. is used here as a Resource Description Framework Schema RDF(S) repository.

[5] `http://www.ontotext.com/`

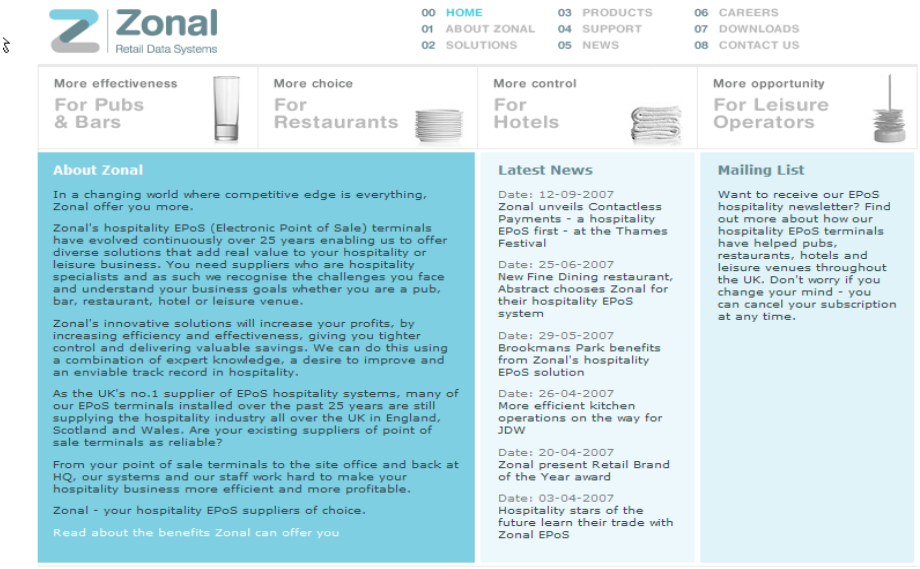


Figure 3.3: "Zonal" corporate web-site : http://www.zonal.co.uk/

- Lucene - an open-source engine for Information Retrieval and keyword-based indexing used for retrieving queried source documents.

### Initial data acquisition using Information Extraction Techniques

In this case study we are interested in corporate vacancies, which are mainly available on the web listed on corporate web sites. In order to automatically collect these job offers from semi-structured or unstructured free text, we use Information Extraction (IE) techniques. The extraction module is responsible for the detection of vacancies in a set of web pages within a given web-site.

The input documents - web pages are already pre-classified - are job related, i.e., possibly contain job offers by a third party module. The exact algorithm used for classification go beyond the scope of this work, however the impact of its applications is significant for the IE module performance reducing the processing time.

The extraction algorithm takes web-pages one by one and processes them separately, extracting listed vacancies. At this preliminary stage each page is pre-processed and certain types of named entities are recognised and annotated with respect to the ontology. After that, the set of extracted vacancies will be further investigated for duplicates and finally inserted into the knowledge base. As we will see later in Section 6.2.1, pre-classification of web pages also improves overall extraction accuracy by filtering irrelevant noisy documents.

### Pre-processing

Since the processed web-pages are presented in HTML, they are first parsed so that images, scripts and forms are discarded. Then, we use domain-specific wrapping to transform all tables that list job positions.

We transform the table so as to include the column title in each of the succeeding cells of the same column. In this way, the title and the value are bound together in a single cell string, and we use the title as a context for correct attribute recognition. The most difficult part of this transformation is to recognise the title row, since it can happen to be not the first one, but it might be preceded by other tabular information (because of the free HTML usage). The technique we use is based on key word recognition and follows the approach described in [Tao 03].

The flat text pre-processing, which is the main task in this step, presents the classical IE pipeline, consisting of tokenisation and sentence splitting, performed by the corresponding GATE processing resources. Then, we lookup known (already persisting in the knowledge base) entities by the KIM Gazetteer[6]. In contrast to other gazetteers (e.g. the build in GATE gazetteer component) that mark only the type of the found entities, KIM Gazetteer adds information about the corresponding ontology class of the annotated entity as well as its instance id within the knowledge base. Thus created metadata is more detailed, and allows for reasoning and generalisation based on the ontology hierarchy. For example in contrast to the well known type annotation of "London" being of type "Location", KIM Gazetteer will mark it as belonging to class "Capital" that is a sub-class if "City", which is sub-class of "Location".

The next step in the text processing is NE recognition. The technique is rule based, where the rules are a set of JAPE (Java Annotation Patterns Engine)[7] grammars using patterns based on both the gazetteer lookups and on the context in which the entities appear. Although both classical and domain-specific entity types are recognised, the former are limited to very basic Persons and gazetteer based Locations and Organizations. NE types representing the recruitment domain are: Job Titles, Job Type, Job Category, Reporting To Position, Starting and Closing Date, Salary Rate, Reference Number, etc. All these categories present possible vacancy attributes and would be analysed and ranked during the subsequent extraction steps.

### Single Vacancy Extraction

As we mentioned already, the focus of this work is the extraction of **vacancy** facts. Both facts are defined by templates, which slots can be filled by the following related

[6] KIM Gazetteer is a module of KIM platform `http://ontotext.com/kim`

[7] `http://www.gate.ac.uk/sale/tao/index.html`

to each other entities - concept instances in our KB : Vacancy Title, Location, Reference Number, Salary, Reporting To, Start Date, End Date, Job Type, Job Status, Company Name, and Category. The proposed values for these attributes are those annotated as named entities by the previous NE recognition step. Hence the extracted facts are actually a compilation of the attribute values in accordance with the domain constraints (see Figure 3.4).

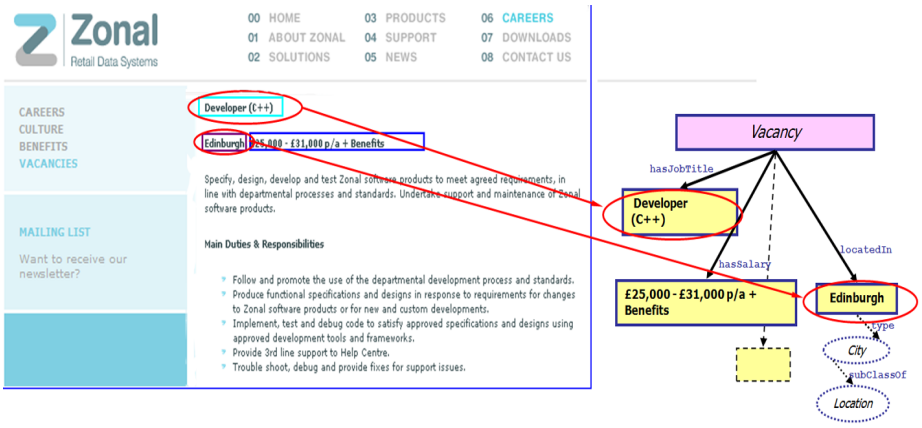


Figure 3.4: Single Vacancy extraction from a web page

Even though more than one fact can be presented in a single document, the analysis of the domain shows that vacancies are consecutively described in the text. Thus, the chosen approach for discovering fact boundaries is based on the assumption that each attribute has only one value per fact. So, if there are several annotated values for a given attribute, we take them as referring to different facts. Following the assumption for sequential description of facts in the text, we associate the second found value per attribute with the next fact.

Finally the vacancy extracted from the webpage in Figure 3.4 will look like the following:

```
<joci#Vacancy.15893>  <rdf#type>   <joci#Vacancy>
<joci#Vacancy.15893>  <joci#hasOrganisation> <joci#Company.zonal>
<joci#Vacancy.15893>  <joci#hasJobTitle> "Developer (C++)"
<joci#Vacancy.15893>  <joci#hasSalary>
                              "J25,000 - J31,000 p/a + Benefits"
<joci#Vacancy.15893>  <joci#hasDatePosted> "1st of April 2010"
<joci#Vacancy.15893>  <joci#hasTownName> "Edinburgh"
```

Once a fact is extracted, we use two features to check the reliability of the extracted vacancies.

- The first one is the level of recognised attributes. The extracted facts should, at least, contain a value for "Vacancy Title" attribute for an open position. If this attribute is not provided, the vacancy would not be compiled.
- We call the second reliability feature *confidence level*. Its value is a real number from 0 to 1, showing the probability that a given extracted fact presents a real vacant position. This is required because a great variety of information presented in the web-pages, such as biographies and head position descriptions, is wrongly recognised and extracted as vacancy facts. The measure of the *confidence level* is based on fact attributes binding, as well as on the page structure and context, including page title and page URL.

### Needs for Identity resolution of Vacancies

When all possible vacancies are extracted, we proceed with identification of the unique ones.

The set of job offers collected in the first experiment consists of automatically extracted vacancies with 45% rate of redundancy (see Table 6.4), e.g. two of each five vacancies are a duplicated version of another vacancy in the set usually described with less vacancy details.

What we achieve as a result of merging two vacancies is a new vacancy composed out of the most specific values among the two proposed values for each and for every attribute. Attribute values present only in one of the merged facts are also taken.

Following the above definition of one to one merging of vacancies, we will discuss the more complicated merging of one vacancy to a set of candidates. As described in Chapter 1, the collecting of a candidate set is based on attribut comparison. However, we are motivated, considering the domain, to use one of the attributes as the main reference point, namely we are looking for a strong equivalence of "Vacancy Title" attribute values.

Moreover we found it practical to filter out the candidates set on three consequent levels. The first two are based on the candidates' origin (single page or whole web-site) and the third is the merging to the knowledge base itself. The motivation behind the breaking of the merging process in pieces is that, once reduced, the set of facts is more consistent and contains only the most correct and detailed facts, and this would ease the further merging to the knowledge base. To achieve this, we present extracted candidates/facts as self-dependent, but we keep all known mappings of their attributes values to the knowledge base (e.g. the value "London" of attribute "Location" would be mapped to a specific instance in the ontology that presents a city - the capital of the UK, called London)

- Page level merging - helps with improving single fact extraction. Very often the information about current job position is given in two parts of the same document and the vacancy detection algorithm recognises two, not just one, vacancies with no duplicating attributes. Their combination presents the full vacancy description. Their merging at this step will improve vacancy correctness and will help further merging mainly because of the completeness of the fact attributes.

- Site level merging - The motivation for merging at this level is the fact that one and the same position could be promoted several times on a single (company) web-site, starting with the list of vacant positions, followed by a very short description and usually there is a separate page with a detailed description of all vacancy details. All this information gives us a chance to check the extracted facts and to collect all the available information provided by the employer when it is distributed on several pages.

- KB level merging - Another very challenging task (still in progress) is to use the knowledge base itself as the last level, or the final sieve, for the extraction of new facts. This will help us ensure that the fact is actually new and was not already inserted in the KB (extracted from another source, e.g. another web-site on the Internet e.g. job-boards). At the current stage of the development of the project, all facts extracted from a single company web-site are considered unique. For simplicity, we assume that organizations publish on their web-sites only vacancies of their own, although we are fully aware that this is not the case with the job-boards and the recruitment organizations. Following this assumption, the set of vacancies merged on the site level are simply added to the knowledge base.

### Company Profiles Collection

The next of our case studies in identity resolution applications is in business intelligence. The work to be described here has been carried out in the context of the business intelligence (BI) Musing Project[8]. The goal of Musing project is developing tools and modules based on natural language processing (NLP) technology and reasoning to mitigate the efforts involved in gathering, merging, and analysing/annotating multisource information for BI applications. Here multisource information plays an important role since it is unlikely that a single (textual) source will contain all up-to-date information required by the target application.

[8]EU-funded projects MUSING (IST-2004-027097)

For this use-case we use two data sources. First one is a manually built database of static company information that consists of basic information such as name, addresses of the head office and the site offices, contact details and management team. The second set is dynamically obtained from various web sites and apart from company identification details (e.g., name and address), it describes valuable financial information.

We start with a custom defined ontology specified by the application that will use the result of the identity resolution process, called Musing. It describes the company profiles entities with all attributes provided in both data sources. Therefore we align the original data formats to it. The extracted company details are described as text on a web site and we translate them in terms of entities with respect to the ontology. The database records are consistent to a certain schema and in order to make them useful to the identity process we map the schema to the ontology as well.

### Mapping a Database Schema to Ontology

Many databases already contain information relevant for our application domain. Therefore we have pre-populated our knowledge base with the data for 1,801,868 different companies in the UK that is already available in database format from a company called "Market Location"[9]. There are three tables in the database schema shown on Figure 3.5. This description holds the meaning of the data, although the relations and the semantics of table elements are limited and often hard to interpret. As a consequence, bringing databases to other knowledge representational formalism e.g. ontologies requires deep understanding and domain expertise and is usually done manually producing mapping between the particular database schema and the given ontology.

In this case, we have obtained a detailed textual manual about the meaning of each field and its relations to the others. Based on it, one can manually compile transaction rules that explicitly map the fields to entity properties. In this work, we use company profiles stored in a MySql Relational DataBase Management System which has been manually mapped to the domain ontology using scripts e.g. on Figure 3.6. Examples of the record fields in the database are: organization, section, url, name, address, etc. The scripts map for example a record field such as "organization name" into the attribute "hasAlias" in our Musing ontology.

Once the mapping between the database and the ontology is produced, the records of interest are easily transformed to formal entity descriptions with respect to the ontology. Before inserting a new instance in the database, the identity resolution

[9]http://www.marketlocation.com

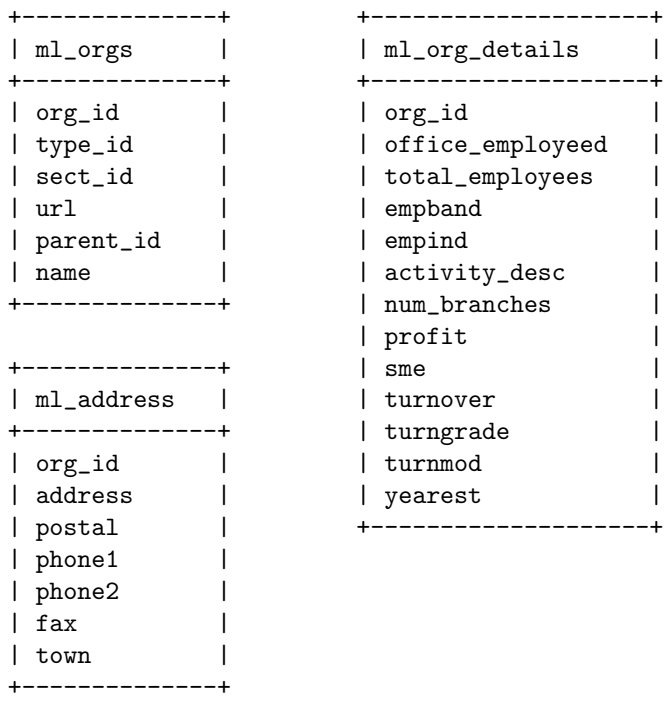


Figure 3.5: A sample of the RDBMS schema related to company profiling

```
_columnToURI.putForward("ml_orgs.org_id",    "musing#hasOrgId");
_columnToURI.putForward("ml_orgs.type_id",   "musing#hasType");
_columnToURI.putForward("ml_orgs.name",      "rdfs#label");
_columnToURI.putForward("ml_orgs.sect_id",   "musing#hasSector");
_columnToURI.putForward("ml_orgs.url",       "musing#hasWebsite");
_columnToURI.putForward("ml_orgs.parent_id","musing#hasParentID");
_columnToURI.putForward("ml_address.postal","musing#hasPostal");
_columnToURI.putForward("ml_address.phone", "musing#hasPhoneNumber");
_columnToURI.putForward("ml_address.phone2","musing#hasPhoneNumber");
_columnToURI.putForward("ml_address.fax",    "musing#hasFaxNumber");
_columnToURI.putForward("ml_address.address","musing#hasAddress");
_columnToURI.putForward("ml_address.town",    "musing#hasTownName");
_columnToURI.putForward("ml_org_details.total_employees",
                                              "musing#hasNumEmp");
_columnToURI.putForward("ml_org_details.turnover",
                                             "musing#hasPhoneNumber");
```

Figure 3.6: Mapping between RDBMS and an Ontology for Company Information

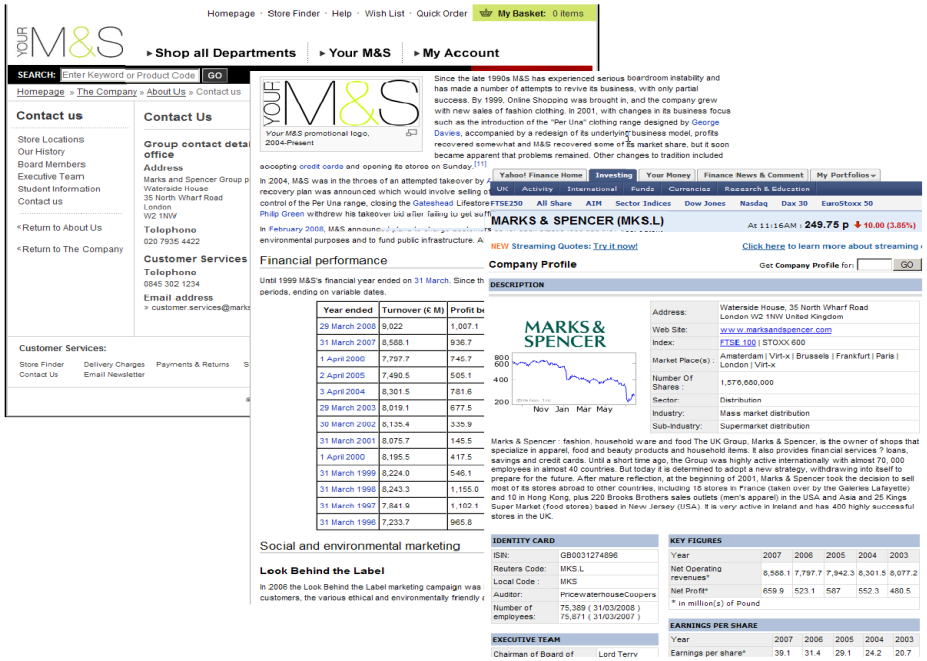


Figure 3.7: Example of a company profile across different sources

process is called upon to find the referent of the instance: inserting only the new details or creating a new instance for it. The resulting knowledge base is a reference point to all different mentions of known entities.

### Ontology-Bases IE of Company Profiles

Another set of company profiles are extracted from various web page, e.g., as is shown on Figure 3.7, using a third party ontology-based information extraction system. It has been developed within the GATE platform which provides a set of tools for development of information extraction applications.

The extraction prototype uses some default linguistic processors from GATE, however the core of the system, the concept identification program was developed specifically for this application. In addition to specific processes such as phrase chunking, lexicons and gazetteer lists have been created to perform gazetteer lookup processes. Rules for concept identification have been specified in regular grammars implemented in the JAPE language. A key element in the annotations created by the system is the encoding of ontological information - our applications create *Mention* annotations which make reference to the target ontology as well as the ontological concept string of text refers to.

### Needs for Company Profile integration

The main reason for integration of data from different sources in this BI application is the high demand for up-to-date information about companies. The databases tend to be exhaustive but static and because they contain huge number of record they are often not updated regularly. On the other hand the web sites compete to be the most accurate source and indeed quickly reflect changes in company related information although the number spotted companies per source are restricted to its area of interest (e.g., only listed on the stock exchange market).

In this case study, we are interested in consolidation or merging of information found in different sources by ontology-based information extraction (OBIE), with the one that already resides in the knowledge base. The process identifies text concepts with instances, and relations expressed in an ontology.

## 3.2 Data Preparation and Candidates Selection

In the overview of this section we presented the main logic of the identity resolution process and showed that the actual processing of entities and their attributes is achievable only through pairwise comparison. As will see later in 4.1 this is also the most computationally expensive step in the whole identity resolution process. If we have a set of millions of entities then the pairwise comparison will challenge the overall process performance and success. Therefore we are introducing a reduction step - **candidates selection** - that selects a small subset from all entities using more effective algorithms and only the elements within the set will be compared.

This technique is also known as blocking (or creating "buckets") in databases where one uses a key for selecting a block of records. For instance, if the first character of the record field is selective enough, then the entire data set can be partitioned into 26 "buckets" and the search will be reduces 26-fold (assuming an equal number of members in each bucket); another possible restriction, in address records for example, is blocking by postcode.

There are different strategies for applying key restriction. One option is to initially process all objects and pre-create a key for each of them and then store the key values into a suitable data structure , e.g., hash-table or an inverted index. Then all objects with the same or similar key will form a block of comparable entities that can be efficiently retrieved. The second option is to create the key at run time, which means each time the retrieval algorithm will calculate the key for all entities. This approach works well for simple keys where one uses one entity attribute as a key and the entity storage - the semantic repository - maintains an index over the attribute values anyway. The main disadvantage here is if one uses complex keys that have to be re-calculated constantly, because this is computationally expensive. The first approach where all entities are pre-processed requires some time for initialisation, however the retrieval time will be reduced in comparison to the second approach. The main disadvantages here are the additional memory required for storing the new data structure in the case of huge number of entities; and the need for recalculating the index each time a new object is added to the set.

We reuse the notion of key for selecting similar entities from a pool of objects and extend the idea of how such key can be build from entity attributes. As the entity description consists of various properties and relations, their values will also be used for forming a blocking key.

For example a $key_1()$ function for entities of class *"Person"* as described on page 56 can be defined as the value of *hasBirthDate* attribute:

$$key_{per}(e_{person}) = e_{person}.hasBirthDate$$

Then the value for the entity with *example#Person.007* identifier will be

$$key_{per}(example\#Person.007) = \text{“}1952 - 01''.$$

The exact definition of the key function $key()$ depends on two aspects of the entity:

- the entity class which holds the description of all possible properties and relations of the entities. Since the key depends on one or more particular attributes, its definition is connected with their specification. However the class hierarchy reduces the number of keys that have to be specified, because the classes inherit the attributes from their superclass. For example, if classes “Man” and “Woman” are subclasses of “Person”, then all keys defined for “Person” will be also applicable to their instances.
- the domain in which the key is used - selection of the right key is crucial for the blocking process, therefore its definition depends very much on the domain of the identity application as well as the nature of the data sets. For example, the property *hasName* may be a good key in data sets with strict standards about the data collection, e.g., national security system, or to be very ambiguous for an enterprise customer relationship management system.

Once we have a key definition, then we can precalculate all their values if we have chosen this approach, or simply move to the retrieval phase. For each object that is currently passed through the identity resolution process, we select a block of entities in the target set that may be similar to it. To do that one can use different blocking strategies and some of them are described later in this section. However they all share a common understanding that the retrieval function $filter()$ takes an entity, calculates the key value and compares it to the keys of all elements in the target set. Finally it selects those elements in which keys are similar to the current one, using a predefined logic. Formally a $filter()$ function that uses one key per class of entities (e.g., Person) can be defined as:

$$filter(e_{person}, key_{per}) \rightarrow T' = \{e_i \in T : key_{per}(e_{person}) = key_{per}(e_i)\} \tag{3.1}$$

Selecting the most appropriate key may be very difficult especially for very noisy data sets. The best selection should maximise the precision and the recall of the retrieval process. Both metrics are standard evaluation metrics in information retrieval and are defined [10] as following:

[10] `http://en.wikipedia.org/wiki/Precision_%28information_retrieval%29`

- Precision - the number of relevant documents retrieved by a search divided by the total number of documents retrieved by that search - this shows the exactness of the retrieval. Instead of relevant documents, we are interested in similar entities. However at the retrieval step we are not calculating the exact similarity, but it will be performed at a later step, therefore maximising the precision is not always achievable.

- Recall - the number of relevant documents retrieved by a search divided by the total number of existing relevant documents (which should have been retrieved) - this shows the completeness of the retrieval. Again one can easily relate this definition to retrieving entities and here we are interested in retrieving all entities that are similar to the current one.

Maximising the precision will reduce the complexity of the overall identity resolution, lowering the number of pairwise comparison only to candidates that have certain similarity. It will not affect the accuracy of the overall process since only non-matching pairs will be ignored. The recall however plays a significant role in the entire process. The main obstacle here would be to omit candidates that are potentially matching, thus reducing the accuracy of the identity process. Therefore selecting the right key for blocking candidates into potentially matching and non-matching is essential.

Since often the number of the entity that are subject of identification is unknown (e.g. entities extracted from the web), the recall can be measured only over a pre-collected subset of all potentially available entities. The corpus is then manually annotated and the efficiency of the candidate selection is evaluated only for those entities that are available in the corpus. If one can not use any other characteristics of the entities during their collection, one can assume that the larger the corpus size is, the better representativeness it has.

To overcome some of the limitations of using a single key for blocking we introduce a $filter()$ function that is executed over a complex key definition. The complex key is defined by the logical connectives "$\vee$", "$\wedge$", "$\neg$" and "$\Rightarrow$" using keys defined over entity attributes. For example a complex key $key_c$ over an entity $e_i$ is defined as follows:

$$key_c(e_i) = key_1(e_i) \wedge key_2(e_i) \vee key_3(e_i). \tag{3.2}$$

Then the $filter()$ function will interpret the complexity of the key transforming the

logical connectivities to set operations:

$$filter(e, key_1 \vee key_2) = filter(e, key_1) \cup filter(e, key_2) \tag{3.3}$$

$$filter(e, key_1 \wedge key_2) = filter(e, key_1) \cap filter(e, key_2) \tag{3.4}$$

$$filter(e, \neg key_2) = \top \setminus filter(e, key_2) \tag{3.5}$$

(3.6)
$$filter(e, key_1 \Rightarrow key_2) = filter(e, \neg key_1 \vee key_2) = \top \setminus filter(e, key_1) \cup filter(e, key_2)$$

Following the above example 3.2 of a complex key, and using the definitions of its application on $filter()$ function, we can now elaborate the calculation of $function(e, key_c)$ using its definition over an attribute key:

$$\left.\begin{array}{l} filter(e_i, key_1) = T_1 \\ filter(e_i, key_2) = T_2 \\ filter(e_i, key_3) = T_3 \end{array}\right\| \Rightarrow filter(e_i, key_c) = T_1 \cap T_2 \cup T_3$$

As shown above the selection of keys is crucial for the quality of retrieval and there are several factors that can be monitored during the selection stage. Since the key will be built using the values of one or more attributes, the quality of the values should be taken into account. Therefore while choosing appropriate attributes one should examine the values of the attributes. Possible problems may arise from:

- variations in the attributes - they will result in a record to be inserted into the wrong block, thus missing true matches. Some of the variations can be resolved at this stage through data preparation and normalisation process described below. However, others are not trivial and are subject to more complex comparison algorithms which are computationally expensive and do not fit to the goal of the blocking stage. Here we aim at resolving only the most simple variations or errors without spending much effort on it.
- missing values of the attribute creating indefinite keys - then all entities with missing keys can be either always included, or always excluded from the retrieved set. Both options, however, do not suit the goal of the candidate selection step - artificially enlarging the candidate set will lead to many unnecessary pairwise comparisons; and leaving always a certain part of the candidates pool outside the selection will decrease the quality of the identity resolution process.

- frequency distribution of the values in the attributes used for blocking keys affecting the size of the block - the problem here arises only when selecting very large blocks where size could not be managed by the retrieval algorithm.

There is not much one can do about missing values, since this is the data source that has control over what is collected and what is omitted. However one can lower the effects from attribute variation going through one intermediate step that unifies the values - a data preparation step.

### 3.2.1 Data Preparation

As noted already the entity attributes can hold the following type of values: string, numeric or reference to other objects. While numbers and objects are well specified by type, string values can have a wide range of variations. They are also the most used type for describing real world objects as people, addresses, locations, companies, etc. While populating a data set automatically or manually, most of the data providers apply certain rules for standardisation of free text fields, however when dealing with heterogeneous sources one cannot rely on the fact that aligned attributes from different data sets will share a common data format. Therefore the attributes values are passed though data preparation stage, where variations of one and the same values are resolved.

Preparation rules depend on the type of the attribute and may vary from date records, names or address holding attributes. Therefore they are defined per type or class of type and consist of the following elements:

#### Parsing Strategy

Parsing or decomposition aims at discovering the building blocks of the value, these are the elements that the attribute consist of and can be separated, reordered or normalised without changing its values. For example a date record “2010-04-20 22:40:03” holds the same value as “20/04/2010 22:40:03” and both can be decomposed to the following elements:

year: 2010; month: 04; day: 20
hours: 22; minutes: 40; seconds: 03

Parsing rules have to be defined separately for a wide range of attributes: e.g., organisation names with company suffix like “Ltd.”, “Limited” and “PLC”, person names consisting of first name, surname and other names; addressed build from

street name, building name or number, postcode, etc. However, with very few exceptions, e.g., date, they are domain dependent, therefore hardly portable. They reflect the nature of the data and parsing for example Spanish names may differ significantly from parsing Hungarian names.

Parsing may be unnecessary if the attributes hold atomic values that cannot be decomposed, e.g., first person name only, or if attributes are represented as relations to other entities.

### Normalisation

The main goal of data preparation is to translate all values to a common format that can be easy to interpret later. Therefore one has to define unification rules for each type of attributes. They should describe the order of the building elements as well as the normalisation form for the element with spelling variations. Without normalisation the attribute values could be wrongly classified as not matching because they are not significantly similar. Very often the same values can be decided dissimilar if they use different:

- **unit system** - if the attributes represent measure with respect to a given unit system. For example weight can be described in kilograms, grams, or pounds; time can be presented in milliseconds or hours and minutes.
- **coding system** - both string and number values can be used with respect to a given coding system. The coding system is strictly defined per source and when merging several sources, one should explicitly describe the coding system they use. The coding system defines the string representation for possible attribute values , e.g., "M/F" for gender and numbers, or names for months ( e.g Apr. $\Rightarrow$ 04).

  While there are generally accepted coding systems as month's names, others are specially defined for the data set domain. For example a system collecting job offers in UK may code the country as 1/0 for UK-based and abroad, while another system applied worldwide can use the country abbreviation, e.g., "UK", "USA", "BG". This is an example where not only the values have to be translated, but also the granularity of the attribute values change. Then normalising them may involve using other attributes as well, e.g., the exact location, city of the vacancy, in order to extract the corresponding country code and unify the values per attribute country.
- **spelling variation** - This is a big group of problems with using different spelling for one and the same name. It includes (i) abbreviation - as in the

example given above the company suffix may be abbreviated (e.g., "PLC" and "Ltd.") or written in full (e.g., "Limited"); (ii) geographical and historical valuations, e.g., "John ⇒ JOHAN (Germany)"; (iii) using nicknames, e.g.,"Maggie ⇒ Margaret"; (iv) misspelling, e.g.,"Margret ⇒ Margaret". Wide range of string comparison techniques (see 4.1) can be used for resolving spelling variations, however they are computationally expensive and do not provide a normal form for the attributes. Therefore they are used during the actual entity comparison in the identification process. At this step we are aiming at partially normalising attribute values to facilitate their further processing. The proposed solution is to use a catalogue of name variations and replace them with a standard form. Example of such catalogue is ODM (Ordinance Data Management) catalogue developed since about 1969 consisting of 20 regional catalogs (North America, British Isles, Norway, Central America, etc.)[Wilson 05] that is build manually and maps name to different standards , e.g.,

Maggie, Peggy, Margaret ⇒ MARGARET - for all regions
John ⇒ JOHAN (GE) - for Germany,
John ⇒JOHN (NA) - for North America

At the end of the preparation stage the attributes are normalised and can be used further for building a selection key. So far we have ensured that variations in their representation will not result in different keys for the same values. This step is also called data cleaning or attribute-level reconciliation [Gu *et al.* 03] in the database community which faces the same type of problems. Then we move to the retrieval stage and there are various strategies for implementing the $filter()$ function.

### 3.2.2 Retrieval Strategies

#### Standard Blocking

The standard retrieval strategy is to select all entities that share a common key. Then the $filter()$ function will look like the one shown in example 3.1, where the entities will be selected only if their key values are equal. It can be efficiently realised using inverted index [Witten *et al.* 94] which makes it a preferred strategy when one needs a simple and quick method for blocking. The major drawback of this approach is that the variations or errors in the values of the attributes selected for building the key have direct impact on the retrieval performance. Another limitation is the lack of mechanism for sizing the result set. Still one can experiment with the selection of keys and overcome these disadvantages.

### Sorted Neighbourhood

A modification of the standard strategy is the sorted neighbourhood approach [Hernandez *et al.* 95]. The keys are pre-generated, then sorted and finally stored into an inverted index. Then the algorithm selects entities with not only precisely equal keys, but also with similar keys in a window of size $w$ around the exact match. It relies on the assumption that duplicated records will be placed close to each other in the sorted list, therefore retrieved and compared at the next stage. The generated set will be superset of the one retrieved by the standard strategy, which reduces the risk of possibly matching entities to be left outside the set. The largest window size will dominate in performance, but still it is highly dependent on the selection of keys, as well as on the sorting algorithm chosen for building the index.

The sorting process assumes that the attribute values are pre-processed, otherwise keys for two names "christina" and "kristina" will very likely be too far from each other in the sorted list. Therefore the authors of this strategy propose to run several independent runs on different sorting key and a relatively small window each time. This strategy is called a multi-pass approach where each independent execution produces a set of entities that can be merged. The resulting set then consists of all entities obtained during the passes. The main disadvantage is that the recall drops, because false positives are propagated across the subsequent passes.

### Priority Queue

[Monge & Elkan 97] propose an extension of the sorted neighbourhood method where the records are initially clustered. Then only representative entities from each cluster are considered and their keys are added in a sorted list. The leading assumption here is that identity resolution is transitive. This means that if $e_1$ is a duplicate of $e_2$ and $e_2$ is a duplicate of $e_3$ then $e_1$ and $e_3$ are also duplicates. In this way connected entities build a transitive closure of matching records and only a representative entity of the cluster is kept for subsequent comparison. This lowers the total number of comparisons without reducing the performance. The authors actually increase the complexity of the algorithm by introducing two new steps - clustering and selection of a representative element from a cluster. However it corresponds to diminished needs for sorting and searching in a big list, which will improve the performance of the method in very large data sets.

### Rejection Rules

[Neiling & Muller 01] aim at quick computation of the retrieval therefore instead of using similarity of a key they define a set of *rejection rules*. The intuition behind this

approach is that almost all entity pairs can be classified as non-matching through a simple computation. They extract the rules from a training sample where each entity is classified as *not_same* or *same*. Then the rules are looked up in clusters that contain :

- mostly *not_same* elements
- number of *not_same* element above the average
- only a small number of *same*-labeled elements.

The main disadvantage of this method is the fact that errors in clustering may be passed to the rule extraction process. The heuristics of how the rules will be defined also creates a room for reducing the performance of the method. If used in combination with other methods, it may significantly increase the speed of processing.

### Canopy clustering

The idea behind this technique is again to use computationally cheap similarity measure to select subset of entities on which to perform more expensive comparison measures. [McCallum *et al.* 00] propose construction of high-dimensional clusters, called *canopies* for extracting similar entities. The canopy functions are intended to be used as a "quick-and-dirty" approximation and [Cohen & Richman 02] propose *TF.IDF* similarity metric as a canopy distance. For building keys for each entity, the chosen attributes are converted into a set of tokens and then inserted into an inverted index. Then the *TF.IDF* is calculated and attached to each entity. Thus the key holds the distance between the entities in the pool and the currently processed entity.

At the retrieval step all entities that are closer to the currently being processed one - with key similarity above a threshold - are inserted into the canopy. There are two thresholds defined - more restrictive $\theta_{tight}$; and less restrictive $\theta_{loose}$ where $\theta_{tight} \geq \theta_{loose}$. Then all entities with similarity above the loose threshold are added to the canopy, and those that are also above the tight threshold are removed from the entity pool. Formally each canopy starts from an empty set $C_e = \emptyset$ and a target pool of entities $T$. Then

$$C_e = \forall e_i \in T(key(e_i) \geq \theta_{loose}) \tag{3.7}$$

At this step one can build a canopy for each newly processed entity at run time. However the authors propose initial customisation of the entire entity pool. This

is an iterative process that starts from a randomly selected entity from the pool and creates a canopy around it, as defined in (3.7). Then the pool is reduced by all entities in the canopy that are above the tight threshold.

$$T = T \setminus \forall e_i \in C_e(key(e_i) \geq \theta_{tight}) \tag{3.8}$$

A new randomly selected element from the rest of the pool is used for forming the next canopy and the process continues until there are elements in the pool. Finally all canopies will be formed and because of the different values of $\theta_{tight}$ and $\theta_{loose}$ they may overlap (see Figure 3.8). The $filter()$ function of an entity from the pool will then retrieve all other elements of all canopies where this entity is placed. If both $\theta_{tight} = \theta_{loose} = 1.0$ the canopy clustering becomes the same as standard blocking. Finally the $filter()$ function will retrieve all elements of the canopy where this entity appears.

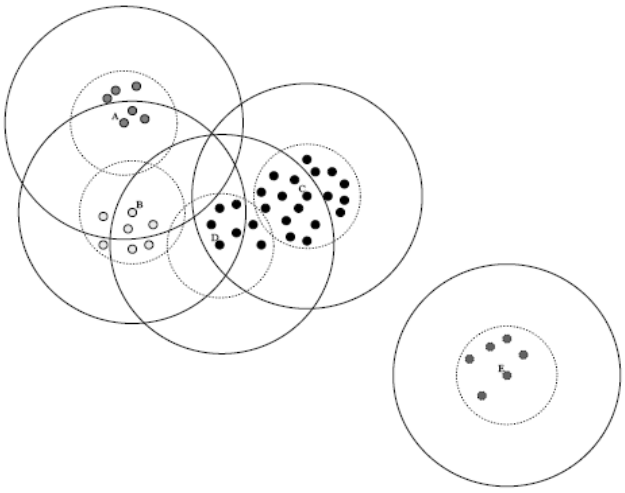

Figure 3.8: Canopy clusters

For the purpose of the usecase described in the next section we have implemented "standard blocking" retrieval strategy as part of the Identity Resolution Framework presented in Chapter 5. As part of the implementation the notion of selection key was extended to support complex key definitions.

A good overview and comparison of some of the presented techniques as well as other blocking techniques for record linkage is given in [Christen 07] and [Elmagarmid *et al.* 07].

### 3.2.3 Use-case Candidates Selection

In Section 3.1.3 we have aligned the original schema used in our usecases to an appropriate ontology. Based on the created mapping, then each object is transformed into an entity with attributes of the corresponding types. Here we move to the next stage - data pre-processing - starting from parsing and normalisation of attribute values; and defining the blocking strategy and key values for candidate selection. We will show how the above described techniques have been applied to each use-case data source: organisation profiles collection and job offers collection.

#### Job offers Collection

In this use-case we extract vacancies from corporate web sites and collect them in a data set, where the vacancies are presented with their entity descriptions built with respect to the system ontology as discussed in Section 3.1.3. Each time a new vacancy is extracted from the web, it is compared to the previously discovered entities in the set. If a duplicate is identified, the new information is merged, or otherwise a new vacancy is added to the set. Following the example of Figure 3.4 the initial entity description of this vacancy will be formally presented as follows:

```
<joci#Vacancy.15893>  <rdf#type>   <joci#Vacancy>
<joci#Vacancy.15893>  <joci#hasOrganisation> <joci#Company.zonal>
<joci#Vacancy.15893>  <joci#hasJobTitle> "Developer (C++)"
<joci#Vacancy.15893>  <joci#hasReferenceNum> "34"
<joci#Vacancy.15893>  <joci#hasSalary>
                              "J25,000 - J31,000 p/a + Benefits"

<joci#Vacancy.15893>  <joci#hasDatePosted> "1st of April 2010"
<joci#Vacancy.15893>  <joci#hasJobType> <joci#JobType.permanent>
<joci#Vacancy.15893>  <joci#hasTownName> "Edinburgh"
```

The main goal of the candidate selection step in the process of identifying possible duplicates is to form a subset from all vacancies. The selected jobs should be those that may be relevant to the one that is currently processed. This pre-filtering is needed because of the big data volume accumulated during the system run. The number of all live vacancies in the system varies between 400K and 650K, to which about 25K unique jobs are added each day (see Figure 3.9).

**Vacancy attribute normalisation**

Looking at the entity description above one can notice few sources of problems that arise from the string values of the attributes: (i)references to other objects are not

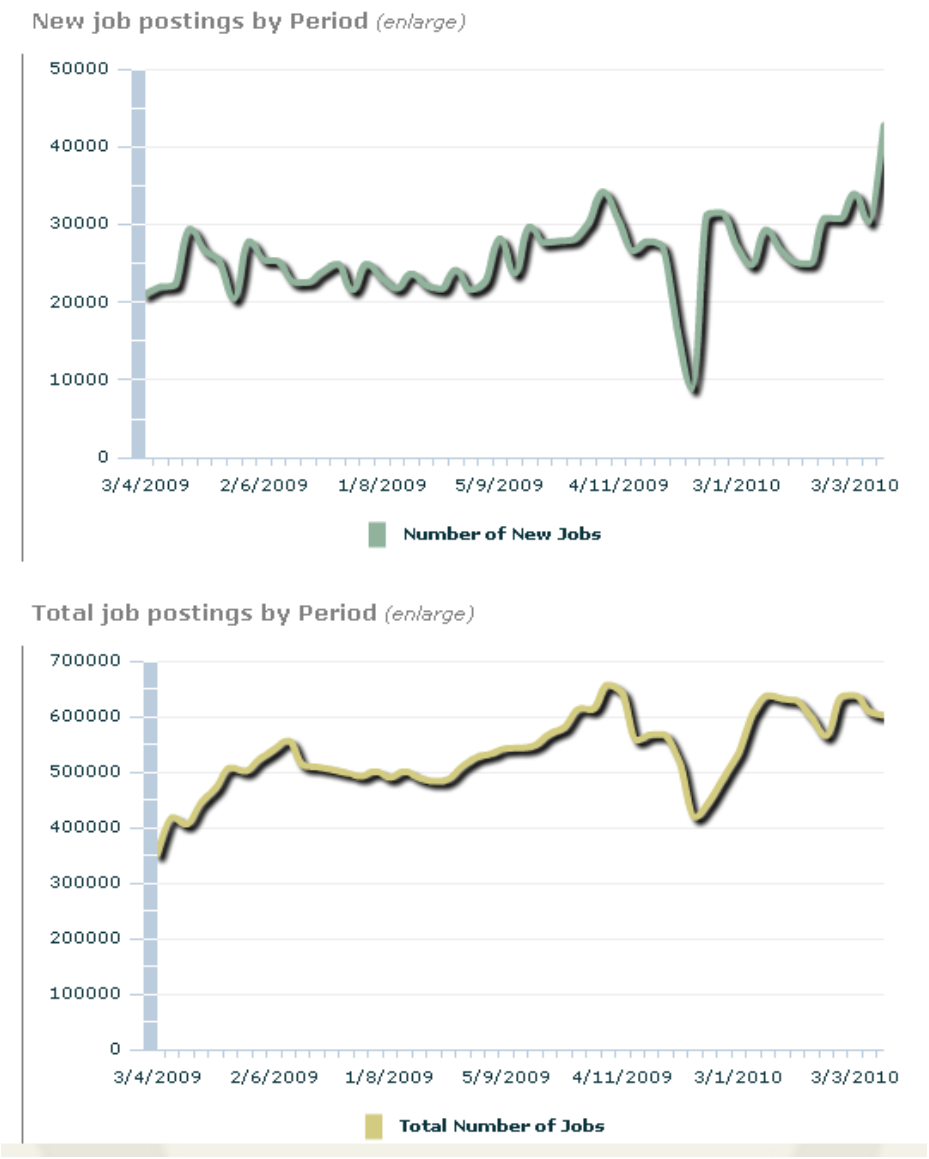


Figure 3.9: Statistics on the volume of live postings collected in the vacancy data set

resolved, e.g., the location; and (ii)numeric values appear as not normalised strings, e.g., salary. All these need to be sorted before searching for identical entities in the collected data set.

The vacancies are not the only type of entities that are accessible to the system, but there is a wide range of real world objects and concepts that are part of the system knowledge base. For example possible job types are modelled as concepts that each vacancy can refer to - e.g., *joci#JobType.permanent* stands for permanent jobs in contrast to *joci#JobType.contact*. The reference to a concept in a knowledge base eliminates the ambiguity in the string representations that may vary, e.g., contract type can be expressed as "Temporary ", "Contract", "Project". The type unification in our system is done during the vacancy extraction; however there are other attributes that require unification of their values in the resulting entity description , e.g., locations. The attribute *joci#hasTownName* contains the name of the town or the city where the job has been advertised, but it does not correspond to the knowledge about UK geography incorporated in the system. Therefore, these attributes are normalised and a new attribute pointing to the location concept in the knowledge base is created. Following the example above:

```
<joci#Vacancy.15893>  <joci#hasLocation> <CountryCapital_T.237>
```

Another type of problems arises from using numeric values as part of a string. In Section 3.2.1 we have outlined how the coding systems can be normalised. This mainly concerns using dates in this use-case, therefore we parse the date string to obtain its elements (e.g., "$1^{st}$ of April 2010" ⇒ "day=1", "month=4", "year=2010") and then rewrite it in the format we have chosen as native for the application "yyyy-mm-dd". Thus the value of the of the attribute will be changed:

```
from: <joci#Vacancy.15893>  <joci#hasDatePosted> "1st of April 2010"

to:   <joci#Vacancy.15893>  <joci#hasDatePosted> "2010-04-01"
```

More difficult for normalisation is the *joci#hasSalary* attribute. Here we start with parsing the value and extracting different attributes: minimal salary, maximum salary, currency, period, benefits, etc. For example the salary for vacancy *Vacancy.15893* initially presented as a string : "£25,000 - £31,000 p/a + Benefits" will be divided into

- "min=25,000",
- "max=31,000",

- "currency=GBP",
- "period=year",
- "benefits=yes".

Then the numbers in "min" and "max" split will be normalised so that they represent a yearly income. In our example, no conversions are needed, however the original numbers are often given per hour, or per month, therefore the necessary transformations to salary calculated on a yearly basis may apply. We have chosen GBP as native currency for the system as it dominates in the salary of vacancies we extract. Rarely other units may be used and in this case again the values should be transformed with respect to the corresponding exchange rate. Finally the original *Vacancy.15893* representation will be enriched with two more statements:

```
<joci#Vacancy.15893>  <joci#hasMinSalary> "25,000"

<joci#Vacancy.15893>  <joci#hasMaxSalary> "31,000"
```

**Candidate retrieval**

Once all attributes are normalised we are ready to proceed with candidate selection. Since the size of the target set is dynamic and processing a new vacancy often results in enlarging the candidate pool, using clustering techniques is not effective. Therefore we stick to the standard blocking approach, but we define a *complex key* over several vacancy attributes as defined on Page 72. The retrieval of the corresponding set of entities is then defined as a $filter()$ function (see Equations 3.3, 3.4, 3.5 and 3.6).

We have chosen three attributes for restricting the candidate selection:

- *joci#hasOrganisation* is the main restrictive attribute. The intuition behind using it is that in order to be the same, two vacant positions should be in the same organisation. If there is a job that is identical to the current one and it has been already collected it is one of the vacancies of this company. Although two job offers can advertise position with the same job title, salary, etc., if they are indicated to be in different companies, then there is no chance that they will be identical. An exception of this rule can be if the information of the company is missing, e.g., the job offer is posted by a recruitment agency, however the setup of the system ensures that only direct employer websites are processed. In case of a missing organisation this attribute should be ignored or substituted by (a combination of) other attributes.

$$key(Vacancy.15893, joci\#hasOrganisation) = joci\#Company.zonal$$

- *joci#hasReferenceNum* if present, is used as a complimentary restriction. It reduces the set of candidates from all adverts of the same company to only those that are referred by the organisation itself as identical. For example

$$key(Vacancy.15893, joci\#hasReferenceNum) = 43$$

- *joci#hasJobTitle* is also used for building a key for selection, however not the entire string of the name is used. There are certain parts of the title that are used in general to describe positions or professions, e.g., "Developer", "Engineer", "Account Manager", and we use them as a further selection requirement. To build the key for this attribute, we look up some domain keywords pre-collected in a set of job title databases. Then the keywords found in the vacancy title are considered as selection criterion at query time. For example, the key for *Vacancy.15893* the full value of *joci#hasJobTitle* attribute is "Developer (C++)" while the key will be:

$$key_{title}(Vacancy.15893) = "Developer"$$

The complex key is then defined as a combination of the above three attributes:

$$\begin{aligned} key_c(e_i) \quad = \quad & key(e_i, joci\#hasOrganisation) \wedge \\ & (key(e_i, joci\#hasReferenceNum) \vee \\ & key_{title}(e_i)) \end{aligned}$$

The filter function on each atomic key retrieves entities that have had selected value per corresponding attributes. The final candidate set then will be build from the individual retrieved sets using the operators defined in the complex key as shown below:

$$\begin{aligned} filter(e_i, key(e_i, joci\#hasOrganisation)) &= T_{org} \\ filter(e_i, key(e_i, joci\#hasReferenceNum)) &= T_{ref} \\ filter(e_i, key_{title}(e_i)) &= T_{title} \\ filter(e_i, key_c) &= T_{org} \cap (T_{ref} \cup T_{title}) \end{aligned}$$

### Company Profile Collection

In this use-case we have two data sets - one clean consisting of about 1.8M entities, and one duplicated set that is obtained and enlarged in real time from various

```
<musing#Organisation.2250547>  <rdf#type>   <musing#Organisation>
<musing#Organisation.2250547>  <musing#hasOrgId> "2250547"
<musing#Organisation.2250547>  <rdfs#label> "Marks & Spencer Plc."
<musing#Organisation.2250547>  <musing#hasType> "1"
<musing#Organisation.2250547>  <musing#hasSector> "3-14-650"
<musing#Organisation.2250547>  <musing#hasWebsite>
                                "http://www.marks-and-spencer.co.uk"
<musing#Organisation.2250547>  <musing#hasPhoneNumber> "02079354422"
<musing#Organisation.2250547>  <musing#hasAddress>
                               "Waterside House 35 North Wharf Road"
<musing#Organisation.2250547>  <musing#hasTownName> "London"
<musing#Organisation.2250547>  <musing#hasPostal> "W2 1NW"
```

Figure 3.10: Entity description of a company called "MARKS & SPENCER"

sources. The attributes of the entities in both data sets are preserved as they appear in the original source therefore differ in quality and formats. An example of an organisation record for "MARKS & SPENCER" as it is presented in the clean data set but transformed to RDF with respect to the description in section 3.1.3 is given on Figure 3.10.

**Organisation attributes normalisation**

Most of the attributes hold string values, which makes their processing not obvious. We start with normalising the locations that are already parsed and recorded as *musing#hasTownName*. The name of the town however may be ambiguous therefore we want to match it to a knowledge base of UK locations. The mapping will introduce a new value of a different attribute *musing#hasLocation* that will point to the URI of the location resource. Following the above example the string "London" will result in URI *CountryCapital_T.232* which further is known to be in relation *subregionOf* with another resource *County_T.15* that describes UK. Thus the new attribute will be:

```
<musing#Organisation.2250547> <musing#hasLocation>
                                        <CountryCapital_T.232>
```

One can define normalisation rules for each of the text attributes, e.g., phone number, address; however the most important for our application will be the organisation name. It is the attribute that is of crucial importance for the candidate selection as it is also the one of the most representative attributes for an organisation. For example in Figure 3.11 are shown some of the variations of how the name of "Marks & Spencer Plc." is presented in the automatically extracted entities from the web.

```
<musing#Organisation.w_03_987><rdfs#label> "Compass -Marks&Spencer"
<musing#Organisation.w_20_7>  <rdfs#label> "Marks and Spencer"
<musing#Organisation.w_04_3>  <rdfs#label> "MARKS & SPENCER GROUP"
<musing#Organisation.w_256_10><rdfs#label> "MARKS & SPENCER PLC"
<musing#Organisation.w_11_24> <rdfs#label>
                    "Symeonides Fashion House Ltd - Marks & Spencer"
```

Figure 3.11: Variations of the organisation name of "MARKS & SPENCER"

In order to normalise the organisation names we start with parsing. The goal of this process is to recognise the different elements of the name consists of. Then we will use different techniques for normalising each of these elements, depending on their nature. The three main elements the normalised name will consist of are:

- canonical name - the name that will be used for building the retrieval key;
- organisation suffix - e.g., PLC. - it will be needed during the pairwise comparison;
- site location - sometimes specified in the name.

We start with text segmentation also known as tokenisation that aims at dividing the whole name string into a set of meaningful substrings - words. Tokenisation is defined by several splitting rules that initially use the space character for splitting, however it is often omitted and other means for word separation are used instead , e.g., punctuation. Another examples is using letter capitalisation and numbers, e.g., "123RecruitmentLtd". At a preliminary step we recognise possible abbreviations roughly described as a consequence of upper letters that do not consist of vowels or do not match a common word. The offsets of the abbreviations are then also used for splitting, for example "BAIInternational" will be split to "BAI" and "International".

Once the name is divided into a set of words instead of a single string of characters, the parsing process continues by grouping different words to segments. These segments will become organisation name elements. Not all the words will be considered, however. The punctuation marks will be discarded and only "&" will be preserved; the determiner "the" at the beginning of the name, which does not play any significant role in identification process will also be ignored, e.g., "The Bank of England".

After a detailed investigation of the corpus of names obtained from both the clean and the web collected data sets we have noticed that certain words in the organisation name play similar role in building the name. For attaching meaning to these

words we use predefined lists of words grouped by type. One of the types is location appended to the organisation name that may vary depending on the site of the organisation which has been in focus in the original source. There are two ways of using locations

- pointing to the site of organisation - e.g., “RSD Group - London”
- using a location as part of the name of the organisation - e.g., “The College of North East London” or “Japanese England Insurance Brokers”

Therefore the location parsing strategy could not simply rely on lookup of gazetteer names, but also needs to apply strict rules for isolating the first usage from the second. For example, we considered only locations at the end of the name while those in any other position are found to be part of the name. Then if the location is connected with a preposition to the previous part of the name , e.g., “Church Of England” or “Homes For England Ltd” this will also be taken as a part of the name. As anywhere else where natural language labels are processed, locations can be ambiguous and it is difficult to decide whether the location is part of the name or it is not , e.g., “Visit London” vs. “NHS London”. It is out of the scope of the parsing process to resolve the ambiguity, therefore there is not enough evidence that a location points to a site of the organisation, it is kept as part of the name and used later in the identification process.

The next big group of words with an independent meaning is company suffixes. We have defined a set of abbreviations and full length company suffixes , e.g., “PLC”, “Limited”, “Group” that are recognised at the end of the organisation name. The string of the suffix is preserved in the organisation suffix element of the name.

Once the organisation name string is divided into three parts - words of the name, location and company suffix - the normalisation step will be needed. From the three types of normalisation given in section 3.2.1, we will apply a common coding system and filtering of spelling variations. The coding system concerns the locations - so far they are represented as strings, however the locations are represented as separate entities in the application knowledge base. Therefore the strings are translated to URIs of the corresponding entities in the same manner as *musing#hasTownName* attribute of the organisation is transformed to *musing#hasLocation*.

Although companies are registered with a single official name, third parties often use spelling variations as shown on Figure 3.11. We process the following types of variations:

- ordinal numbers - very often ordinal numbers are expressed by figures. To normalise them the digit containing strings are substituted by their full form

using a mapping table. For example “First Class Motors” is also referred as “1st Class Motors”.

- shortening using symbols - there are certain word forms that become popular and used to shorten organisation name. An example of such is the concatenation symbols - “&” and “+” are often used as a symbolic representation of the word “and”, or “@” used instead of ‘at”. Other example is using “.” while the original name contains “dot” , e.g., “Isleofman.Com ” vs. “Isleofman Dot Com Ltd”. Regardless the legal name of the organisation we decided to normalise the punctuation and other symbols that represent certain words with their literal spelling.
- abbreviations - the organisation domain uses a set of abbreviations, especially in organisation suffixes but not only , e.g., “srvs” instead of “services”. Those abbreviations are explicitly specified in a mapping list and substituted with their normal form.

After normalisation the entity on Figure 3.10 with name “Marks & Spencer Plc.” will be enriched with the following statements:

```
<musing#Organisation.2250547>  <musing#hasCanonicalName>
                                          "Marks and Spencer"
<musing#Organisation.2250547>  <musing#hasOrgSuffix>
                                     "Public Limited Company"
```

**Candidate retrieval**

Although we have been inspired by the way canopy clustering uses *TF.IDF* measure, the retrieval strategy that was implemented in this use-case is based on standard blocking , i.e., looking for exact matches of key values. The main argument for not using the canopies was the nature of the data sets. Our main data set is static, however the other one is dynamically collected from the web and its size is unknown at the beginning. Once the crawler identifies a new web page, the extraction algorithm described in Section 3.1.3 creates a new object that is passed to the identity resolution and the entity corresponding to an extracted company profile is built at run time. Thus the set of entities that need to be processed has no fixed size, which is one of the requirements for forming canopies. Therefore all retrieval algorithms that rely on pre-clustering are not suitable for this particular use-case.

The idea of approximate matching or maximum allowed distance in canopies, however, motivates us to extend the notion of a key in standard blocking with defining

possible key variations. Then each element, instead of being represented as an atomic key value, is matched to a *complex key* as defined on Page 72. The retrieval of the corresponding block of entities follows the definition of Equations 3.3, 3.4, 3.5 and 3.6, where $filter()$ on an atomic key executes exact match on the key values as required by the standard blocking.

Formally each atomic key is built from the normalised values of an entity of type *musing#Organisation*. After investigating the static data set and about 100K companies from the web extracted corpus, we have chosen three attributes as being the most descriptive:

- *musing#hasWebsite* appeared to be one of the restrictive attributes. It increases the precision of the overall blocking showing that all entities with the same web address are identical, with the exception of some errors in the way the value of this attribute is extracted from the web. The actual extraction however is outside the scope of this work.
- *musing#hasPostal* was fond to be distinctive for companies that have similar names, however placed in different locations. An important note in using this attribute concerns the site offices of the same organisation. They are represented as entities of *musing#Organisation* type as well and in contrast to the headquarter, they refer to the main organisation entity with *musing#isOfficeOf* relation. Therefore the attribute *musing#hasPostal* is not restricted to the head office only and an organisation with a site office with a given postcode will also be retrieved.
- *musing#hasCanonicalName* attribute values are the basic source for selecting similar candidates. The intuition behind it is that companies with completely different names could not be identical, therefore a set of keys is based on variations of words appearing in the name of the organisation.

One can notice that attributes, e.g., *musing#hasOrgSuffix* have not been used for building a key. Each of these attributes was carefully examined first and the reasons for excluding attributed from the selection fail into two groups:

- missing values for a significant number of the entities. An example of such attribute is *musing#hasOrgSuffix*, because the suffix usually mentioned at the end of an organisation name is often omitted when cited on the web. While the entities that need to be identified in this use-case actually come from the web, their missing attributes values will result in missing restrictions in the entity key. Then this attribute, having an empty value, will not be used in the retrieval query, which makes the calculation of this part of the key for all target entities in the rest of the data set pointless.

- inequitable distribution of an attribute value over the data, e.g., *musing#hasType*. This attribute represents the type of the organisation, e.g., government, educational, commercial, NGO, and selecting each of these values will result in very different size of the set. For example, selecting the code used for commercial organisation will result in millions entities, therefore will artificially increase the size of the selected candidates.

The above described two groups of concerns correspond to two of the problems in selecting key attributed presented earlier on Page 73. The third problem of attributes selection is caused by possible variations which will mostly effect the selection using *musing#hasCanonicalName*. We address it with building several keys - one per each variation. Differences in spelling organisation names often originate from the intention of quick typing. There are several cases where people can easily resolve the ambiguity in spelling , e.g., using the "+" sign that can stand for the word "plus" or "and" , e.g., "One+one Ltd " vs. "Radley + Co. Ltd", however this decision is very hard to be implemented in a software system. Since the quality of an automatic disambiguation process is questionable, we propose a work around where we form all possible interpretations assuming that only the correct one will match the correctly spelled name. The same strategy is used also for variations that are almost impossible to be correctly resolved even manually , e.g., the digit "2" standing for "to" and "two" or "too" in "Sharp 2 Ltd". Other spelling variations in spelling may arise from using space characters in abbreviations, e.g., "N H B S LTD".

Once we have selected the attributes, we can proceed with a complex key that we will use later in the retrieval. It is formally defined as a formula using the atomic keys of each of the attributes.

$$\begin{aligned} key_{org}(e_i) &= key(e_i, musing\#hasWebsite) \vee \\ &\quad (key(e_i, musing\#hasPostal) \wedge \\ &\quad key(e_i, musing\#hasCanonicalName)) \end{aligned}$$

The calculation of keys for *musing#hasWebsite* and *musing#hasPostal* are defined as strict match of the values as follows:

$$\begin{aligned} key(e_i, musing\#hasWebsite) &= musing\#hasWebsite \\ key(e_i, musing\#hasPostal) &= musing\#hasPostal \end{aligned}$$

The $key(e_i, musing\#hasCanonicalName)$ key, however, is again built as a complex

key over all variations of the organisation name:

$$\begin{aligned} varation(e_i, musing\#hasCanonicalName) &= \{v_1, v_2 ... v_k\} \\ key(e_i, musing\#hasCanonicalName) &= key_{name}(v_1) \vee \\ & \quad key_{name}(v_2) \vee \ldots \\ & \quad key_{name}(v_k) \end{aligned}$$

The names are really a special kind of values that are not flat strings but an ordered list of tokens. At this stage however one does not need to define a complex similarity measure on this data structure, but a simple selection algorithm. Therefore we chose a subset of the tokens that need to be present regardless their order in the selected entities. The key build on that set can be easily defined again as a complex key over the token values.

$$\begin{aligned} tokenise(v_i) &= \{t_1, t_2 ... t_m\} \\ key_{token}(t_1) &= t_1 \\ key_{name}(v_i) &= key_{token}(t_1) \wedge \\ & \quad key_{token}(t_2) \wedge \ldots \\ & \quad key_{token}(t_m) \end{aligned}$$

The algorithm that plays the most significant role here is the one that is used for the $tokenise(v_i)$ function. If one chooses more tokens, then the result set will contain fewer entities, and if one selects all tokens then only exact matches will be retrieved. Our goal however is to obtain a larger set of possibilities, that will not exclude possible deviations from the searched string, e.g., spelling errors. Therefore we want to extract a subset of all tokens that are the most representative for the organisation name.

To achieve this goal and motivated by the canopy clustering algorithm we use *IDF* index. It is build over the tokens extracted from company names in our static data set using the algorithm for *TF.IDF* measure in Section 4.1.2. As mentioned earlier it consists of almost 2M names and claims to hold almost all legal entities in UK. Based on these facts we assume that this data set contains a representative name set for UK organisations. Another reason to build our metrics on the static data set is that it is already cleaned - does not contain duplicates, therefore it will serve as an initial target for identification, i.e., the entity candidates will be selected from there. For each token in a name variation we calculate its *TF.IDF* measure and then select only tokens that pass an heuristically defined threshold.

Finally the organisation key is evaluated though selecting entities with the exact

```
  select DISTINCT
  V1
from
  {V1} <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            {<http://ontotext.com/2007/07/musing#Organisation>};
       [<http://ontotext.com/2007/07/joci#hasURL> {V2}];
       [<http://www.w3.org/2000/01/rdf-schema#label> {V3}];
       [<http://www.w3.org/2000/01/rdf-schema#label> {V4}];
       [<http://ontotext.com/2007/07/musing#hasPostal> {V5}]
where
  (V2 = "http://www.marksandspencer.com") or
  ( (   ( (V3 like "*marks" IGNORE CASE )
           or (V3 like "*marks *" IGNORE CASE )
        ) and
        ( (V4 like "*spencer" IGNORE CASE )
           or (V4 like "*spencer *" IGNORE CASE )
        )
    )
    and (V5 = "W2 1NW")
   )
```

Figure 3.12: Example of SeRQL query for a selecting candidates similar to *musing#Organisation.2250547*

match restrictions on several values. In this use-case the target entities are stored in a semantic repository which provides SeRQL query access to the data. The key then will be translated into a SeRQL statement like the one on Figure 3.12 and the result set of its execution will be the used for forming candidate pairs that will be passed to the next similarity measure step in the identity resolution process. More details about the implementation of "a key to query" are given in Section 5.3.

# Chapter 4

# Similarity Measure and Data Fusion

As discussed in the previous chapter we suggest dividing identity resolution in four steps. Once the two preliminary steps - schema alignment and candidate selection - are performed, one can proceed with measuring entities' similarity and their final fusion. The similarity measurement provides evidence for the identity of two objects as a step following the pre-selection. While the pre-selection uses simple techniques to roughly identify possibly similar entities in order to decreases the total amount of available object, the similarity measure step uses more sophisticated and accurate algorithms for denoting the level of similarity between two entities.

The final decision about the identity is taken in the data fusion step. Considering all the evidence collected during similarity measurement, in this step we choose the best candidate and fuse the new coming entity with it. The resulting updated entity details are stored back to the knowledge repository and made available for identifying new coming entities.

## 4.1 Similarity Measure

Similarity measure is the third step in the identity resolution process. Initially we have aligned different data representations of different sources, then we have selected a subset of all already identified objects and chosen objects that are eventually similar to the currently processed one, and now we can proceed with pairwise comparison. In order to decide whether the new coming object is identical to any of the already collected entities, we need some evidence of their similarity. Therefore, each entity in the candidate pool forms a candidate pair with the processed entity that will be evaluated. The evidence of their similarity is calculated by a $sim()$ function that will add an evidence score to the pair:

$$sim() : T, T' \Rightarrow R' = \{r \in R, 0 \leq r \leq 1\}$$

In this way each time a new entity $e_{cur}$ and a set of candidates $T'$ are passed to the similarity measure step, it results in building a set of triples $C$ so that

$$\forall e_i \in T'(C = \{e_{cur}, e_i, sim(e_{cur}, e_i)\})$$

The $sim()$ function of two entities is based on comparing their attributes. It formally represents the criteria for their identity, therefore it is crucial for the entire process. We describe these criteria following the semantic representation paradigm where the entity description holds all the details of a given object. Description attributes are: (i) properties and (ii) relations to other objects, and the criteria are designed to work on the attribute level. They may also make a significant use of the data that comes with the entity description like the class hierarchy in the ontology, the range of the properties, or some features of the relations , e.g., transitivity.

The evidence of similarity of two entities is more than a comparison of values. It holds the logic of the importance of different attributes and the implicit relations between them. It is not obvious how to combine the measures from different compare functions. Some authors propose various machine learning techniques that require big training sets, a good survey of which can be found in [Elmagarmid *et al.* 07]. However the main goal of this work is to outline the necessary steps in the identity resolution process, but not to advocate one particular implementation among the others. Furthermore building a training set of appropriate size for training a comparison model was not visible for the selected use-cases, thus we propose a rule based approach measuring pairwise similarity.

We define the $sim()$ function as a formula combining different criteria that are selected with respect to the entity class, the application domain, and the nature

of the data set. Therefore providing a universal $sim()$ function is not viable, but we propose a customisable definition that is based on different criteria. The $sim()$ function is therefore built from attribute comparison bits calculated by a $compare()$ function. The domain of the similarity criteria function $compare()$ are the set of new coming objects $T$; the corresponding candidate pool $T'$; and the set of possible types of attributes $A$. Like the $sim()$ function it results in a real value between 0 and 1:

$$compare() : T, T', A \Rightarrow R' = \{r \in R, 0 \leq r \leq 1\}$$

Similar to the $filter()$ function in Section 3.2 we introduce logical operators for combining criteria. In this setting the usual logical connectives are expressed as arithmetic expressions:

$$c_1 \vee c_2 \equiv c_1 + c_2 - c_1 * c_2 \tag{4.1}$$

$$c_1 \wedge c_2 \equiv c_1 * c_2 \tag{4.2}$$

$$\overline{c_1} \equiv 1 - c_1 \tag{4.3}$$

$$c_1 \Rightarrow c_2 \equiv \overline{c_1} \vee c_2 \equiv 1 - c_1 + c_1 * c_2 \tag{4.4}$$

The similarity function is then expressed as a combination of criteria, e.g.,

$$sim(e_1, e_2) = compare(e_1, e_2, hasName) \wedge compare(e_1, e_2, hasBirthPlace)$$

Similarity criteria can be application specific, e.g., working with a specific property or relation type; or domain independent predefined criteria , e.g., generic string comparison algorithms. We distinguish between two types of algorithms that implement the actual comparison between values of the selected type of attributes: (i) attribute specific algorithms; (ii) general algorithms.

**Attribute specific algorithms** reflect the nature of the attributes, e.g., person names. They allow certain variations of their values that do not result in different entities and can be further customised according to the specificity of compared object. These criteria are domain specific and usually correspond to particular phenomena in the data sets , e.g., using people nicknames. For example two values of person name attribute - “Matt” and “Matthew”- can be considered 90% similar.

Formally:

$$e_1\#hasName = \text{“}Matt''$$

$$e_2\#hasName = \text{“}Matthew''$$

$$compare_{names}(e_1, e_2, hasName) = 0.9$$

Another example for a specific comparison uses the relations between the values and measure the distance between the concepts in the graph. One can define the distance between two concepts following a given relation edge , e.g., “partOf” between a city that is part of a country:

$$City.London\#partOf = Country.UK$$

$$d_{partOf}(City.London, Country.UK) = 1$$

Then the comparison function between values of “hasBirthPlace” relation can be defined as following:

$$compare_{loc}(e_i, e_j, hasBirthPlace) = \frac{1}{1 + d_{partOf}(e_i, e_j)}$$

$$e_1\#hasBirthPlace = City.London$$

$$e_1\#hasBirthPlace = Country.UK$$

$$compare_{loc}(e_1, e_2, hasBirthPlace) = \frac{1}{1+1} = 0.5$$

**General algorithms** are applicable to all types of attributes and do not take into account the specific semantics of the values. Very intuitive type of a general criterion is the exact match of values, which results in boolean values 0 - not match; and 1 - match, and is applicable both to atomic values (e.g., string of numbers) as well as to relations.

Strong equivalence is such a general algorithm. It can be defined as following:

$$compare_{equal}(e_1, e_2, aType) = \begin{cases} 0, e_1.aType \neq e_2.aType \\ 1, e_1.aType = e_2.aType \end{cases}$$

For example a person being an employee of a company

$$e_1.isEmployeeOf = Organisation.256$$

$$e_2.isEmployeeOf = Organisation.256$$

$$e_3.isEmployeeOf = Organisation.89$$

$$compare_{equal}(e_1, e_2, isEmployeeOf) = 1$$

$$compare_{equal}(e_1, e_3, isEmployeeOf) = 0$$

The final choice of criteria that a particular system will apply is a composition of various individual comparison algorithms and should reflect the nature of entity properties and relations. In the following section we briefly present a number of both general and domain specific algorithms that work on different attribute levels. The list does not pretend to be exhaustive but to give general overview of some of the most popular techniques , e.g., string comparison metrics and to explore some ideas of how the other aspects of entity descriptions can be used.

### 4.1.1 Compare Relations

Relations can be seen as links from one entity to another and we can mainly compare the objects of the relations. They are usually represented as triples <s, rel, o> where *s* is the subject of the relation, *rel* is the relation type and *o* is the object of the relation. For example the relations of type "partOf" describe the nesting of entities of class Location. Then a possible relation of entity E1 is <E1,"sub region of", "UK">.

Exploring relations is one of the most promising parts of entity description criteria usage. It provides the opportunity to use relation properties (e.g., transitivity) as well as to explore complex constructions and relations chains. The links between entities build a graph where nodes are the entities and the relations are typified edges. This graph is directed and comparison of relations can be seen as searching for a path in the graph.

For example the graph on Figure 4.1 describe the following statements about the location relation of two entities: E1 and E2.

- E1 #hasBirthPlace "Country.UK"
- E2 #hasBirthPlace "City.Glasgow".

and the following relations between the location concepts:

- "City.Glasgow" #partOf "Country.Scotland",
- "Country.Scotland" #partOf "Country.UK".

The actual comparison of the objects of *#hasBirthPlace* relation will be tracking the path between *Country.Scotland* and *Country.UK*. The comparison algorithm can be specified to search for the shortest path regardless the type of the edges, or following

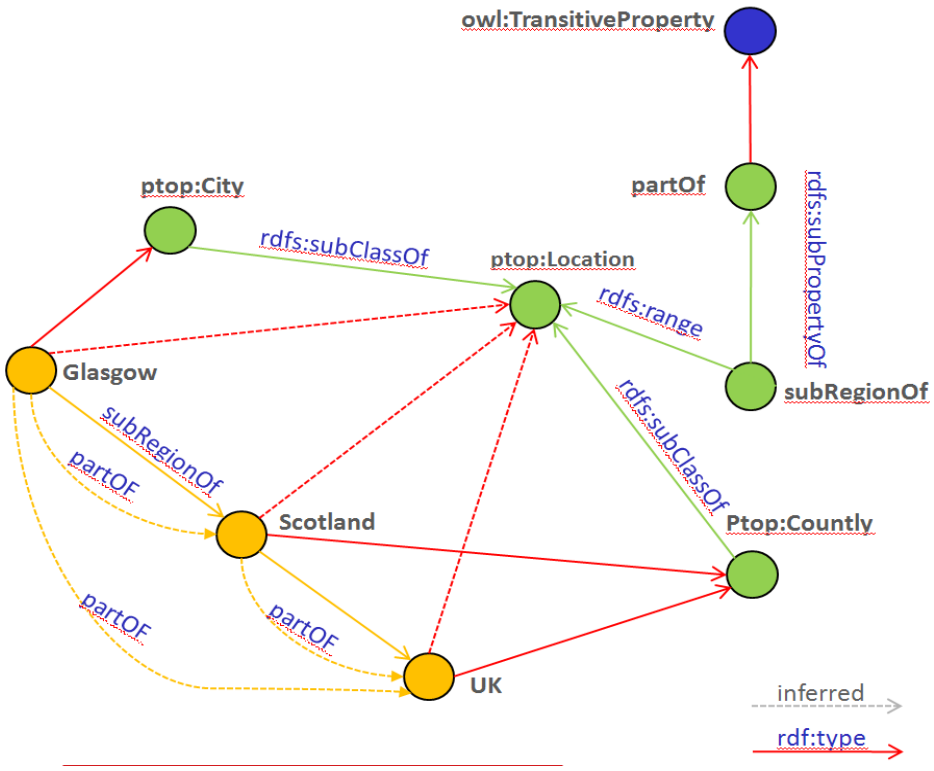


Figure 4.1: Part of a graph showing relations between Locations

only preselected type of edges. The benefit of using semantic representation is that the relations can be given certain characteristics , e.g., transitive, symmetric. Using these characteristics the graph of explicit relations will be enriched with implicit ones inferred by the corresponding rules, e.g., transitivity is defined as if xRy and yRz then xRz.

Following the example of Page 96 we are only interested in *#partOf* relations and we will measure the shortest path and normalise it using the same definition of the compare function, where $d_{partOf}()$ returns the length of the shortest path between two entities.

$$compare_{loc}(e_i, e_j, hasBirthPlace) = \frac{1}{1 + d_{partOf}(e_i, e_j)}$$

In Figure 4.1 the implicit relation between *Country.UK* and *City.Glasgow* is noted as a dotted line and it has been retrieved using the the transitivity of *#partOf*. Thus the minimal distance between *Country.UK* and *City.Glasgow* will be $d_{partOf}(e_i, e_j) = 1$.

Using the paradigm of graph representation for working with relations one can apply other graph algorithms, e.g., subgraphs comparison, distance to the lowest common ancestor. The rules for inferring implicit relations can also be customised with respect to the needs of the algorithms. As mentioned earlier the exact choice of an algorithm depends on the semantic of the relations types and the domain they are used in.

### 4.1.2 Compare Properties

Property values can be either strings or rarely numbers. Thus identity resolution based on property matching relies mainly on string comparison techniques. A lot of effort in this direction has been done in connection with database records deduplication and they deal primarily with typographical variations of names. These methods usually work well for particular types of errors or deviations in the string. Some of them exploit the sequence of characters the string consists of, while other divide the string into tokens and calculate string similarity based on the tokens that are found. Regardless the knowledge representation formalism we have chosen, string similarity metrics are also applicable for property comparison. Therefore we will shortly present some of the most popular ones, which serve as general criteria that depend only on the semantic description of the objects and are applicable to all entities. They are used for comparing attributes values without knowing anything about the nature of the attributes themselves.

#### Spelling similarity

**Edit distance** is one of the well know string comparison methods. It measures the minimum number of operation that are required in order to transform one string $s_1$ to another string $s_2$. It is designed to reflect and catch typographical errors, but typically fails on other type of errors. There are different metrics for measuring edit distance that vary in complexity and operation definitions. Levenshtein distance [Levenshtein 66] uses the following operators that have the same weight:

- *insert* a character to the string
- *delete* a character from the string
- *substitute* one character in the string by a different one.

For example the Levenshtein distance between “Schmidt” and “Smith” will be calculated as 3 deletions ( delete “c”; delete “h”; delete “d”) and 1 insert (insert

"h"), thus the overall distance is:

$$ld(Schmidt, Smith) = 4.$$

Computing the Levenshtein distance between two strings of length $|s_1|$ and $|s_2|$ takes $O(|s_1| * |s_2|)$.

Another metric for calculating edit distance is Hamming distance [Hamming 50]. The only allowed operator this metric is "*substitute*" and it can be calculated only between two strings of equal length.

**Sequence alignment** is a different distance measuring approach that gives different cost to different operators. It is primarily used in bioinformatics, where the names entities (e.g., genes and proteins) are long chains of characters. Similar to edit distance, metrics apply *insert*, *delete* and *substitute* there, but introduce "gap penalty" sores for mismatches in the original position of the characters in the strings.

- Affine gap distance uses two types of gap penalties: *gap opening* and *gap extension* penalty where the penalty of extending a gap is always smaller then opening a new gap - itA + (n-1)B, where A is cost of opening a gap, and B is cost of extending a gap. In this way a few big gaps are preferred while many small ones are discouraged.
- Needleman and Wunsch [Needleman & Wunsch 70] give also different costs to different substitutes according to a similarity matrix. It gives lower cost for substituting more similar characters and higher cost for dissimilar character. For example the cost between "O" and "0" may be smaller then "T" and "D".
- SmithWaterman algorithm [Smith & Waterman 81] is a variation of the previous one which allows for better local alignment of the strings. It searches for substrings of all possible lengths and chooses those that maximise the overall similarity measure. The complexity of this algorithm is O(mn) for two sequences of lengths m and n.

**Jaro distance** is a metric used extensively in record deduplication systems. It decides on the difference between two strings based on the number of matched and the number of misplaced characters. It is calculated by the following formula:

$$d_j(s_1, s_2) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m - \frac{t}{2}}{m}\right).$$

where $m$ is the number of matched characters and $t$ is the number of transpositions - the swap of two elements. This is a real number between 0 and 1 where 0 means not

matching and 1 means total match. For example the Jaro score for $s_1 =$ "$Schmidt''$ and $s_2 =$ "$Smith''$ will be:

$$d_j(Schmidt, Smith) = \frac{1}{3}\left(\frac{4}{7} + \frac{4}{5} + \frac{4 - \frac{0}{2}}{4}\right) = 0.79.$$

Winkler [Winkler 95] modified the Jaro metric adding more weight if the prefixes of the two strings match. They calculate the length of the common prefix substring. The modified formula is:

$$d_w(s_1, s_2) = d_j + l * p * (1 - d_j),$$

where $l$ is the length of the prefix, $p$ is a modification weight (normally $p = 0.1$) and $d_j$ is the Jaro distance between $s_1$ and $s_2$.

The **Soundex** metric is primary designed for matching person names in English. It is a phonetic algorithm that compares two names based on their pronunciation, thus allowing for different spelling. The phonetic coding scheme assigns the same digit to a group of letters with similar sound omitting vowels (which makes the metric unapplicable to Chines names with many vowels). The original code scheme is the following:

- b, f, p, v $\rightarrow$ 1
- c, g, j, k, q, s, x, z $\rightarrow$ 2
- d, t $\rightarrow$ 3
- l $\rightarrow$ 4
- m, n $\rightarrow$ 5
- r $\rightarrow$ 6
- h,w are ignored

The algorithm keeps the first letter, removes the vowels and then applies the code scheme for the rest of the letters replacing them with digits. Identical digits are consolidated and only the first one is kept. The final code of the string consists of the first letter of the original string and the following only three digits obtained after replacement. If there are less than three digits, it is completed with zeros. Finally, strings with equivalent codes are considered identical.

For example "Schmidt" and "Smith" get the same code:

- "Schmidt" = S-530 (S, C consolidated, H ignored, M = 5, I ignored, D = 3, T consolidated)
- "Smith" = S-530 (S, M = 5, I ignored, T = 3, H ignored)

Following the same idea of Soundex, Taft [Taft 70] proposes *New York State Identification and Intelligence System (NYSIIS)* algorithm with improved phonetic coding scheme. It differs from the previous one by allowing transformation of not only single letters but sets of characters , e.g., "MAC" ∈ "MCC" and "SCH ∈ "SSS". The author claimed about 2.7% improvement compared to Soundex.

Philips suggested another phonetic algorithm called *Metaphone* [Philips 90], which uses larger rule set compared to Soundex. Its successor *Double Metaphone* [Philips 00] enhances the rule set further and allows for a double key for ambiguous strings. Thus strings that share a common key are considered similar.

For example

- "Schmidt" will be described with (XMT, SMT)
- "Smith"will be described with (SM0, XMT).

However both share the "XMT" key, therefore they are considered identical.

### Complex string similarity

While the previous algorithms work on character level, there are a number of other metrics that use tokens instead of letters. They assume that each string is composed of one or more consequences of tokens divided by a separator, usually divided by space, number or punctuation. Each of the tokens can also be compared using plain string matching techniques, but here we are interested in combining the results for all individual tokens into a single metric.

Here we focus on vector based representation where strings are presented as vectors in the common token space. The attributes in the vector are the frequency of the corresponding tokens, which makes this metric also applicable to large texts. The particular order of the tokens does not play any role in the calculations, therefore this metrics is widely used for comparison of attributes without strict format. Example of such are the address records where the name of the street, the name of the building and its number, postcode, etc., can be listed in free order; or person name where first name, surname and title can be swapped.

In **cosine similarity** The metric is calculated as dot product of vectors representing two strings and normalised by the Euclidian norm. For two vectors $A$ and $B$ the cosine similarity will be:

$$similarity(A, B) = cos\theta = \frac{A.B}{||A||||B||}.$$

For example two strings "Smith, John" and "Mr. John Smith" can be represented in the three dimensional space formed by the tokens found in each of the records $T = \{Mr, John, Smith\}$ as $A = (0, 1, 1)$ and $B = (1, 1, 1)$. Then the cosine similarity is calculated as:

$$similarity(A, B) = cos\theta = \frac{0 * 1 + 1 * 1 + 1 * 1}{\sqrt{0^2 + 1^2 + 1^2} * \sqrt{1^2 + 1^2 + 1^2}} = \frac{2}{\sqrt{6}} = 0.8165.$$

Although quite similar the two names have only about 80% similarity which is caused by the title "Mr." in the second string.

This limitation of the cosine similarity metric is addressed in **Term Frequency - Inverse Document Frequency (TF-IDF)** weighting scheme. It is originally designed for retrieving documents similar to a particular query therefore tokens here are considered terms and whole string is called document. The motivation behind this metric is select a number of documents from a corpus that are relevant to a set of terms/query. The algorithm initially selects all documents that contain terms and then measures how relevant to the query they are. Terms are counted how often they appear in the document, usually normalised by the total number of tokens in the document. Each of the terms is also weighted by importance - how often it appears in documents in the corpus. In this way common tokens (e.g., titles in person name or articles in free text) will get lower weight.

Formally *Term Frequency (TF)* is calculated as total number $n_{ij}$ of mentions of term $t_i$ in document $d_j$ normalised by the total number of term $m$ in the document.

$$tf_{ij} = \frac{n_{ij}}{m}.$$

*Inverse Document Frequency (IDF)* is logarithm of the total number of documents in a corpus $D$ divided by the number of documents containing a term $t_i$

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}.$$

The metric is calculated separately per term and document

$$(tf - idf)_{ij} = tf_{ij} * idf_i$$

and then a ranking function combines the scores for individual terms. The simplest function only sums the scores for all terms per document to acquire the final ranking of the document in the selection.

*TF.IDF* metrics is generally applicable to all kinds of string collections and can be adopted for the identity resolution subtask of retrieving entities with similar attributes to a given one.

**Vector Space Model (VSM)** [Salton *et al.* 75] combines *cosine similarity* metric with *tf-idf*. In contract to the original cosine similarity where the vectors of the compared strings are built from the number of occurrences of the corresponding $n$ tokens, VSM assigns weights to tokens.

$$vsm(A,B) = \frac{A.B}{||A||||B||} = \frac{\sum_{i=1}^{n} w_{iA} * w_{iB}}{\sqrt{\sum_{i=1}^{n} w_{iA}^2} * \sqrt{\sum_{i=1}^{n} w_{iB}^2}},$$

where $A = (w_{1A}, w_{2A}..., w_{nA})$ and $w_{iA} = (tf - idf)_{iA}$.

### 4.1.3 Comparing Entity Context

We distinguish two types of criteria according to the input data we use. The entity description criteria are based only on the information about the entities regardless the particular context they appear in, and context criteria take into account the phenomena around the entity mentions. While the entity description is always present, the context it appear in may not be always available, e.g., if the entities are already collected and provided to the system as a set of isolated items.

So far we have discussed using the entity descriptions attributes, however the context the entity appear in may play significant role in their identity resolution. It contains implicit knowledge about entities that could now be formally retrieved as part of the entity description. Not all the sources provide context along with the entities, for example the nature of the database stores presumes lack of details that cannot be formalised as part of the corresponding record. Textual documents wherefrom several entity descriptions can be extracted are usually good source of contextual information, e.g., the news articles often mention a number of entities with a connection to a story, however the relations between them are difficult to be standardised and extracted.

#### Co-occurrence Criteria

The criteria that come intuitively directly from the context are based on appearances of different entities together within same contexts. The authors of [Popov *et al.* 08]

named this criteria co-occurrence, since the context is composed by the objects that co-occur with the given fact. It calculates the probability of a given entity to appear in the same context with the other entities. In order to calculate these criteria we are interested in all possible entities that the current mention can be identified with, in order to choose the entity with the highest score.

For example the new mention is "Cambridge" and the possible entities for identifications are "Cambridge @ UK" and "Cambridge @ US". Further, it appears in the context of "Cambridge University", so we calculate the co-occurrence of ["Cambridge @ UK" and "Cambridge University"] on one hand, and ["Cambridge @ US" and "Cambridge University"] on the other. Finally co-occurrence criteria will give better score to the entity that is more relevant in the context, i.e. "Cambridge @ UK".

In order to calculate co-occurring probabilities one needs a corpus of contexts. It can be either pre-available set or it can be also collected during the system run. For example, a very popular publicly available collection of documents is the Internet. One can use different search engines like Google and Yahoo! for tracking co-occurrence on the Web, where the simplest strategy is to query for documents where the chosen entities co-occur and count the hits. The great benefit of using the Internet is the enormous amount of available documents. It mainly covers topics related to people common knowledge like geography, politics, religion, science, social life, etc. However it may be poor or misleading in closed domains like any organisation's internal knowledge, e.g., medical treatments and clients' database. Other sources collected for various reasons may be available (e.g., DBLP [1] citation database), but how they can be processed highly depends on their data structure.

Another possibility for obtaining a corpus is to collect it during the system run. Initially the corpus is empty therefore if no other sources are available, the co-occurrence criterion cannot be used. However during the system run the corpus will be constantly filled with data, which can be then used for identifying new coming entities.

### Popularity Criteria

The popularity criteria like the Co-occurrence criteria are based on the assumption for context that the entities appear in [Popov *et al.* 08]. As the other context methods it serves as a model of background knowledge. Thus it is based on the already

[1] urlhttp://dblp.uni-trier.de/

processed contexts and relies on popularity trends for possible identification of entity mentions. It takes a certain meaning of an ambiguous mention and compares its distribution to all unification candidates in the accumulated contexts.

This aspect of multiplicity imitates human-like interpretation of the data. The assumption we make here is that: if the description of a certain fact is not detailed enough it refers to the reader background knowledge and other way round, if the reader is not expected to know about a certain fact or can be misled, a more detailed disambiguating information is provided. Popularity can be calculated based on various domain characteristics. It can start from very simple counting of news article a person appears in, to more complex scoring - tracking events related to an entity (e.g., movies where an actor stars in, sales growth for a stock, or publications and their citations for a scientist).

There are two ways of using the popularity according to the nature of the contexts, namely static and dynamic.

- **Static** - a preferred entity that corresponds to a given mention is defined explicitly, based on its highest popularity among the other candidate entities. These are pre-compiled values that are used statically.

  For example there are two entities that can be identified by the string "Paris", namely: "Paris @ France" and "Paris @ Texas". If "Paris @ France" is found to be more popular (e.g., in tourist adverts) than it will be preferred identify candidate for "Paris".

- **Dynamic** - popularity at a given moment - it is applicable in time spanned domains and takes a specific time segment to count the popularity of the identity candidates. It relies on the assumption that if one of the candidates is extracted significantly more often, this will indicate that this fact is related to an important event for this period. So we conclude with higher probability that the other sources reporting on the same topic will be about the same event, respectively will be pointing to the same entity.

  Although specification of the time segments could be a big challenge that goes beyond the scope of this work, there are domains where this criterion is easily applicable. These are domains that provide explicit time stamp of the context , e.g., scientific papers, news articles. An important note is that the required time stamp refers to the context where the facts appear and not to the facts themselves.

  For example in news domain - if we extract that most of the sources on a certain date report about Cambridge @ UK (let's say a big event at the University) and much less are about its competitor - Cambridge @ US - then

we can decide that the candidate (Cambridge @ X), which since now is equally possible to be attached to each of the competitors, should be identified with the most popular one at the current time segment - Cambridge @ UK.

Another example can be that - if something happened in "Paris, Texas" this week, then most mentions of "Paris" will be associated to "Paris, Texas".

### 4.1.4 Use-case Similarity Measure

In Section 3.2.3 we have shown how the data of the two use-cases has been preprocessed and prepared for future comparison. We have also described the mechanism for selecting potential candidates for identification. Here we will discuss in details the pairwise comparison of the two types of objects which are obtained from the selected sources: job offers and company profiles.

#### Job Offers Collection

The entities that are subject of identification in this use-case are of type *joci#Vacancy*. They are described using various properties and relations to entities of other types in the system knowledge base and a subset of these will be used for calculating their similarity. At this stage all attributes have been normalised as shown in Section 3.2.3. Furthermore the candidates per each new coming vacancy are already chosen and the process continues with pairwise comparison on entities. The retrieval key we have used in the corresponding $filter()$ function was based on three attributes namely *joci#hasOrganisation*, *joci#hasReferenceNum* and *joci#hasJobTitle*. The strong equivalence required for *joci#hasOrganisation* relations guaranties that both vacancy entities in the pair will have the same value for this attribute, therefore it will be excluded from the comparison itself.

The rest of the attributes we consider for measuring similarity between two vacancies are the following:

- *joci#hasJobTitle* - holds the name of the opened position. This is one of the most distinctive properties, however the advertisers tend to shorten the name when it is mentioned in a job listing and expose more descriptive name in the vacancy detailed page.
- *joci#hasKeyJobTitle* - is derivative form of the full job title that is obtained during the attribute normalisation. It usually presents the profession key , e.g., *engineer*, *developer*, *specialist*. As we will show later, it is primary used as controlling check when comparing full titles.

- *joci#hasReferenceNum* - is a string that holds an internal reference that the advertiser uses for tracking different positions
- *joci#hasLocation* - refers to other entities of type *joci#Location*. One vacancy can be advertised as placed in different locations, therefore there might be more than one relation between a *Vacancy* and other *Location* objects in the knowledge base.
- *joci#hasMinSalary* - a number corresponding to a currency amount that is advertised as salary for the current position, or a minimal salary if it is given as a range. The value is calculated during the data preparation phase there the string representation of the salary is parsed and normalised.
- *joci#hasMaxSalary* - a number corresponding to a currency amount that is advertised as maximum salary in a range. This property may have empty value if the salary is provided as a fix amount, not as a range.
- *joci#hasDatePosted* - a date in "yyyy-mm-dd" format that shows when the vacancy has been advertised for the first time. The original format of this data may differ from the system internal representation, therefore it is parsed and translated to the chosen format at the data preparation stage
- *joci#hasExpiryDate* - a date showing when the vacancy offer expires. It is often used as application submission deadline and if not present is calculated 30 days after the *joci#hasDatePosted*. It is normalised in the same way as *joci#hasDatePosted*.
- *joci#hasReportingTo* - is a job title and marks the position hierarchy in the organisation
- *joci#hasJobType* - shows the contract type of the position, e.g., permanent, temporary. The normalisation for this value is briefly given in Section 3.2.3
- *joci#hasJobStatus* - shows if it is a part time or full time position.

Each of the properties described above are used for building one of the criteria for similarity measurement. We will consider then one by one, outlining the specificity of the comparison algorithms starting from the *joci#hasJobTitle* property. As mentioned above the job title is one of the most restrictive attributes and does not allow many variations. Two vacancy titles are compared by measuring the percentage of the overlap between both strings. The criteria are calculated as number of common subsequent tokens excluding stop words, e.g., conjunctions, determiners, divided by the maximal total number of token in a title:

$$compare_{jobtitle}(e_i, e_j) = \frac{subsequentT(e_i, e_j)}{max(|e_i, e_j|)}$$

$$\begin{aligned} compare_{title}( \quad & "Structural, Civil, Consultant", \\ & "Structural, Civil, Engineer" \quad ) = \frac{2}{3} \\ & = 0.67 \end{aligned}$$

As one can notice on the example above the overlap method does not put any preference on which part of the job title will be shared. Therefore we use this metrics in a combination with comparison of *joci#hasKeyJobTitle* attribute. The strong equivalence (i.e., $compare_{equal}()$ given on Page 96) of the key words in the title will further approve or retract the similarity of two job titles.

$$\begin{aligned} compare_{title}(e_i, e_j) \quad = \quad & compare_{jobtitle}(e_i, e_j) \wedge \\ & compare_{equal}(e_i, e_j, joci\#hasKeyJobTitle) \end{aligned}$$

$$\begin{aligned} compare_{title}( \quad & "Structural, Civil, Consultant", \\ & "Structural, Civil, Engineer" \quad ) = 0.67 * 0 \\ & = 0 \end{aligned}$$

A different metric is used for comparing *joci#hasLocation* relations. As suggested in Section 4.1.1 entities and the relations between them are seen as a graph. The similarity function we apply in this use-case is based on the length of the shortest path between the two compared relation objects. As noted above one vacancy can be connected to several locations, which however are independent. If the vacancy is originally listed as located in two locations that are nested, meaning one is more specific than the other, e.g.,"London, UK", then only the most specific location is assigned to the vacancy. In terms of the graph representation, this means that path between locations related to one and the same vacancy does not exist.

The actual comparison of location relations start from searching for a path between any of the locations in the source set, to all locations associated with the target vacancy. If such a path exists, this means that the two locations are nested, therefore their similarity can be measured. We use a definition of the $compared_{loc}()$ function similar to the one given in Section 4.1.1, however applied directly to entities of type *joci#hasLocation*:

$$compare_{loc}(loc_i, loc_j) = \frac{1}{1 + d_{partOf}(loc_i, loc_j)}$$

If the relation of type *joci#hasLocation* of two vacancies probably "vacancy" or "vacant"

$$compare_{loc}(loc_i, loc_i) = \frac{1}{1 + d_{partOf}(loc_i, loc_i)} = \frac{1}{1+0} = 1$$

The overall criterion calculation will maximise the scores from individual location comparisons, minimising the length of the path between them.

$$compare_{loc}(e_1, e_2) = Max^{n}_{i=0,j=0}(compare_{loc}(loc_i, loc_j))$$

One of the important attributes of the vacancy entities is the salary, which is formally divided into *joci#hasMinSalary* and *joci#hasMaxSalary*. We are not interested in the currency of the money amounts, because it is normalised during the data preparation stage to reflect amounts in GBP. If the currency is not explicitly mentioned and since the job offers are collected from corporate web sites of UK based companies, we assume that it is the national monetary unit.

We distinguish between three cases of using salary range:

- both salaries are given as a single value - it will be collected in *joci#hasMinSalary* and the salary comparison will be limited to comparing two numbers. In this case we do not have a range therefore we can only rely on boolean equivalent values.

$$\text{if } e_1.hasMaxSalary = e_2.hasMaxSalary = null, \text{ then}$$

$$compare_{salary} = compare_{equal}(e_1, e_2, joci\#hasMinSalary)$$

- both salaries are given as a min-max range - alike the previous case there is no possibility for scalar measurement and the algorithm will accept only exact matches for both values in *joci#hasMinSalary* and *joci#hasMaxSalary*

$$\begin{aligned} compare_{salary} \quad = \quad & compare_{equal}(joci\#hasMinSalary) \wedge \\ & compare_{equal}(joci\#hasMaxSalary) \end{aligned}$$

- one of the salaries is given as a single value, while the other one is presented as a range - In this case we have to check for the possibilities. The single value may refer either to the minimal value or to the maximal value in the range. Even if the minimal values in the first salary is equivalent to either minimal or maximal value in the range, we consider this as less strong evidence of

similarity than the previously presented cases. Therefore we decrease the result of comparison by a coefficient $\alpha$.

$$\begin{aligned} \text{if } e_1.hasMaxSalary &= null, \text{ then} \\ compare_{sal}(e_1, e_2) &= \alpha \wedge \\ & compare_{equal}(e_1.hasMinSalary, e_2.hasMinSalary) \vee \\ & compare_{equal}(e_1.hasMinSalary, e_2.hasMaxSalary) \end{aligned}$$

For the rest of the attributes that are listed above as descriptive for entities of type *joci#Vacancy* we apply string equivalence metric. Thus the final similarity will be measured with the following $sim()$ function.

$$\begin{aligned} sim(e_1, e_2) &= compare_{title}(e_1, e_2) \wedge \\ & compare_{loc}(e_1, e_2) \wedge \\ & compare_{sal}(e_1, e_2) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasReferenceNum) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasDatePosted) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasExpiryDate) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasReportingTo) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasJobType) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasJobStatus) \end{aligned} \tag{4.5}$$

### Company Profile Collection

In this use-case we define a custom $sim()$ function for comparing company profiles. Due to the limited information coming from the extracted sources the choice of attribute is very restricted. The similarity criteria in this use-case are based on the attributes that were already selected during the data preparation and selection stage, and the argumentation of this choice is given in details in Section 3.2.3. The main reason for omitting the rest of the attributes is missing values and inequitable distribution. Thus we will discuss the comparison algorithms for the following entity properties for *musing#Organisation* class as defined in the application ontology:

- *musing#hasWebsite*

- *musing#hasPostal*
- *musing#hasCanonicalName*

Although using the same attributes as in pre-selection, there is a significant difference in the algorithms used for calculating similarity scores. During the previous run the selected attributes were used for building a query and the result is set of entities, while here the values are compared one to one and the result is a real number between 0 and 1 that reflects the level of similarity between the two.

We start with domain independent criteria for *musing#hasWebsite* and *musing#hasPostal* attributes. For those two we use strong equivalence which is one of the most general algorithms for value comparison $compare_{equal}()$ given on Page 96.

Processing of organisation names, however, requires more specific algorithms. Trivial equivalence here will lead to high precision, but very low recall due to the fact that one of the sources in this use-case are web sites which do not apply strict rules and allow organisation name variations in the information they present. For example "Perseus Management Consultancy Ltd" has been found as "Perseus Consultancy". Some of the variations, e.g., using company suffix, are filtered during the pre-processing stage, while others, e.g., missing words, should be handled during the actual comparison.

Another aspect of pre-processing is generation of different variations of organisation names, which are accessible as different values of *musing#hasCanonicalName* property. All these variations should be taken into account during the comparison and the maximum score will be selected as a final measurement. For example *musing#Organisation.w043* with label "MARKS & SPENCER GROUP" will be associated with three values:

- *musing#hasCanonicalName* = "marks and spencers"
- *musing#hasCanonicalName* = "marks & spencers"
- *musing#hasCanonicalName* = "marks spencers"

And for an organisation named on the Web as "People 1st" will be generated both:

- *musing#hasCanonicalName* = "People 1st"
- *musing#hasCanonicalName* = "People First"

In this use-case we propose using modification of two of the string similarity metrics that are described above, namely **Edit Distance** and **TF.TDF**. The final score is then derived from a weighted combination of both individually obtained scores.

We modify the way the overall edit distance is calculated to be an average edit distance per token. It is implemented as Levenshtein Distance [Levenshtein 66] normalised over the length of the longer string :

$$nED(s_1, s_2) = 1 - \frac{Levenshtein}{max(|s_1|, |s_2|)}$$

Initially the canonical names are divided into tokens, i.e., separate words, and then for each token we select the maximal score that can be achieved comparing it to each of the tokens in the target name. In this way the algorithm searches for the best match within the set of target tokens ignoring the words order in the organisation name. This results in high scores in cases where the words in the names are swapped, e.g., "University of Sheffield" v.s. "Sheffield University". In this example both strings will be tokenised and all stop words, e.g., prepositions, determiners, will be ignored, thus instead of two strings the input to the algorithm will be two sets of strings:

- "University of Sheffield" $\rightarrow$ {"university", "sheffield"},
- "Sheffield University" $\rightarrow$ {"sheffield", "university"}

Then each of the tokens in the source set will be given a similarity score compared to the tokens in the target set :

$$score(\text{“}university'') = max \begin{Bmatrix} nED(\text{“}university'', \text{“}sheffield''), \\ nED(\text{“}university'', \text{“}university'') \end{Bmatrix} = 1$$

$$score(\text{“}sheffield'') = max \begin{Bmatrix} nED(\text{“}sheffield'', \text{“}sheffield''), \\ nED(\text{“}sheffield'', \text{“}university'') \end{Bmatrix} = 1$$

The overall edit distance between two organisation names is calculates as average edit distance for each of the tokens in the source name. In this way the similarity between $e_1.musing\#hasCanonicalName$ = "University of Sheffield" and $e_2.musing\#hasCanonicalName$ ="Sheffield University" will be the following:

$$nED(e_1, e_2) = \frac{score(\text{“}university'') + score(\text{“}sheffield'')}{2} = 1$$

Although very efficient, *edit distance* can be very misleading where the deviation is found mainly in abbreviations or companies intentionally use spelling mistakes to create unique names.

- "D&S Personnel Ltd" v.s. "P&S Personnel"

- "GCS IT Recruitment" v.s. "GBCS It Recruitment Consultants"

Therefore we introduce another similarity criterion based on *TF.IDF*. The *IDF* index has been build over the static corpus of about 2M organisations names that serves as target data set in this use-case. This is the same index used already in the data retrieval stage described in Section 3.2.3, where all variations of the normalised organisation names are tokenised. All the variations of a organisation name are considered as a single document and the index is calculated as logarithm of the total number of organisations divided by number of organisations which name variation contains the given token.

Similar to the *edit distance* this metric is also calculated per each variation of an organisation name, and the maximal score is chosen. The variations are represented as set of tokens keeping their original order and initially we calculate *TF.IDF* score per each token in both target and source set. For example two token sets will be build for "GCS IT Recruitment Specialists" and "GBCS It Recruitment Consultants" respectively :

- $E_1 = \{$"gcs", "it", "recruitment", "specialists"$\}$
- $E_2 = \{$"gbcs", "it", "recruitment", "consultants"$\}$

*TF.IDF* score is then calculated per each token and the overall score per the name variation is composed taking into account the position of the token in the name. Our investigation on the corpus of organisation names showed that the words used in the beginning of the name are more distinguishing then those used at the end; the number of tokens in the name also plays significant role. For example the word "people" in the following two examples will have the same *TF.IDF* score, however has different importance for the name construction.

- In "Royal National Institute For Deaf People" one can drop the last word "people" and the organisation name will still have clear distinctive name
- "People Focus Europe" is an example where the same word "people" is inseparable part of the company name. "People Focus Europe"[2] is a human resource training company, while "Focus Europe"[3] is a technology company.

Another phenomena that can be tracked as word order is different meanings are homographs and homophones. However sense disambiguation is a task that goes over the scope of this work, and even if applied it may significantly increase the complexity of the calculations. Therefore we partially handle it using the words

[2] `http://www.peoplefocuseurope.co.uk/`
[3] `http://focus-europe.com/`

order, although we realise that this is very limited approach to the problem. For example the word "bee" in the sample below will be tracked differently because it is placed on different position in both names, however "it" will be given the same score regardless the particular meaning.

- "Bee IT"
- "Let It Bee"

To formalise the importance of the token positions in the name we introduce the following weighting function:

$$weight(t_{ij} \in E_j) = tf.idf_{ij} * \frac{|E_j| - i}{|E_j|}$$

Then the overall score for each set of tokens is calculated as sum of all weighted tokens scores:

$$tf.idf_{modified}(E_j) = \sum_{i=1}^{n} weight(t_{ij})$$

Similarity criterion is finally based on modified *TF.IDF* scores for both names that are compared. However it often appears (as in the examples above) that not all the words in the source string are also presented in the target string. Therefore we give penalty for the tokens that are not common for the two strings and decrease the total score by the percentage of shared tokens out of all unique tokens in both strings.

$$tf.idf(e_1, e_2) = max \left\{ \begin{matrix} [tf.idf_{modified}(E_1), \\ [tf.idf_{modified}(E_2) \end{matrix} \right\} * \frac{|E_1 \cap E_2|}{|E_1 \cup E_2|}$$

The main disadvantage of this metric is that *TF.IDF* tends to underestimate commonly used words within organisation names, which however clearly distinguish between companies. Therefore the pure string comparison is needed to balance the metrics and apart from rarely used words that hold the identity of the company. Such a commonly used term in our web crawled companies appeared to be "recruitment". It is therefore given lower importance weight in the name comparison and does not affect the overall matching score. For example:

- "Nexus Recruitment" will get high similarity score compared to "Nexus Institute"

- "Evolution Recruitment Solutions Ltd" will get high similarity score compared to "Evolution Mortgage Solutions Ltd"

Still we found that often repeating words in organisation names are not important for the comparison if they are not substituted by other words as in "Perseus Consultancy" v.s. "Perseus Management Consultancy Ltd" or the examples above. To assure this fact we use a weighted combination of *edit distance* and *TF.IDF* score using a constant $\beta$ :

$$compare_{combined}(e_1, e_2) = \beta * nED(e_1, e_2) + (1 - \beta) * tf.idf(e_1, e_2)$$

These criteria are calculated over all possible combinations of name variations that are available as *musing#hasCanonicalName* property values. We take the maximum over all combinations as final similarity score. In this way we formalise *compare*() function working with this specific property. In the evaluation presented in Section 6.2.2 we use a heuristically set value for the weighting constant $\beta = 0.3$.

The *sim*() function over entities of type *musing#Organisation* is defined similarly to the complex *key*() function in Section 3.2.3. It uses the logical operators for combining different criteria, but the result is a real number between 0 and 1.

$$\begin{aligned} sim_{org}(e_1, e_2) \quad = \quad & compare_{equal}(e_1, e_2, musing\#hasWebsite) \vee \\ & (compare_{equal}(e_1, e_2, musing\#hasPostal) \wedge \\ & compare_{combined}(e_1, e_2, musing\#hasCanonicalName)) \end{aligned}$$

## 4.2 Data Fusion

The last stage in the identity resolution is to fuse the entity candidates. This process includes making a decision whether two entities are identical or not as well as merging their attributes if they are found identical. At this stage the new coming entity has been already compared to several candidates selected from the pool of all already processed entities. The comparison score for each candidate pair reflects their similarity as a real number between 0 and 1.

In order to decide whether the processed entity is new for our data set, or it is identical to an entity that has already been stored we apply a similarity threshold $\theta$. We say that the entity is new if:

$$\forall e_i \in T'(sim(e, e_i) \leq \theta).$$

Then the new entities will be stored and made available for selection when the identity process starts for another object. In this way the information in the knowledge base is updated continuously and the information from the previous run is immediately available during the next round. The main advantage of this approach is that the data repository is kept consistent all the time and the process of adding new entities can be stopped and resumed at any time. This is especially valuable for realtime systems where new objects are non stop passed to the process and the results are needed immediately. There is a certain disadvantage of this approach, and this is the fact that the order of entities processing may influence the results. In this case one should consider different approaches , e.g., forming a transitive closure over duplicate pairs in order to obtain cluster of objects that may be identical to a given target entity. The nature of data sets we are using in this work is primary dynamic, therefore using such methods is not visible.

The second option is when one or more candidates pass the threshold which means that they are found similar to the new coming object. Only one of the candidates can be identical, because of the fact that the candidates from the already processed set are found different on a previous run. Thus we have to choose one of all similar candidates to be mention of the same object as the new coming entity. To do this we chose the most similar one , i.e., the candidate with the highest similarity score:

$$e \equiv e_k \text{ if } sim(e, e_k) > \theta \text{ and } \forall e_i \in T'(sim(e, e_k) \geq sim(e, e_i)).$$

Once the entity candidate has been chosen we proceed with merging the attributes values. Since the two entities will become one in the knowledge base they will be represented by a single entity description - a set of attribute/value pairs. Therefore

the attribute values of the target entity will be enriched with new values. Only values of the same type of attributes are compared, therefore the attributes are considered one by one. We distinguish between three cases of attribute merging:

- equivalence - all values of a given attribute for the source entity are equal to the values in the target entity. Then the resulting values will be the same as those in the target entity and there is no need for update.
- uncertainty - the values of the attribute either of the source or the target entity are null. Then we take the value that is present and attach it to the resulting entity description.
- contradiction - some values in the source entity are not identical to the values in the target entity. This case requires more complex techniques for handling compared to the previous two. Different strategies for discovery and resolving conflicts exist and which one will be used depends on the entity domain and the application goals.

### 4.2.1 Conflict Resolving Strategies

When the values of a certain attribute in the source and target entities are not identical, then one can apply different algorithms to resolve the conflicts. A detailed survey of strategies used in database records fusion is given in [Bleiholder & Naumann 06]. Here we will briefly outline some of the strategies and their possible application in merging entity descriptions.

#### Consider All Possibilities

One of the options is not to resolve the conflicts at all. This will result in adding all possible values from the source entity to the target entity. Thus the result set of the values of a given attribute will be joint of all available values:

$$\begin{aligned} merge() : E_1, E_2 &\rightarrow V = A_1 \cup A_2, \\ \text{where } A_i &= \{v_j : e_i.attrType = v_j\}, \text{ i.e.} \\ merge(e_1, e_2, attrType) &= e_1.attrType \vee e_2.attrType \end{aligned}$$

The ontology based knowledge representation allows multiple values per attribute type, therefore one can easily apply this strategy. However the main disadvantage will be, in case the attribute is used for similarity measure or candidate selection, that noisy values, or values containing errors may reflect to the quality of the overall

identity resolution process. For example if person name “Matthew” is given as nickname “Matt”, including this to the list of names will increase the similarity to other names which have the same nickname , e.g., “Mattias”. If the attribute plays an important role and its values need to be accurate some of the other strategies should be considered.

### No Gossiping

This strategy is the most conservative one and suggests removing of all values that are not equal. It originates from collecting consistent answers from queries over different data sets [Arenas *et al.* 99]. It efficiently separates conflicting from non-conflicting values and is used for filtering of noisy data. This means that the attribute values that are present in the target entity description, but are not found in the new coming entity should be removed from the updated entity.

$$\begin{aligned} merge() : E_1, E_2 &\rightarrow V = A_1 \cap A_2, \\ merge(e_1, e_2, attrType) &= e_1.attrType \wedge e_2.attrType \end{aligned}$$

This strategy is very useful when processing dirty datasets with a lot of errors. Then all values that are found in one of the datasets but not in the other will be cleaned and only values that are evident in both sources will be left as trusted. However when the datasets are seen as updates or complementary to each other, this approach may lead to fragmental instead of enriched data due to the incomplete nature of each of the source datasets.

### Trusted Source

The intuition behind this strategy is that one of the datasets comes from a trusted source. It prefers data coming from one of the sources over the data from the others. An example of a trusted source is manually collected corpus which is enriched with automatically collected data. Then the values of manually entered entities will be preferred among the others.

$$\begin{aligned} merge() : E_1, E_2 &\rightarrow V = A_1, \\ \text{where } A_1 &= \{v_j : e_1.attrType = v_j, e_1 \in E_1\}, \\ \text{and } E_1 &\ is \text{ trusted source} \\ merge(e_1, e_2, attrType) &= e_1.attrType \end{aligned}$$

The benefit of using this strategy is the fact that the trusted values are preserved

while in the same time the entity description is enriched with new attribute values. It is however difficult for implementation since during the system run the entities will obtain new attribute values which will not be originally in the trusted set. Therefore one should define a mechanism which identifies those values that were originally provided by the trusted sources and distinguish them from those coming from different datasets.

### Cry With The Wolves

Here the strategy relies on the fact that one of the contradicting values has been previously seen more times than the other. However the choice of the most common value is only possible if the entities are passed to the identity resolution process at once and not one by one. Then the transitive closure around the target entity will allow for comparison of several entities coming from different sources that can "vote" for the attribute values. Then it will be the decision of the majority which values will be the most representative and which ones are incorrect.

When the entities come one by one and the history of previous merges is missing, this strategy is inapplicable.

### Random Choice

Although it may sound not very sophisticated picking up one of the values randomly is a valid strategy. A more reasonable case of this unguided choice is to select the value that comes first. Thus the process order of the entities will result in preferring one or another value per an attribute. Since the processing order often depends on external factors which are difficult or impossible to be controlled, selecting the first coming value is effectively a random action from the application point of view. The main advantage of this strategy is that it resolves conflicts in computationally inexpensive way.

### Meet In The Middle

This strategy fabricated a new value that is close to both the source and the target values. It considers the fact that both datasets may be saying different parts of the truth and the intersection between them can be more correct then each of them separately. However this assumption is very restrictive and is applicable to limited types of attributes, therefore choosing this strategy should be made after a detailed investigation of the attribute values. Formally the new values are composed by an

*align*() function, the results of which are assigned to the entity attribute as a new value replacing the old ones.

$$\begin{aligned} merge() : E_1, E_2 &\rightarrow V = A, \\ \text{where } A &= \{v_j : align(e_1.attrType, e_2.attrType)\} \end{aligned}$$

For example if two company records say the range of employees is "20-50" and "10-35", then the middle value can be calculated as intersection of the two ranges , e.g.,

$$merge(e_1, e_2, numEmployees) = align(\text{"20-50"}, \text{"10-30"}) = \text{"20-35"}.$$

### Most Specific

A new strategy that can be derived from the ontology based knowledge representation is to choose the most specific value. This is possible in cases where the attribute values are connected with a certain relation in the ontology graph , e.g., "City.Glasgow" and "Country.UK" in Figure 4.1. Although formally the two values not equal, they can be seen as different granularity of one and the same value. Here we use the inference mechanism of the ontology representation and alike "Meet in the middle" the new value will be derived by a function over the relation characteristics.

$$\begin{aligned} merge() : E_1, E_2 &\rightarrow V = A, \\ \text{where } A &= \{v_j : specific(e_1.attrType, e_2.attrType)\} \\ specific(e_1.attrType, e_2.attrType) &= \begin{cases} e_1.attrType, \\ e_2.attrType \end{cases} \end{aligned}$$

The main difference from the previous strategy is that the algorithms select one of the already existing relation objects and does not invent a new value. One of the drawbacks of this strategy is that the $specific()$ function has to be defined per each type of relations with respect to the domain specificity and could not be applied to all cases. Further if there is no path in the relation graph between the two relation objects, then the conflict stays unresolved and another backup strategy should be considered. The main benefit of this approach is that it allows for continuous refinement of the attribute values.

#### Update

Very often one dataset is merged with another one in order to obtain more recent information about the entities. In this case the most appropriate strategy for resolving conflicts will be to select those attribute values which are updated more recently. The crucial requirement for applying this strategy is to have a time-stamp associated either with the entire entity or preferably directly with the attribute values. Example of an application that will benefit from this strategy is a web crawler collecting vehicle offers from dealers' websites. Although the newly found vehicles are easy to be identified with previously collected information (e.g., using their VIN number) the price of the vehicle may change. Then the value of the "price" attribute in the knowledge base will differ for the newly found one and this conflict will be resolved with selecting the most up-to-date value. Another example where the values are time-stamped is merging data from two Customer Relationship Management systems. Again the recently changed attributes , e.g., telephone number will be chosen for the resulting entity description.

### 4.2.2 Use-case Data Fusion

Data fusion is the last step of the identity resolution in our two use-cases. So far we have shown how the schemas are aligned and the data is normalised. Then we have discussed the candidate pre-selection criteria and how to obtain similarity score for each of the candidate pairs. Finally we will focus on making the decision whether the currently processed entity is new for the target dataset, or it is identical to any of the already processed objects. The most interesting part at this stage is the strategies for merging conflict values and their application in the presented use-cases.

#### Job offers Collection

The fusion process starts with a set of candidate pairs. The source entity is the new coming vacancy while the target one that is used for composing the pair, is selected from the pool of all already processed vacancies. After the similarity measure process described in Section 4.1.4 each pair is associated with a similarity score. Based on this score we decide on the uniqueness of the entity.

In this use-case we define the unique vacancies, those that will be added as new to the target dataset, as those that are totally dissimilar to any of the candidates. The intuition behind this definition comes from the similarity function shown as Equation 4.5 on Page 4.5. As one can notice most of the attributes are compared

with $compare_{equal}()$ function which has boolean values. Some of the other attributes are also used for marking dissimilar attributes, e.g., $compare_{title}$. Therefore if the two vacancies do not possess all evidences for similarity then the $sim()$ function will return 0, which we treat as total dissimilarity.

Formally we define the threshold $\theta = 0$ and since the range of the $sim()$ function is $\{0, 1\}$ the unique entities as :

$$e : \forall e_i \in T'(sim(e, e_i) = 0).$$

All the candidates that pass the threshold are considered for further fusion. However the candidates are found unique on a previous step therefore we have to select only one of them as identical to the new coming vacancy. We choose the one with the highest score and proceed with merging attribute values. Some of the attributes used in the $sim()$ will need no merging, since by definition they should be equal , e.g., *joci#hasReferenceNum* and *joci#hasDatePosted*. The values of others, e.g., *joci#hasLocation* relation, allow special handling:

- relations - all the conflicts in relation objects are treated following **Most Specific** strategy. However if there is no path in the relation graph we apply **Consider All Possibilities** and add all the available values to the target entity description. A case of a relation attribute is *joci#hasLocation*. It allows for multiple values and therefore can be safely enriched with links to other entities in the graph. For example:

$$\begin{aligned} & \text{if} \\ e_1.hasLocation &= \text{“}City.Glasgow''\text{ and} \\ e_2.hasLocation &= \text{“}Country.Scotland'', \\ & \text{then} \\ specific(\text{“}City.Glasgow'', \text{“}Country.Scotland'') &= \text{“}City.Glasgow'' \\ merge(e_1, e_2, joci\#hasLocation) &= \text{“}City.Glasgow'' \end{aligned}$$

  An exception of this approach is *joci#hasOrganisation* relation, which however is already used in pre-selection stage. Therefore the equality requirement for this attribute is placed at the first stage and one should not expect any conflicts in its values.

- salary range - The expected conflicts here come from using a salary range in one of the entity description and a single value in the other. Here we apply

a specific case of **Meet In The Middle** strategy. As result of the comparison function used for similarity measurement (presented on Page 4.1.4), the atomic value can be equal to either the minimal or to the maximal value in the range. Therefore the merging of both attributes *joci#hasMinSalary* and *joci#hasMinSalary* should be performed in one run. Thus we modify *Meet In The Middle* strategy to take into account both attribute values.

$$
\begin{aligned}
&\text{if} \\
e_1.hasMinSalary &= e_2.hasMinSalary, \\
&\text{then} \\
align(e_1.hasMinSalary, e_2.hasMinSalary) &= e_1.hasMinSalary, and \\
align(e_1.hasMaxSalary, null) &= e_1.hasMaxSalary,
\end{aligned}
$$

$$
\begin{aligned}
&\text{if} \\
e_1.hasMinSalary &< e_2.hasMinSalary, \text{ and} \\
e_1.hasMaxSalary &= e_2.hasMinSalary, \\
&\text{then} \\
align(e_1.hasMinSalary, e_2.hasMinSalary) &= e_1.hasMinSalary, and \\
align(e_1.hasMaxSalary, null) &= e_2.hasMinSalary.
\end{aligned}
$$

- job title - After the similarity comparison of the *joci#hasJobTitle* attribute the values that will be found similar are two strings sharing a sequence of common words. To resolve possible inconsistency in the data we apply **Most Specific** strategy, where instead of looking at the relations in the graph we compare the length of both strings. The title with the hight length is considered more specific and is attached to the resulting entity description. For example:

$$
\begin{aligned}
&\text{if} \\
e_1.hasJobTitle &= \text{“Structural and Civil Engineer” and} \\
e_2.hasJobTitle &= \text{“Consulting Structural and Civil Engineer”,} \\
&\text{then} \\
specific(e_1, e_2) &= max\_length(e_1.hasJobTitle, e_2.hasJobTitle) \\
merge(e_1, e_2, hasJobTitle) &= \text{“Consulting Structural and Civil Engineer”}
\end{aligned}
$$

For the attributes that do not require special handling we apply a modified version of **Trusted Source** strategy.

In this use-case the vacancies are collected from thousands of different web sites, which make the choice of trusted source inapplicable. Moreover the identical vacancies often come from the same source, e.g., crawled several times over a certain period. Therefore we adopt the notion of a trusted source as preferring the values in this entity description that holds more attributes. The intuition behind our formulation of a trusted source is that very often the vacancies are shortly presented in a listing page, where some of the details are omitted and others are presented briefly. A more elaborated description is provided in a separate job detailed page. It usually emphasises on the required skills and education level as well as on offered benefit packages. The vacancies that come from a job detailed page are extracted with more attributes then those in a listing page, and they are considered as trusted source in applying this strategy.

The advantage of using modified Trusted Source strategy can be easily seen in resolving conflicts in vacancy *joci#hasDescription* property. It contains a free text description of the opened position, which is very difficult for interpretation. Therefore we chose this value that comes with a vacancy possessing more attributes, assuming that more detailed formal description will reflect in more detailed textual description.

### Company Profile Collection

At this stage all possible candidates for identification with the newly extracted company are selected from the target dataset and their similarity is already calculated. The decision on whether the extracted profile defines a new company or refers to a previously extracted entity is made by applying a threshold. If all candidates are found similar to a score below the threshold, the extracted entity will be added to the knowledge base and it will be given a new URI.

In order to choose an optimal value we have used a Receiver Operating Characteristic (ROC) curve analysis. Following [Davis & Goadrich 06] proof that a curve dominates in ROC space if and only if it dominates in a Precision&Recall space, we use this method to select the best value for our company profiling threshold. As shown on Figure 4.2 the threshold of $\theta = 0.4$ gives the best performance and therefore balances the system's Precision and Recall. More details on the evaluation are given in Section 6.

Once the new facts are identified as not passing the threshold, the rest entities are fused with the best of the candidates , i.e., the candidate with the highest similarity

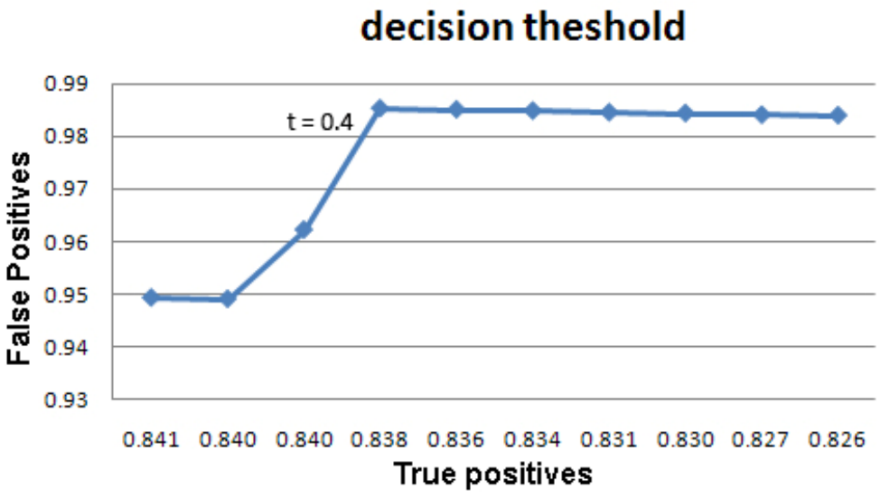


Figure 4.2: ROC curve analysis on company profiles

score. We distinguish between three types of attributes according to the fusion strategy that is applies:

- organisation description attributes - these are attributes that hold mostly static values and are often connected to the legal registration of the organisation, e.g., address of the headquarters. In contrast to the job offers collection, company profiles are obtained from two sources - the one is manually collected, while the other one is extracted from the Web. In this use-case we assume that the one of the sources is more reliable than the other because the quality of the entity details that are manually entered are higher than those that are automatically extracted. This assumption will play a significant role in resolving conflicts between rarely changing attribute values , e.g., company address, phone number. For these attributes we apply **Trusted Source** strategy.
- organisation name - As discussed earlier in Section 4.1.4 organisation names are free text and may vary depending on the source they are extracted from. The legal registration name is provided by the static source, however the same organisation may be referred with slightly different spelling, typos, etc. Once the identity of two entities representing the same company is resolved, then we collect all variations and use **Consider All Possibilities** strategy.
- organisation activity indicators - these are attributes that reflect the activity of the organisation , e.g., number of employees, stock exchange rates. The

main source of this information is the Web. As it is dynamically updated the organisation related facts that is automatically extracted is often more amended. Therefore we apply **Update** strategy for all attributes that have temporary values. The time stamp is acquired from the crawler indicating when the information is seen on a given webpage for the first time.

# Chapter 5

# Identity Resolution Framework

In the previous chapter of this thesis we introduced the theory for identity resolution as well as its application in two use-cases. To extend our contribution on the practical level we also provide an implementation of the proposed approach. The Identity Resolution Framework (IdRF) implements the general solution to the identity problem presented in Chapter 3. The main goal of the framework is to be flexible enough to fit in different applications and work with respect to their particular domain or objects which identity need to be resolved. As any other framework, the purpose of IdRF is to improve the efficiently of creating new systems. It is implemented as an object-oriented software library, but in contrast to other libraries, its objects and components can be customised while the control of their invocation follows the framework's internal logic.

There are several benefits to having an implemented framework performing the identity resolution task, which are similar to those of using frameworks in general:

- established implementation practice - the framework API provides design patterns for any new system that faces the identity problem. It clearly outlines the processing stages, the obligatory calculations and possibilities for customisation. This ensures that the process will be implemented correctly without omitting important stages and in the same time it provides the flexibility to meet the application specific requirements and domain constraints.
- code reuse - the main stages of the identity process are pre-built and tested, therefore the reliability of the new systems is increased and the development effort is reduced. This comes from the fact that the complex algorithms are

already implemented and the research phase of building a new system can start with executing experiments on the default framework setup, instead of building a software prototype from scratch. Furthermore the development phase is also reduced by combining existing algorithmic blocks instead of dealing with the more mundane low-level details, implementing and testing them.

- lay the basis for more complex processing - The identification process is often part of more complex scenarios for data processing and the proposed approach aims at solving only the core of the problem. Thus its implementation as a framework provides the means for extending its default behaviour and constructing more complex and case specific solutions.

One of the characteristics that make this framework a unique tool for identity resolution is that it is based on an ontology - used as both an internal and resulting knowledge representational formalism. The rich internal semantic representation extends the possibilities of the identity criteria to operate with the already discovered entities or the context of their appearance. The ontology finally presents the resulting objects with their full semantic description aggregated during the identity resolution process. Thus the outcome of the framework is a single integrated representation of all the particular descriptions of a given object that are identified.

Another important feature that improves the usability of the framework is that it allows the user/application to customise the identity criteria. The importance and respectively the weights of these criteria can be also set for a particular task, so to meet the specificity of the objects, the context they appear in, and the system's requirements. Thus, the support for customisable criteria and tunable weights helps IdRF to resolve identity for wide variety of object types for different applications that use information integrated from different sources.

The main goal of the IdRF is to find an appropriate place for each of the incoming objects within the target ontology. The incoming object is a candidate for an instance of the ontology and it is either a new instance or it is identical to some instance which already exists. So the framework aims at resolving the identity of incoming objects according to the ontology using different criteria. While the data may come from different sources, its integration is the main goal and desired achievement of the proposed framework architecture.

## 5.1 The IdRF Architecture

The IdRF architecture consists of three main data processing components that correspond to the last three main stages of the identification process discussed in Chapter 3. The processing (see Figure 5.1) starts from a new entity that is passed to the system. Then the pre-filtering component will retrieve all entity candidates from the target dataset and for each of them will build a new-coming/candidate pair. All the pairs will then be passed to the Evidence collection module that will calculate the similarity between the two objects in the pair. The results will be finally passed to the decision maker component which fuses the data and activates the entity storing procedure.

The framework's native data model is based on ontologies therefore the first stage of schema alignment should be performed in advance and the data passed as entity descriptions. The processed instances will be stored in the target knowledge base. The current implementation of IdRF is based on the PROTON [Terziev *et al.* 05] ontology, which is designed to be easily extendable to different domains or specific tasks. The native data store of the framework is OWLIM[1] [Kiryakov *et al.* 05], a repository built as a Storage and Inference Layer (SAIL) for the Sesame[2] RDF framework.

The backbone of the framework is the **Semantic Description Compatibility Engine (SDCE)** which is responsible for the mediation between the data processing component and the repository. It is presented in details in Section 5.3. It translates the pre-filtering restrictions into semantic queries and similarity rules into executable methods. SDCE is entirely based on the **Class Model** described later in Section 5.2 that plays the most significant role in the framework implementation and contains domain specific rules for identification. According to our knowledge representation, each entity is presented as belonging to a particular class in the ontology and it is also associated with a set of predefined properties and relations. Therefore, the set of entity description criteria can be retrieved based only on the entity description. In the proposed framework the criteria are coded as predicates and form rules/formulas that describe the whole class of entities. Then the interpretation of these rules is performed by the SDCE. The predicates may provide general purpose calculation or a specific measure and might be used only for a certain class model. For example, specific handling for alias similarity for the class "Person" deals with abbreviations of the first person name, while the alias similarity for "Organisation" class includes analysis of company suffixes. Thus both formulas describing both examples will use different predicates for alias comparison.

[1] `http://ontotext.com/owlim/index.html`
[2] `http://www.openrdf.org/`

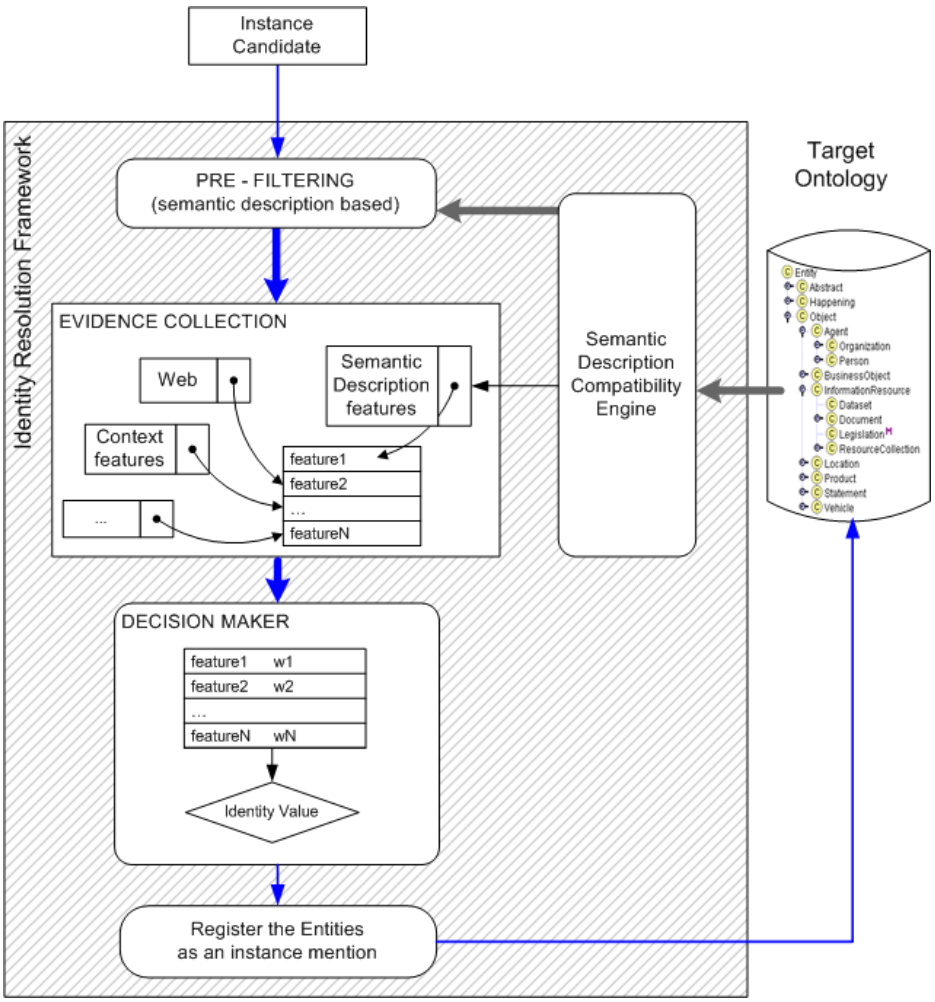


Figure 5.1: —The IdRF Architecture

The component corresponding to the *candidate selection* stage (see Section 3.2) is called **"Pre-filtering"**. It filters out the irrelevant part of the ontology and forms a smaller set of instances similar to the source object. Since the ontology can be full of data - irrelevant to the identification process, pre-filtering is intended to restrict the ontology instances to a reasonable small number, to which the source object will be compared. While the total number of facts stored in the ontology can be exceedingly big - millions or even billions, the identical object to the new coming fact is only one. Therefore, it may be more efficient to compare the fact only to those objects in the ontology that pass the initial criteria. It can be regarded as pre-selection of ontology objects that are eligible for identification. The selected instances are potential target instances that might be identical to the source object; they already appear in the knowledge base and are somehow similar to the source object. Pre-filtering is realised by the Semantic Description Compatibility Engine and applies strict criteria described in details later on (see Section 5.3).

The **"Evidence Collection"** component corresponds to the *similarity measure* stage of the identity resolution process described in details in Section 4.1. It aims to collect as much as possible evidence about the similarity between the source object and each of the identification targets in the ontology. The evidence comes from various identity criteria that can by either the default or custom. They express different aspects of the identity of the source object with each of the instances selected during the Pre-filtering stage. Simple weighting of evidences can often be the cheapest solution (both in terms of setting up and calculation), especially when dealing with well defined objects described in simple structures. Then the indicators are direct - based on the properties of the entity - and their priorities can be easily tuned by domain experts. They are natively stored as part of the Class Model described in 5.3. The default identity criterion is based on the semantic descriptions of the objects and it is calculated according to class definitions and the ontology data that are compared by the Semantic Description Compatibility Engine. Other custom criteria could be lexical distance of names of the objects, web appearance, context similarity, etc.

Once all the evidences for different identity possibilities are collected the **"Decision Maker"** concludes which is the best identity match. The Decision Maker is the fourth stage that decides on the strength of the presented evidence and whether it demonstrates identity between the source object and the target instance it is compared with. The final decision about the identity of a given fact is actually a choice between all possible candidates. It is based on the collected evidence about each of the candidates and decides which or none of them is identical to the currently processed object. Obviously, this decision is not trivial and it strongly depends on the entity type and the application domain. Once all identity evidence

is collected, they need to be ranked and combined in such a way that their relevance to the entity type and domain is preserved. For example, "birth date" can be a very strong indication of People identity, but "published date" is a questionable evidence for equivalence of job offers. Furthermore, this component fuses the new object with the identity candidate resolving all possible conflicts between the attribute values of the two entity descriptions.

After the decision is made, the incoming object is registered to the ontology as a final stage in the IdRF. The source object can be either new one or successfully identified with an existing instance. If the system is not able to find a reliable match, the incoming object is inserted as a new instance in the KB. In case it is associated with an existing instance, then the object description is added to the description of the identified KB instance. Thus, the result from the current identification is stored in the ontology and is used for further identity resolution of the next incoming objects. The integration of the newly processed data enriches the ontology adding either entirely new objects or only new attributes sources to the existing ones. In this way the ontology aggregates the description of all the incoming objects and provides a single view to the processed data.

As an effect of the constantly enlarging KB, the identity criteria are dynamically refined improving the identity resolution by both refining the evidence calculation and introducing new entities serving as identity goals. Details about the two effects are given below:

- The evidence calculation is refined when a new value, attribute, property or relation is added to an existing instance description. Then, the identity criteria for this instance is changed in order to reflect the newly available data adding new comparison restrictions. For example, if the person age is added to his/her description, the age restriction will be added as a new identity criterion.
- New identity goals are all the new entities that are added to the knowledge base. They are created by insertion of entirely new objects to the KB. Then the entities that are processed in a later stage have to be compared not only to the previously available entities but also to the newly added instances.

Each of the above stages is supported by a default implementation that is easy configurable for a new domain. SDCE requires access to the repository as well as a class model definition as described in the next section. The pre-filtering stage is based on a specific class model interpretation that is fully based on the provided formula, therefore does not require any other specific configuration. Evidence collection stage is also based on the class model formula, therefore dependant only on

the predicates used there. Default predicates include strong equality as well as basic string similarity measures e.g. Levenshtein distance, etc. Using of default implementation of the decision making phase requires setting up a decision threshold. It executes "Consider All Possibilities" conflict resolving strategy (see Section 4.2.1)

## 5.2 Class Model Definition

The hot spot in the IdRF is the definition of the Class Model. It describes the algorithms used for processing each type of entities that are passed for identification. It reflects the specificity of each class of objects as they are defined in the ontology, i.e., particular usage of their properties and relation during the identification process. One class model specifies a combination of algorithms needed for handling entities of the corresponding type and serves two types of customisation needs: on one side the model describes the restrictions needed for *candidate selection* stage; and on the other side, it expresses the criteria used by the *similarity measurement*.

Formally the class model is derived from a set of rules configured as formulas and express different conditions. Rule inheritance between classes is also supported allowing the set of formulas to be easily expanded for a new class. This is especially useful when the ontology is extended and refined or the focus of a particular IdRF application is changed. Basically the formulas are valid for entities of the same type or class and if two instances are from different classes then, the formula that is attached to the most specific class common for both of them is used. In case one of the classes is a subclass of the other, e.g.,

(5.1) class(C1)&class(C2)&subClassOf(C2,C1)

the formulas are composed by predicates from a common pool of predicates which are implemented as Java classes making them extensible. Each formula is composed by combining predicates by the usual logical connectives:

- "&",
- "|",
- "not" and
- "⇒".

```
namespace: rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns"
rdfs:"http://www.w3.org/2000/01/rdf-schema"
protons:"http://proton.semanticweb.org/2005/04/protons"
protonu:"http://proton.semanticweb.org/2005/04/protonu"
musing:"http://www.ontotext.com/2007/07/musing"


"protons:Entity":
    SameClass()

"protont:Company":
      let parentCond = Super()
          sectorCond =
                   StrictSameAttribute(<musing:hasWebsite>)
      in  parentCond \& sectorCond

"musing:Company":
      Super()|
      OrganizationCombine(<musing:hasCanonicalName>) &
      StrictSameAttribute(<musing:hasPostal>)
```

Figure 5.2: Example of a Class model definition for the "musing:Company" class

The example on Figure 5.2 shows the corresponding formulas for two classes the main class *protont:Company* and its subclass - application specific extension *musing:Company*. It is essential that several formulas can use one and the same predicate as part of their definitions (e.g, *StrictSameAttribute()* on Figure 5.2). The idea of defining a number of simple predicates instead of a single complex one follows the library like "code reuse" approach in software development. This allows us to support an extendable set of reusable primitive predicates from which someone can compose complex formulas in a declarative way.

The predicates are two-method algorithms and behave differently according to how the class models are used. In two stages of the framework pipeline: (i) during the retrieval of potential matching candidates from the ontology - they provide the strict criteria; and (ii) during the comparison of potential matching pairs they calculate soft similarity criteria. They implement a common Java interface - `Predicate` - with two public methods which are called depending on which component uses the model:

- `public Condition prepareQuery (EvalContext context,`
  `String targetVar,`
  `EntityDescription descr,`
  `boolean isImportant)`
- `public double eval (EvalContext context,`
  `EntityDescription descr1,`
  `EntityDescription descr2)`

The logical operators are presented as special predicates implementing the same interface, but take other predicates as constructor arguments. For example "&" corresponds to the following implementation (all system predicates are listed in Appendix A):

```
package com.ontotext.idrf.sdce.predicate;

import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;
import com.ontotext.kim.client.entity.EntityDescription;

public class AndPredicate implements Predicate {

  Predicate m1, m2;

    /**
     * system predicate corresponding to the logical operator 'and'
     * @param m1 - predicate 1
     * @param m2 - predicate 2
     */
    public AndPredicate(Predicate m1, Predicate m2) {
        this.m1 = m1;
        this.m2 = m2;
    }
     /**
     * combines similarity score from predicate 1 with score from
     * predicate 2 as d1*d2
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return similarity score
     */
```

```
  public double eval(EvalContext context,
                     EntityDescription descr1,
                     EntityDescription descr2){
        double d1 = m1.eval(context, descr1, descr2);
        double d2 = m2.eval(context, descr1, descr2);
        return d1 * d2;
  }
    /**
     * combines query restrictions from predicate 1 and predicate 2
     * in AndCondition
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
 public Condition prepareQuery(EvalContext context,
                               String targetVar,
                               EntityDescription descr,
                               boolean isImportant){

    Condition cond1 =
        m1.prepareQuery(context, targetVar, descr, isImportant);
    Condition cond2 =
        m2.prepareQuery(context, targetVar, descr, isImportant);
    if ( cond1 != null && cond2 != null )
        return new AndCondition(cond1, cond2);

    if ( cond1 != null ) return cond1;
    if ( cond2 != null ) return cond2;
    return null;
  }
}
```

The `Super()` predicate is a system predicate that refers to the class model of the direct parent of the given class. The example of Figure 5.2 illustrates the usage of `Super()` in the following hierarchy *Entity* $->$ *protont:Company* $->$ *musing:Company*. Used as part of the *musing:Company* class model, it refers to the formula of *protont:Company* where the same predicate is used for reference to the model of its superclass *Entity*.

Thus building the class model is reduced to the recursive call of `Predicate` interface implementations following the order that is given in the formula.

**Class Model For Candidate Selection**

The *candidate selection* corresponds to the second stage of the identity resolution given in Section 3.2. The role of the class model at this stage is to specify the criteria for selecting identity candidates from the pool of all available target entities. Therefore its formal definition - the formula - has the same structure as the complex key in the $filter()$ function, e.g., the one shown as Equation 3.2 on Page 72. However the atomic keys are presented as predicates which are responsible for calculating their values and again they are combined by logical operators.

For example the complex key we use for candidates retrieval in Company profile collection ( see Section 3.2.3) is described as three atomic keys over attributes *musing#hasWebsite*, *musing#hasPostal* and *musing#hasCanonicalName* of the entities of the corresponding class. Formally it was described as:

$$\begin{aligned} key_{org}(e_i) \quad = \quad & key(e_i, musing\#hasWebsite) \vee \\ & (key(e_i, musing\#hasPostal) \wedge \\ & key(e_i, musing\#hasCanonicalName)) \end{aligned}$$

Looking at the class model example of Figure 5.2 one can find one to one correspondence between the atomic keys and the model predicates:

- $key(e_i, musing\#hasWebsite)$ = StrictSameAttribute(<musing:hasWebsite>)
- $key(e_i, musing\#hasPostal)$ = StrictSameAttribute(<musing:hasPostal>)
- $key(e_i, musing\#hasCanonicalName)$ = OrganizationCombine(<musing:hasCanonicalName>)

Further these predicates are combined by the logical predicates `AndPredicate` corresponding to "&" and `OrPredicate` corresponding to "|" signs in the formula. The recursive calling over the predicates implementation refers to the `prepareQuery()` method. In this way the class model formula defines the $filter()$ function for a class of entities. Then the SDCE is responsible for applying the restrictions over the candidates' pool.

**Class Model For Similarity Measure**

One of the important steps in solving the identity resolution problem is to choose the appropriate criteria for comparing objects. Thus, the ability to customise the identity criteria is an important feature that leads to better maintenance of the

proposed solution. The relevance of these criteria should be decided according to the particular task domain, because they need to reflect the specificity of the objects, the context they appear in, and other system requirements. The presented software module comes with a number of build in comparison algorithms that can be used as predefined criteria. However the framework allows using custom implementations and definitions.

During the *similarity measure* stage (see Section 4.1), class models correspond to the $sim()$ function. It specifies the algorithms for attribute similarity measure as well as their combination. Each of the predicates used in the module takes an argument - entity description attribute - that will be compared. Final combination of single property or relation similarity is achieved like in the candidate selection scenario by using the logical operators. This allows using various algorithms for comparing different attributes. However the rule pairwise comparison is usually based on different attributes than the candidate selection, therefore the corresponding rule in the class model should be separately defined.

Predicates are interpreted as $compare()$ functions over the attribute values and the results are combined according to Definitions 4.1 to 4.4 on Page 95. The logical operators implement this definition as `eval()` methods of the corresponding Java classes. For example four predicates will be used to describe the $sim()$ function used in the Job Offers collection given below:

$$\begin{aligned} sim(e_1, e_2) \quad = \quad & compare_{title}(e_1, e_2) \wedge \\ & compare_{loc}(e_1, e_2) \wedge \\ & compare_{sal}(e_1, e_2) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasReferenceNum) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasDatePosted) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasExpiryDate) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasReportingTo) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasJobType) \wedge \\ & compare_{equal}(e_1, e_2, joci\#hasJobStatus) \end{aligned}$$

- $compare_{title}$ is implemented as `VacancyTitle.eval()`
- $compare_{loc}$ is implemented as `PartOfRelation.eval()`
- $compare_{sal}$ is implemented as `Range.eval()`
- $compare_{equal}$ is implemented as `StrictSameAttribute.eval()`

Although the same predicates are used in the candidate selection formula and in similarity measure formula, they are interpreted as different algorithms calling the corresponding methods of the `Predicate` class.

## 5.3 Semantic Description Compatibility Engine

The role of the backbone of the framework is played by a component called *Semantic Description Compatibility Engine (SDCE)*. It is the one that interprets the class model and interacts with the semantic repository during the identification process. The engine is called by two other components: Pre-filtering and Evidence Collection. The pre-filtering stage uses the *Entity vs. Ontology* interface that returns a set of entities retrieved from the semantic repository. The exact query for gathering entities is based on the class model and the restrictions of a currently processed entity. During the evidence collection stage the *Entity vs. Entity* interface of the engine is called. It is responsible for exposing the algorithm for calculating the similarity between two entities of a given class.

The overall architecture of SDCE is given in Figure 5.3. It consists of three main functional blocks:

- Rule Parser
- Rule Interpretation Engine
- Retrieve Compatible Entities module

### Rule Parser

The *Rule Parser* translates class model definitions from textual format to complex Java objects. It is responsible for loading the implementation of those classes that correspond to each of the predicates used in the model definition. The parser is also aware of the class hierarchy in the ontology used from knowledge representation and it is able to apply the model references expressed as `Super()` predicates e.g. on Figure 5.2. It also resolves namespace definitions of different ontologies and associates each formula to a particular class. Logical operators are interpreted as system predicates by the parser, which is responsible for their correct nesting.

### Retrieve Compatible Entities Module

The main functionality of the *Retrieve Compatible Entities* module is to filter out the irrelevant part of the ontology and form a set of instances similar to the source

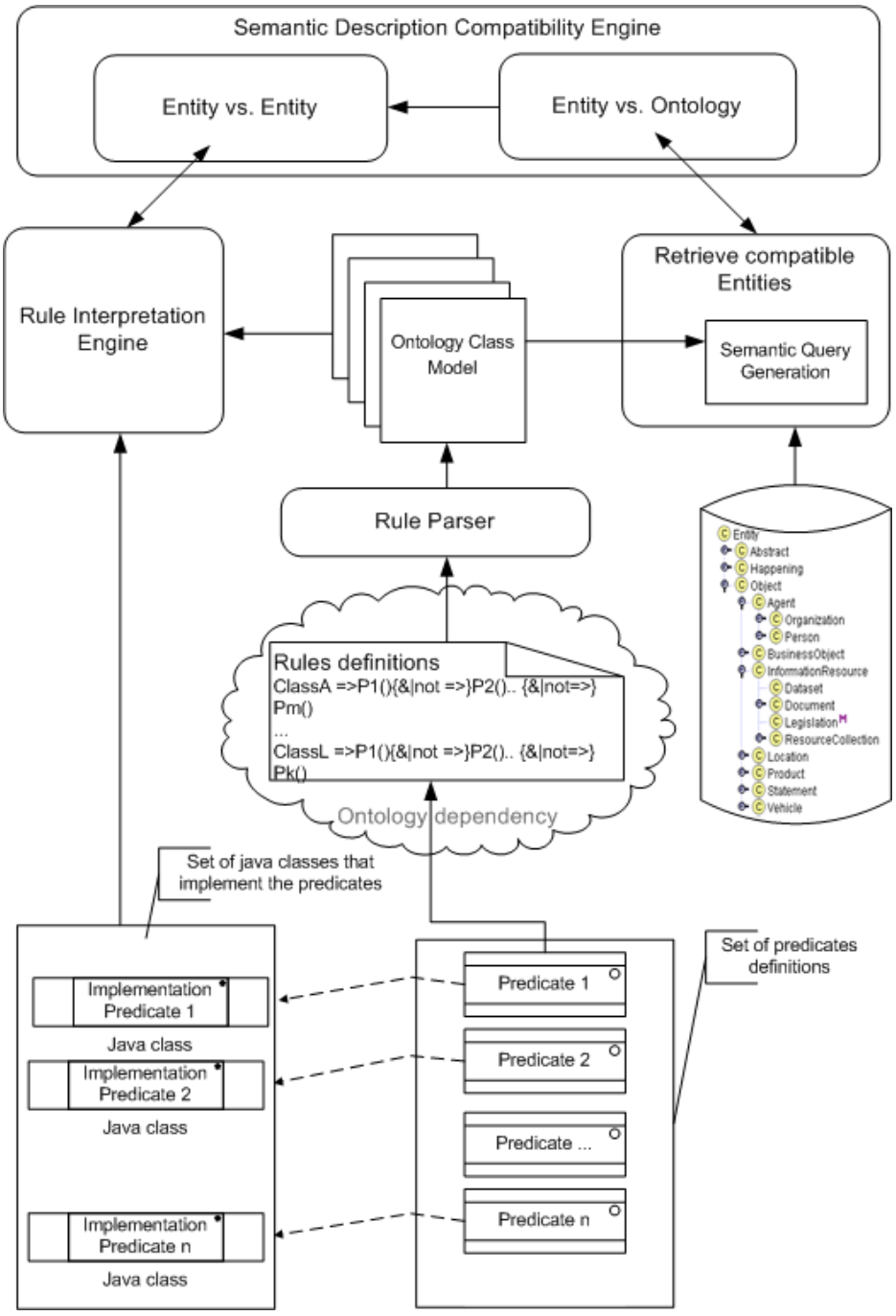


Figure 5.3: SDCE Architecture

entity. It uses the constrains formulated in the class model that are intended to restrict the whole amount of ontology instances to a reasonable number, to which the source entity will be compared. The model itself provides only general algorithm for retrieving filtering keys, and the role of this module is to specify their concrete values according to the attribute values of the incoming entity. Only then the selected instances will be potentially target instances that might be identical to the source entity.

The target entities already appear in the knowledge base and in this case the engine does not formally evaluate the class model/formula but composes a SeRQL[3] query. The query embodies the attribute restrictions with concrete values from the incoming object and it is then used for retrieving entities from the knowledge repository. The SeRQL (Sesame RDF Query Language) is one of the two widely supported semantic repository query languages. It has been chosen for the IdRF as being efficiently implemented in Sesame based RDF repositories as the one we use - OWLIM.

Another popular query language is SPARQL[4] - well developed and promoted by W3C[5]. The current IdRF implementation provides only SeRQL support, however in order to allow better interoperability and possibility for using other semantic repositories apart from OWLIM, the future development plans include providing a SPARQL adapter. The framework already supports a mapping mechanism for database schema to entity description translation. This allows for providing an adapter for generating SQL queries based on the schema mapping. The SQL queries are easily executable over a database e.g. MySQL[6].

The query itself is build from three types of restrictions.

- **result specification** - corresponds to the "select" clause of the query. The default selection result is composed by the unique identifiers (id or URI) of the candidates. Then the engine will retrieve the entire entity description for each of the selected entities. Possible optimisation at this stage is to select also those attribute values that will be needed during the similarity measure step and build partial entity description containing only necessary attributes. In this way the repository is queried only ones and the data volume is reduced to minimum.
- **path restriction** - describes the type of the queried entities as well as the attributes that will be used for expressing value restrictions and corresponds

[3] `http://www.openrdf.org/doc/sesame/users/ch06.html`
[4] `http://www.w3.org/TR/rdf-sparql-query/`
[5] `http://www.w3.org/`
[6] `http://www.mysql.com/`

to the "from" clause of the query.

- **value restriction** - corresponds to the "where" clause in the query. They specify the values, the way they are combined and comparison mechanism in the query.
  - the values are obtained from the attributes of the incoming entity descriptions
  - the comparison mechanism is obtained from the `prepareQuery()` method in the corresponding predicate in the class model and we distinguish between two possibilities: *equality* and *like condition*. The concrete choice of mechanism is predefined in the class model and reflects the nature of the attribute values. The "equality" is usually used for standardised data types e.g. date, postcode, website; while "like condition" is particularly useful for free text attributes e.g. description, title.
  - the combination between different attribute constrains follows the logic described in the formula of each class model. Possible combinations are "and", "or" and "not" conditions which correct nesting is provided in the class model.

The following walk-through example retrieves instances from the system knowledge base which are similar to a company called "MARKS & SPENCER". The input to the process is the class model and the incoming entity description. The company is of *musing#Company*type and according to the class model on Figure 5.2 the formula for this class looks like this:

```
"musing:Company":
    StrictSameAttribute(<musing:hasWebsite>) |
    OrganizationCombine(<musing:hasCanonicalName>) &
    StrictSameAttribute(<musing:hasPostal>)
```

Further, the incoming entity for "MARKS & SPENCER" has the following values for the attribute specified in the class model:

```
<musing#Company.2547> <rdf#type>   <musing#Company>
<musing#Company.2547> <musing#hasCanonicalName>"Marks & Spencer Plc."
<musing#Company.2547> <musing#hasWebsite>
                                "http://www.marks-and-spencer.co.uk"
<musing#Company.2547> <musing#hasPostal> "W2 1NW"
```

Based on the *musing#Company* class model and the definition of the source entity the SDCE will build the following restrictions.

- result specification is left as default - selecting only URIs of the entities.
- path restriction - it includes the class path and all the attributes that will be used in the value restriction part of the query : *musing#Company*, *musing#hasCanonicalName*, *musing#hasWebsite* and *musing#hasPostal*
- value restriction - the resulting restrictions that will be applied to the pool of all available entities in the knowledge repository correspond to the predicate definitions in the class model. `StrictSameAttribute()` will result in "equality" condition while `OrganizationCombine()` will be interpreted as "like" condition. Thus the restriction will be formulated as:

```
(
 <musing:hasCanonicalName> like "Marks" AND
     <musing:hasCanonicalName> like "Spencer" AND
     <musing#hasPostal> =  "W2 1NW"
 OR
 <musing#hasWebsite> =  "http://www.marks-and-spencer.co.uk"
)
```

Once all the three types of restrictions are formulated, they are automatically recomposed as a SeRQL query below, which makes the query ready to be evaluated. It is sent to the semantic repository and the SDCE engine composes entity descriptions for each of the entities behind the selected URIs. The retrieved objects are then returned to the pre-filtering component.

```
select DISTINCT
  V1
from
  {V1} <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
             {<http://ontotext.com/2007/07/musing#Company>};
       [<http://ontotext.com/2007/07/joci#hasURL> {V2}];
       [<http://ontotext.com/2007/07/musing#hasCanonicalName> {V3}];
       [<http://ontotext.com/2007/07/musing#hasCanonicalName> {V4}];
       [<http://ontotext.com/2007/07/musing#hasPostal> {V5}]
where
  (V2 = "http://www.marks-and-spencer.co.uk") or
  ( (   ( (V3 like "*marks" IGNORE CASE )
```

```
            or (V3 like "*marks *" IGNORE CASE )
        ) and
        ( (V4 like "*spencer" IGNORE CASE )
            or (V4 like "*spencer *" IGNORE CASE )
        )
    )
    and (V5 = "W2 1NW")
)
```

The presented modular composition of the query that is based on three elements corresponding to "select", "from" and "where" clauses gives us huge flexibility. The example above concludes with translating the internal query representation into a SeRQL query, however other possibilities are also possible. For example using a SQL query is also one of the options provided by the framework, although using data repositories which do not support ontology base representation require specific mapping mechanism. In order to build an entity description for each of the selected objects, the framework uses datatype adapter that applies a manually created mapping between entity classes and attributes, and tables and columns in the database. However usage of databases does not eliminate the need for having an inference engine which can operate on the ontology and support building the class model and executing the similarity measurement algorithms base on class or property hierarchy.

### Rule Interpretation Engine

The main role of this engine is to interpret the class model and to collect as much as possible evidence about the similarity between the source entity and a target object retrieved from the knowledge base. The input to the engine is the two entity descriptions and the class model formula for comparison. The process corresponds to Entity vs. Entity interface of SDCE.

According to the class model corresponding to the type of both entities, the total similarity score is calculated based on individual predicate algorithms. Once calculated the values of different predicates are combined according to the logical connectives in the corresponding formula. Logical operators are implemented as a system predicates following the similarity measure interpretation given in Section 4.1, where the correct nesting of the operators is encoded in the class model.

The overall score is a real number in the interval between 0 and 1, where 0 means that the given entities are totally different and value 1 means that they are abso-

lutely equivalent. It is computed by comparing entity attributes using the predicates in the class model formula. Although the predicates implement different comparison algorithm e.g. text edit distance, inverted frequency based matching, context similarity; their result by definition is in the range of 0 to 1. The combination of these predicates is specified in the formula using logical operators as defined in Section 4.1, therefore the final result is also in the required range.

Each of the predicates takes an attribute type as a parameter and collects corresponding values from the entity descriptions. Formally, a predicate value is calculated according to its algorithm, which reflects the specificity of the predicate and entity attribute. The algorithm is implemented as `eval()` method, and the corresponding Java class has access to all details of both instances, thus it is possible to use values of different attributes apart from the explicitly passed type.

As an example, *OrganizationLD()* predicate computes Levenshtein Distance between the values of a given attribute. It does not use any other attribute values and this makes it applicable to a wide range of attributes. If any other attribute types are used in the algorithm, it makes it difficult to reuse the predicate in other applications or domains that are defined by another ontology. However, the freedom of involving other attributes is very powerful tool for encoding complex comparison algorithms e.g. relation similarity, where more than one type of relations can be followed during the inference process.

# Chapter 6

# Evaluation of the Identity Resolution Approach

The evaluation in this work is based on the Identity Resolution Framework that implements the proposed approaches for data acquisition from multiple sources. Since it is based on a software system, there are two aspects of evaluation according to its integrity: black-box and glass-box testing. The black-box evaluation requires complete run of the system on a given data set and measuring the quality of the process (speed, reliability, resource consumption) and the quality of the result (e.g. accuracy of the identity resolution). However, it does not highlight the problems and possibilities for improvement within the system. The glass-box evaluation is the one that considers different components that the system consists of (e.g. implemented knowledge representation formalism, various algorithms, etc.) and measures their performance and impact to the overall system performance, therefore the glass-box testing can be seen as complementary to the black-box evaluation. Thus, for each of the proposed solutions we use the aspect, which is the most appropriate for proving the contribution of the approach that is taken.

## 6.1 Evaluation Approach

There are very few standards for evaluation of research prototypes and systems that implement innovative techniques. Some popular ones are ISO 9126 Standard sets[1] that are intended to provide a general framework for evaluation design. They

[1] Search `http://www.iso.org` for the official version of each of the standards in the set.

consist of six quality characteristics: functionality, reliability, usability, efficiency, maintainability and portability. Each of these is specified by its sub-characteristics that can also be further subdivided to reflect the specific system needs.

- **Functionality** - *express the satisfaction level of stated or implied needs.*
- **Reliability** - *measures the capability of software to maintain its level of performance.*
- **Usability** - *is the effort needed for use.*
- **Efficiency** - *is the relationship between the level of performance and the resources used.*
- **Maintainability** - *reflects the effort needed to make specified modifications.*
- **Portability** - *is the ability of software to be transferred from one environment to another.*

Another big effort on providing evaluation framework is presented by EAGLES group. The proposed evaluation approach is intended to serve the NLP system needs and EAGLES group, together with the TEMAA LRE project adopt and extend ISO 9126 standard as a starting point to identify attributes and define measures and methods whereby values for those attributes [EAG95].

Although it is a specific type of NLP systems, evaluation of Machine Translation (MT) defines different possibilities for characterising quality of a system. According to [Jones & Galliers 96] evaluation of MT systems is given in three categories:

- **Linguistic assessments** - *measures the errors in translation in black-box testing and proves a linguistic theory in glass-box testing*
- **Operational assessment** - *measured the translation adequacy and the amount of human intervention needed or more generally what further resources if any would be required to achieve certain level of performance*
- **Economic assessments** - *measures the cast efficiencys of the system for a given task, compared to the cost of human translation.*

Considering the above evaluation approaches, we aim at evaluating the quality of the system that implements the proposed identity resolution approaches by means of different quality characteristics and measures. However, not all the ISO 9126 quality characteristics are applicable to evaluating the proposed Identity Resolution Framework. Therefore, we decided to adopt a sub-set of the standardised characteristics in combination with the quality categories of MT systems. Finally, we have chosen the following characteristics:

- **Accuracy** - *the very first characteristic that is somehow obvious to be measured. It directly corresponds to the functionality characteristics defined by the ISO 9126 and following the MT categories description it covers what is given in linguistic assessments.*
- **Efficiency** - *implies directly the corresponding ISO 9126 characteristics.*
- **Maintainability** - *Since we claim that one of the main characteristics of the proposed approach is it generality or applicability in different domains and various types of incoming data. We extend the notion of the corresponding ISO 9126 characteristics and include the portability aspect as well. In this way it mostly covers operational assessments of the MT categories.*

In view of the fact that we are not interested in the software as such but in implementing and proving research approaches, reliability and usability software characteristics as standardised in ISO 9126 go beyond our needs. Economic assessments category of MT evaluation is also hardly applicable to the proposed system, since there is no direct human action analogue to be compared to. The expected economic stimulus may come from including identity resolution as a component of solving particular problem and measuring its impact. The general benefits of solving the identity resolution problem are discussed in the previous chapters.

A detailed discussion about each of the proposed characteristics and possible measures for evaluating the identity resolution approach are given below:

## 6.2 Accuracy Evaluation

Evaluation has a long history in Information Extraction (IE), mainly thanks to the MUC conferences, where most of the IE evaluation methodology (as well as most of the IE methodology as a whole) was developed [Hirschman 98]. In particular the DARPA/MUC evaluations produced and made available annotated corpora that have been used as standard test beds.

The evaluation of the proposed new technique to combine information from multiple and redundant sources could not be easily performed because of the lack of similar systems. However, the benefits of the technique usage could be shown comparing the results from an implemented system to a classical IE system as a baseline. For this purpose, we will implement two type of systems for each experiment and they will be evaluated on a human annotated corpus. The first system will use current technologies and its result will serve as a base line. Then the results from the newly proposed technology will be compared to the baseline and we expect that this comparison will clearly show us the benefits of using the proposed technique.

On the other hand, there are several specific domains where the proposed identity resolution approach is applicable (e.g. author name disambiguation) and where different approaches are already implemented. Thus we plan to examine the existing systems and use them as a baseline for our accuracy evaluation.

There are several well-established metrics for evaluation of traditional IE systems.

The ***Precision & Recall (P&R)*** is probably the most popular standard metric, which is used since the very beginning of the information extraction research. It is defined and proved during the MUC series [ARPA 93]. The *Recall* score measures the ratio of correct information extracted from the texts against all the available information present in the texts. The *Precision* score measures the ratio of correct information that was extracted against all the information that was extracted. The harmonic mean of Precision and Recall will be reported using *F-measure*, where F = (2*P*R)/(P+R).

One of the main limitations of this metric is that it evaluates all results only as absolutely correct or absolutely wrong and does not originally handle different degrees of partial correctness. However, it is easy to be used and is very popular in the information extraction community.

The ***Cost-based evaluation (CBE)*** [Sassone 87] is more recently established metric used in some of the DARPA competitions, such as TDT2 [Fiscus *et al.* 98] and ACE [ACE04]. It is superior to *P&R* in some aspects because it allows multi-dimensional evaluation. *CBE* is characterized in terms of *"misses"* and *"false alarm"*. A *"miss"* occurs when an entity of a target is mentioned but not output. A *"false alarm"* occurs when a spurious entity is output. One of the advantages of CBE metric is its ability to be adjusted to the different users' needs, who might have different requirements to a system. It is particularly suitable for the industry since the metric allows one to specify their own criteria. For example to give higher priority to the correct extraction of a particular fact/concept (e.g. Person) instead of another (e.g. Location). Although the *CBE* model guarantees the most flexible application of evaluation metric, often more simple version is needed.

Another solution consists of augmenting traditional *P&R* metrics by adding some kind on semantic distance weights, such that the gravity of the error can be taken into account. This metric is called ***Learning Accuracy*** and was initially used for evaluation of ontology population [Hahn & Schnattinger 98, Cimiano *et al.* 03]. It measures how well a concept or instance is added/accociated to an ontology. Essentially, this metric provides score somewhere between 0 and 1 for any concept in an incorrect position in the ontology; 0 for missing and spurious; and 1 if it is correct (as *P& R*).

One recently proposed evaluation metric that aims to preserve the useful properties

of the *P&R* scoring, but combine them with a cost-based component is presented in [Maynard 05] and is called ***Augmented Precision & Recall***. This measure takes the relative specifics of the taxonomy positions of the key and the response into account in the score, but it does not distinguish between the specificity of the key concept on one hand, and the specificity of the response concept on the other. The authors [Maynard *et al.* 06] claim that this metric would be more appropriate for ontology aware information extraction systems, since the traditional *P&R* metric are binary rather then scalar. Essentially, this measure provides a score between 0 and 1 for the comparison of the key and response concept with the respect to a given ontology. In case of an ontology mismatch this method provides an indication of how serious the error is, and weights it accordingly.

In conclusion, the last described measure (*Augmented Precision & Recall*) seems to be the most suitable for our need, because it provides techniques for ontology aware IE evaluation and claims to integrate all the advantages of the previous metrics. However, it was recently presented to the audience and is still on the way to prove its usability.

We have carried out two series of experiments based on different data collections described in Section 3.1.3. In a first experiment, we have merged job offers from different web sites. In a second experiment we have evaluated IdRF in the context of identification and merging of information extracted from company reports.

### 6.2.1 Job Offers Accuracy Evaluation

The set of job offers collected in the first experiment consists of automatically extracted vacancies with 45% rate of redundancy (see Table 6.4), e.g. two of each five vacancies are a duplicated version of an already collected vacancy. Duplicates are usually described with less vacancy details.

#### Data Acquisition

Before merging the extracted vacancies from the web sites, we measured the single page IE accuracy. The results on Table 6.2 show errors introduced by automation of the processing on this step. For this evaluation, we took a sample of documents - 1,266 web pages - crawled from two web sites. Then we measured the accuracy of running the vacancies extraction algorithm on all the documents in the set and the result was very poor precision (37,5%) because of the many false positives - “fake” facts - that are extracted.

In the second experimental setup we ran the system over about 150,000 U.K. company web-sites. The main difference from the previous run is that not all the pages of corresponding websites were crawled, but only those classified as job related. As presented on Table 6.1 only one third of the crawled pages are considered to be related to vacancies by the classification algorithm and respectively processed by the IE module. However this statistics demonstrates the complexity of the chosen domain and the huge amount of data that still needs to be processed.

| Statistics | |
|---|---|
| web-sites processed | 32,063 |
| web-pages crawled | 1,981,634 |
| web-pages that IE is ran over | 514,255 |
| *Vacancies* extracted | 101,671 |

Table 6.1: System Statistics

Based on the restriction of the classification algorithms during crawling, the accuracy of the IE improved. It is due to the fact that the irrelevant pages were filtered before running the IE. It appeared that only about 10% of the pages are jobs related. Moreover the precision of the IE raised dramatically up to 83% avoiding wrong extraction from irrelevant pages (see Table 6.2) .

| Type of crawling | Standard | Focused |
|---|---|---|
| Num. of documents for IE | 1266 | 102 |
| Precision for *Vacancy* extraction | 37,5% | 83,1% |
| Recall for *Vacancy* extraction | 92,3% | 92,3% |
| F-measure for *Vacancy* extraction | 53,3% | 87,4% |

Table 6.2: Evaluation of crawling strategy

The final experiment of the quality of the IE process of vacancies aims at evaluating the accuracy of single attribute values. We compared automatic annotation of the required attributes to a manually annotated corpus of 1,000 documents. The corpus is created from a representative sample of pages estimated by the number of extracted vacancies. Most of the documents contain one to three vacancies, very few none, and some of the pages contain up to thirty vacancies. The overall extraction results for different attributes are given in Table 6.3.

### Data Consolidation

Once having reliable single page IE results we investigated the redundancy phenomena. We took a sample of 3,000 web sites and manually compared the extracted vacancies. Our experiment showed that about two thirds of the company web-sites

| Attribute | Precision | Recall | F-measure |
|---|---|---|---|
| JobTitle | 0.86901778 | 0.864677059 | 0.866841985 |
| ReportingTo | 0.998355263 | 0.994863238 | 0.996606192 |
| Job_Category | 0.990038568 | 0.969606984 | 0.979716264 |
| Job_Location | 0.979849533 | 0.647925438 | 0.780045706 |
| Location | 0.895247003 | 0.917805307 | 0.906385818 |
| Job_Reference | 0.97660681 | 0.895991888 | 0.934564123 |
| Job_Type | 1 | 0.997510669 | 0.998753783 |
| Salary | 0.974346177 | 0.874598217 | 0.921781566 |
| End_Date | 0.996664295 | 0.933743177 | 0.964178287 |
| Start_Date | 0.978004348 | 0.918809399 | 0.947483208 |
| Person | 0.865322878 | 0.944216858 | 0.903050022 |

Table 6.3: Evaluation of single attributes extraction

have redundant job advertising. Moreover, the consolidation successfully reduces the number of facts to about 55% of the single page extracted results (see Table 6.4).

| Statistics | |
|---|---|
| web-sites with extracted *Vacancies* | 2 922 |
| web-sites with redundancy | 2 171 |
| *Vacancies* before merging | 29 963 |
| *Vacancies* after merging | 16 592 |

Table 6.4: Redundancy Statistics

For this purpose of semi-automatic evaluation we define duplicate candidates as follows. Two vacancies can be merged if they have equivalent "Vacancy Title" attribute values and the values of the rest of their attributes are semantically compatible according to the knowledge base, i.e. the two compared instances are connected with certain types of relation that is semantically consistent. An example for such a relation is *subRegionOf* and we say that "*locatedIn* Wales" is comparable to "*locatedIn* UK", since "Wales" is a *subRegionOf* of "UK".

A very simple diagram in Figure 6.1 presents the choice of most specific values for *"Vacancy Title"* and *"Vacancy Location"* attributes presented as KB relations. For simplicity, the relations between *"Vacancy Title"* attributes are presented as "subStringOf", but the actual comparison is mainly based on the Standard Occupational Classification (SOC)[2] system for occupational categories.

The exact matching pairs were decided to be correctly identified and the rest was evaluated manually. The overall evaluation scores are given in Table 6.5.

[2] `http://www.bls.gov/soc/home.htm`

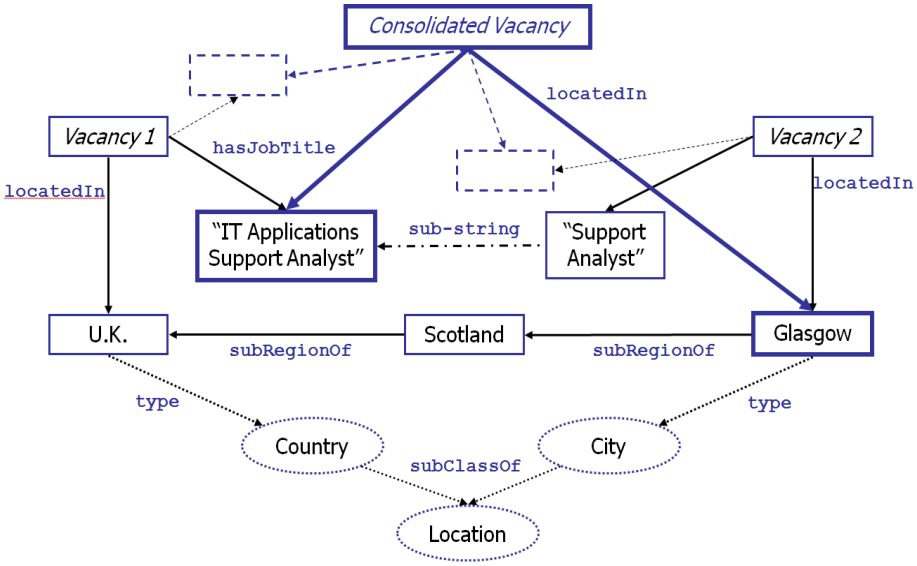


Figure 6.1: Example of consolidation of two Vacancy facts

### 6.2.2 Company Profiles Accuracy Evaluation

The input from the company collection experiment is a set of RDF statements where each statement is either a new fact or a known fact. Three hundred company profiles from the UK have been analysed by third party OBIE system in order to carry out this experiment. The process has targeted a set of 310 UK companies and attempted unification against an initially populated knowledge base consisting of 1,801,868 different companies. They are manually collected and provided by a company called "Market Location"[3]. Formally these company records are stored and retrieved from a RDBM - MySql repository - using manually created mapping between the DB schema and the Ontology as explained earlier.

Table 6.5 presents the results of the overall evaluation. An F-Measure for merging correct information of 0.90 which is rather encouraging.

| | Precision | Recall | F-measure |
|---|---|---|---|
| Company profiles | 0.97 | 0.84 | 0.90 |
| Vacancies | 0.82 | 0.89 | 0.85 |

Table 6.5: Evaluation of Information Merging

It is extremely difficult to compare the accuracy of the presented application on company profiles to other systems, mainly due to the lack of evaluation corpora.

[3]http://www.marketlocation.com

Therefore we take exact match of two attributes and their combination as a baseline. We have chosen company names and company website address and the most distinctive and representative attributes. The exact match on company names takes the two string and performs trivial normalisation:

- ignoring company suffixes if presented only in one of the names e.g. *Plc* in "AEGIS GROUP" vs. "Aegis Group Plc".
- extending domain specific shortenings that appear often in the web corpus e.g. *TELECOM* and *GRP* in "COLT TELECOM GRP"vs. "Colt Telecommunications Group"

| | Precision | Recall | F-measure |
|---|---|---|---|
| EM name | 0.963 | 0.274 | 0.426 |
| EM url | 0.966 | 0.607 | 0.743 |
| IdRF match | 0.970 | 0.838 | 0.899 |

Table 6.6: Comparison between Exact Match(EM) of company names, URL of the company websites, and the approach implemented in IdRF

As shown on Table 6.6 exact match on both company names and websites give good precision, but poor recall. In case of the names, exact match yields poorly compared to more sophisticated similarity metrics used in as part of the application of IdRF. Lower recall in URL comparison is due to the fact that although distinctive, this attribute is not available for all entities. Therefore identical entities, which however are not described with their websites, are omitted during the comparison. Finally we can concluded that using advanced similarity metrics and combination of different attributes performs best reaching nearly 90% accuracy.

In what aggregation of information is concerned, Table 6.7 illustrates the percentage of details updates (entering or change) by attribute for our application on company profiles. The total count of updates includes also newly added records. The reader may notice here that the details that tend to change more often like offices addresses and phone numbers are mostly affected by the updates, however relatively stable details e.g. websites will be updated less often.

| Update statistics | total | Preexisting entities |
|---|---|---|
| postal and address | 60% | 38% |
| website | 35% | 12% |
| phone | 86% | 62% |

Table 6.7: Statistics on company profile integration

## 6.3 Efficiency

It is always the efficiency of the systems that is taken into account when choosing software solutions. Sometimes it may depend on the implementations, but mostly it is the algorithms that make the difference. Thus we are interested in measuring this characteristic as part of the whole quality evaluation of the approach.

Obtaining information from multiple sources deals with huge amount of input data, therefore the efficiency measure that is most important for this work is scalability. In the proposed approach the information sources and the facts they consist of are processed one by one and compared to all the already identified objects and finally the new fact are inserted to the system's knowledge base. Therefore we are interested on how the system scales or its ability to process the incoming data with respect to a constantly growing knowledge base.

There are three characteristics of the efficiency that can be easily measured:

- *scale* - the amount of processed incoming and target facts;
- *speed* - the time it takes for an operation to complete;
- *space* - the memory or non-volatile storage used up by the system

In this work we present two domains of the identity resolution approach: job offers collection and company profile collection. Since they are implemented and serve as use-cases for using IdRF in real applications, we also provide evaluation based on them. In order to prove the efficiency of the chosen approach and its implementation we need a large scale scenario, where small deviations in the performance may make a difference.

In job offers collection we face the identity resolution problem in a very limited scale. The vacancies that need to be resolved are coming from a single advertiser and the average number of vacancies per advertiser is 10 (only about 150 advertisers offer more than 100 positions at a time). Therefore the vacancy amount could not challenge the efficiency of the solution. Thus we focus on the second scenario - company profiles.

### Scale Evaluation

Company profiles are obtained from two sources: the Web, and a static set of manually collected organisation records. The static collection is used for initial pre-population of the target set of entities which are already induplicated (manually). Then the new coming profiles extracted from the Web are compared to them, and finally the target set is enriched with the new information.

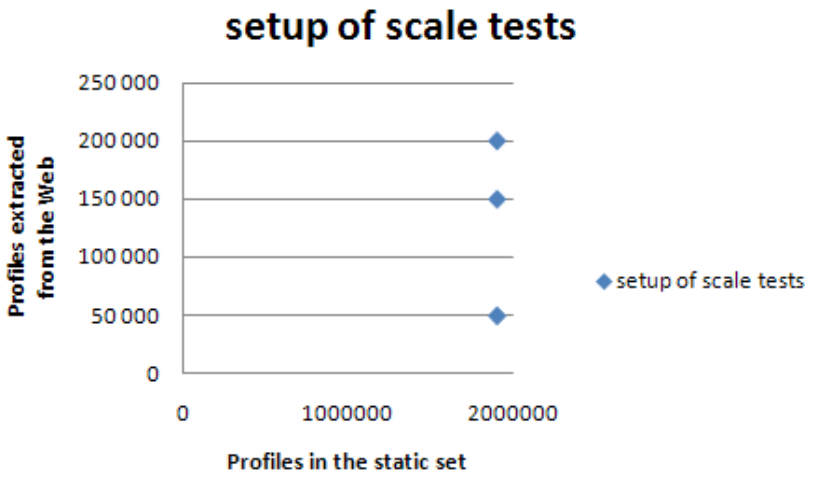


Figure 6.2: Scale tests setup

We planned for three scale tests with pre-collected set of the following sizes: 50K, 150K and 200K entities of crawled profiles. The size of both data sets - static and extracted from the web - wherefrom the entities are identifies is given as one point for each experiment on Figure 6.2. All the three experiments use the full static set consisting of about 1,9M records. Not surprisingly all entities in the three source sets were processed successfully. This is mainly because of the fact that the identity resolution approach we propose takes the incoming entities consequently. Therefore the amount of the incoming entities expectedly does not challenge the ability of the system to process them.

The factor that may reflect on the system performance is the size of the target set to which each new coming entity is compared. Since it grows during the test run with the size of the information that is added as a result of identification of each new entity, it may cause deviations in the system behaviour. To simulate larger incoming sets we pre-populated the target set of already processed entities, that otherwise will be extended during the system run. Hence we use the maximum amount of company profiles that are available. Finally, monitoring the system behaviour during the three tests we found no indications of processing problems. Thus we can conclude that IdRF can handle more than 2M entities, which is close the estimate for all UK registered organisation. Therefore it easily scales for applications in this domain.

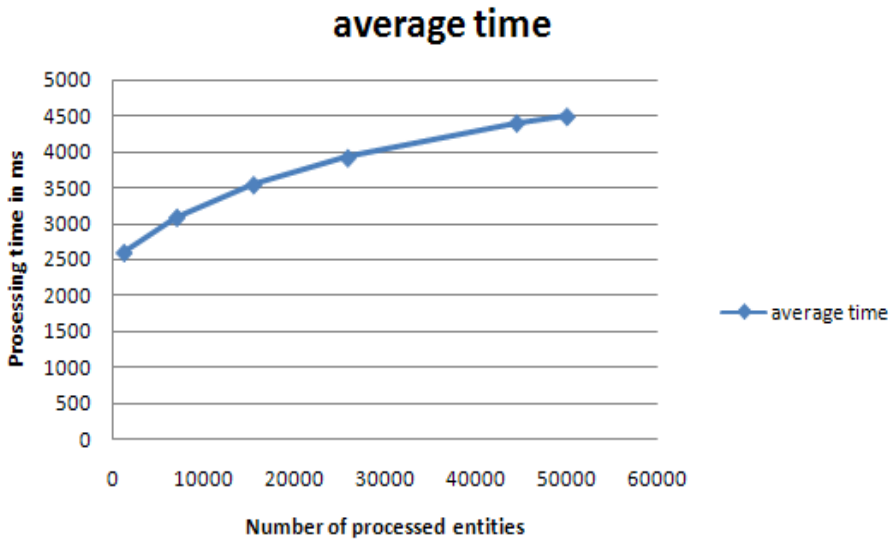


Figure 6.3: Scale tests results

### Speed Evaluation

The time it takes for the identity resolution to complete is one of the important features of any data processing system. It heavily depends on the domain of the application, including the required data pre-processing, the selection criteria and the complexity of the algorithms chosen for entity comparison. Therefore it is difficult to compare experiments on different setups. However here we are reporting the result from a large-scale use-case processing of about 2M company profiles as discussed above.

The environment setup for this experiment is 1024M virtual memory on Intel Xeon CPU on 2.00GHz and we measure the average time for processing incoming entities in the smallest test set of 50K profiles. The results are presented on Figure 6.3. The average time steadily grows with the amount of processed entities, which is expected behaviour due to the fact of enlarging target repository.

Similar effort is reported by [Benjelloun *et al.* 09], where the authors evaluate two domains of maximum size of 2,4K records. Although our evaluation is of different scale, the complexity of the domain is comparable. On Figure 6.4 is presented a visual comparison of the IdRF performance on company profile identification and the result for F-Swoosh algorithm (the best performing out of the three ones presented by Benjelloun at al.) on hotel profiles.

The results for F-Swoosh are taken down from the graphics printed in the paper,

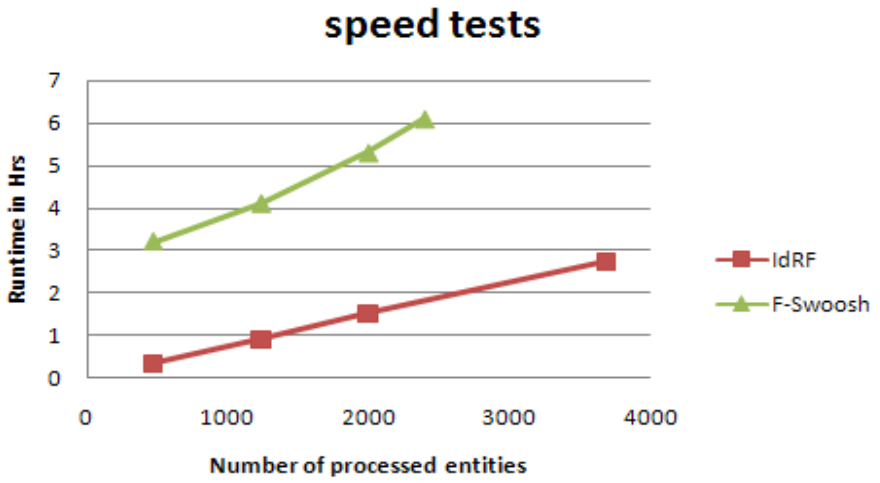


Figure 6.4: Speed test comparison

therefore they may be slightly modified in our representation. However the overall conclusion one can make is that both systems show similar trends over time. The environment setup for F-Swoosh is reported to be 1.8GHz AMD Opteron Dual Core processor with 24.5 GB of RAM which is much more powerful machine then the one used in the company profiles experiment. However the F-Swoosh algorithm takes about four times longer to process a single record mainly due to the complexity of the clustering algorithm they use for blocking comparison candidates. Detailed overview of F-Swoosh algorithms is given in Chapter 2.

### Space Evaluation

The last measure of the efficiency of our approach is space in terms of memory or storage used up for applying it. There are two aspects of this evaluation: the space required for processing entities, and the space required by the semantic repository or any other data storage that is used.

As shown in the Scale experiment above, the IdRF setup with 1024M serves the needs of the smallest experiment of 50K entities. The next two tests of 150K and 200K are also successfully processed over the same setup. These results support our expectations for constant need of memory that comes out of the fact that incoming entities are processed one by one. Any aggregation of in-memory objects will speak about a memory leak, since after identifying a single entity all the system objects created in connection with its processing should be destroyed releasing the taken

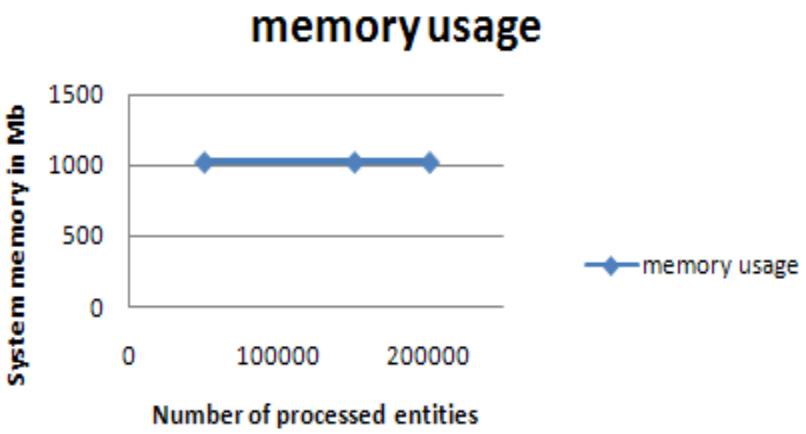


Figure 6.5: Memory usage evaluation

memory. In fact our experiments show that processing more entities does not require more memory (see Figure 6.5).

The second aspect of space usage that was separated from the entity processing is the storage required by the semantic repository. Since it is a third party component and does not influence the space requirements of the core system, we are not considering its needs in the evaluation. Moreover, the semantic repository can be partially substituted by a relational database or other data storages e.g., indexes, which have very different space usage and optimisation possibilities. For example, in the company profile experiment we have partially substituted the semantic repository with MySQL database that allows for storing and retrieval of comparison candidates. We used MySQL setup of 4GB memory.

Finally we can conclude that our identity resolution approach and its implementation as IdRF have relatively low space requirements. Execution of company profile scenario takes 6 times less space compared to [Benjelloun *et al.* 09] F-Swoosh if we take into account the memory used by MySQL and 24 times less if the repository is excluded from the calculations.

### Efficiency Analysis

From a large-scale application prospective, [Kounev & Buchmann 03] define three questions that are used for further analysis. They put the efficiency measured values in the context of an application launch in production. The authors originally design requirements to support size and capacity estimation of the deployment environment

needed to guarantee the Service Level Agreements (SLAs) for the corresponding product. Although we are evaluating a research prototype, it is a software system and the formulated questions are still relevant. Therefore we provide some of the answers below.

- What are the maximum load levels that the system will be able to handle in the production environment?

  As shown in the scale analysis, the proposed approach easily scales up to 2M entities. We could not manage to obtain larger evaluation set, however our estimates for a production setup of the identity resolution on company profiles will not dramatically exceed the tested range. Thus we expect that the proposed application will be able to handle the amount of new coming company profiles.

- What would the average response time, throughput and resource utilization be under the expected workload?

  The speed analysis gives us the average processing time per an entity. In a real-time processing, it can be taken as a response time - how long it takes to return a result after an entity is passed for resolution.

- Which components have the largest effect on the overall system performance and are they potential bottlenecks?

  The answer of this question involves all the three aspects of efficiency measurement: scale, speed and space. We already showed that the evaluated system has the potential to scale on large volumes of data, however the processing speed decreases by the number of input entities. The reason for this is the growing amount of potentially identical entities in the target dataset. The required space of the system however does not depend on the number of processed entities, which is most probably not true for the semantic repository. Therefore we can identify the bottleneck of the overall identity resolution process to be in the repository and its implementation.

  There is a long history record in developing data repositories. Recent enthusiasm for semantic repositories leads to noticeable results in this area which compete in terms of speed and performance. Following [Kiryakov *et al.* 09] we have chosen OWLIM as the repository that outperforms its competitors. In this way we address the risk of an IdRF based application to be stuck in the bottleneck of the process. Apart from this, we have implemented a mechanism for easy substitution of one data storage with another, thus if any other repository better matches the requirements of a particular application or a domain, it can be used instead of the default repository.

Based on the above analysis, we can conclude that the proposed identity resolution approach and its implementation in IdRF are an efficient solution of our problem.

## 6.4 Maintainability

One of the targeted innovative characteristics of the identity resolution approach is its generality. We aim at producing a system that can be configured to solve the identity issue in wide range of domains for various types of objects. Thus the maintainability of the approach is a major quality characteristic of the proposed solution.

Although maintainability can be seen as an aspect of pure software development process, it highly depends on the chosen architecture. Since we aim at researching identity resolution problem, the resulting framework should allow facile replacement and arrangements of its components and implemented algorithms. To assist various test cases and hypothesis to be evaluated we consider maintainability as one of the main quality characteristics of the proposed identity resolution framework.

ISO 9126 standard provides broad and informal definition of the maintainability:

*"Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in environment, and in requirements and functional specification."*

In order to apply maintainability as one of the evaluation characteristics to our identity resolution approach and its implementation in the identity resolution framework, we will further specify possible changes that may require modification of the framework. However it is extremely difficult to formalise any specific metric for calculating the maintainability, therefore we introduce three categories of capability degree of modification:

- hard to modify - the features are hardcoded as part of the internal identity resolution logic, thus any modification will result in changing the theoretical assumption behind the software
- moderately easy to modify - change in the behaviour or implementation of the components without changing their interfaces and communication with the rest of the system
- easy to modify - requires only change in the configuration of components or plugging new implementations of algorithms.

Using the standard description we consider the following changes that result in modifying the IdRF as corresponding to a new identity resolution case.

- **changes in the environment** - In the context of our problem, changing the environment corresponds to changes in the data sources. Since the proposed framework may be used in real time application, the nature of the data sources may change in time. The most common modification is in the exposed schema of the data. However schema alignment step of the identity resolution process is still not held automatically in the framework. It is required as a manual pre-process.

  Another possibility is that the entity description model changes as well as the source schema. Then apart from providing a new schema alignment mappings a new ontology should be applies. If new attributes are included, the old model will still work as before ignoring the new values. Therefore adding new attributes does not increase the maintainability cost, however it may lead to new application requirements (discussed below). If some of the attributes are removed, this technically means that the corresponding attributes hold null values, then the behaviour of the pre-existing setup may be changed and returning no results depending on the role that the missing attributes play in the identity process. This again changes the requirements to the system.

- **changes in the requirement** - The requirements refer to the domain of the application. They specify the entities that need to be identified, the target pool of entities that are already processed - pre-populated or empty - and the comparison mechanism. Change in the requirement can also be seen as change in the scenario where the framework is used. It is the most complex modification that needs to be evaluated, therefore it is presented in more detailed below.

- **change in the functional specification** - The IdRF architecture show on Figure 5.1 consist of several functional blocks. Some of them are more pliable, others have very strict functional role in the framework, therefore are hard to be modified. The main goal of the framework is to provide reusable infrastructure for building identity resolution applications, therefore it is designed to support customisation within the frame of the chose identification approach. Each of the components has a strict role in the entire process, however their specific behaviour with respect to the domain and the particular goals of the application are flexible. It allows for implementation of a wide range of selection and comparison algorithms as well as their fine tuning based on further domain research.

The IdRF architecture matches the dataflow suggested in this work (see Chapter 3). Thus its components are specified to meet the requirement coming from the chosen

approach. Their high level behaviour is not a subject of modification; otherwise they will result in a completely different system. Therefore what can be modified is not what the components do, but how they do it. Each of the components allows implementation deviations to certain extend.

- *the semantic repository.* Although it is formally not part of the framework, the semantic repository plays a significant role in the identity resolution approach providing semantic data representation storage. It is heavily used by the other components, especially in the data selection stage of the process. The repository that is used in IdRF is OWLIM, however it is easy to be substituted by any other semantic repository that supports SeRQL.

  It is also possible to be partially substituted by other data storages e.g., relational database. The semantic repository will still be needed for applying inference during class model interpretation, however the target data used in data selection stage can be efficiently stored and retrieved from other types of repositories e.g. file storage, database, full text indexes. Detailed explanation on how to add adapters for different data storages is given in Section 5.3.

  ⋆ *easy* to modify - Modification of the semantic repository is *easy* to be obtained in case the type of the repository is already supported it is a matter of configuration only.

  ⋆ *moderately easy* to modify - If the chosen repository is not supported already, it requires implementation of additional adapter by example, using the existing interface, which makes it *moderately easy* for modification.

- *semantic description compatibility engine* supports interaction between the components in IdRF. It encodes the logic of how the class model configuration should be interpreted and provides access to the semantic repository. It behaviour should neither depend on the particular usage of the framework, nor it should be accessed by third parties. Therefore it is not subject of modification.

  ⋆ not a subject of modification

- *pre-filtering* component is responsible for retrieving compatible candidates from the data repository. However its communication with the repository itself is isolated by the SDCE. Therefore it does not need any modification while changing the repository. The retrieval process depends on the selection criteria that are configurable.

  The behaviour of this component is specified by the data selection rules provided as a class model. Each formula may describe different selection constrains, thus the modification in the pre-filtering stage are *easy* to be

made using the configuration mechanism provided by the SDCE class model implementation.

⋆ *easy* to modify

- *evidence collection* is the most suspected subject of modification. It executes the actual comparison of entities and provides evidence on this similarity. The similarity measure is specific for each type of entities, therefore when a new entity type is added for support it needs configuration. The attributes it consists of may hold different semantics and allow for different interpretation than entities of other types. Thus the change in the similarity measure - the main function of this component - can be easily configured in the class model of the corresponding type of entities.

  The logical operators that combine similarity measured on individual attributes are hard coded in the framework intentionally, so that they cannot be modified. Any change on the operators may question the total framework performance and the range of corresponding similarity function.

  The algorithms that calculate similarity scores over a given attribute type, however, can be two types: default and custom. Default algorithms, implemented as pre-defined predicates in the framework, provide a basic set of comparison mechanisms. Their usage is simply a matter of configuration of the class model. This set can be further extended by custom implementations and domain specific algorithms. They have to implement `Predicate` interface as well. In order to be used they should be pointed in the class path of the java virtual machine when calling the framework. Application of custom predicates in the class model formulas does not differ from default predicates, apart from the full package name specification needed for resolving the corresponding java implementation.

  ⋆ *easy* to modify

- *decision maker* embodies the final stage of the identity resolution process. It consists of two parts: threshold filter that marks new entities as not identifiable and fusion component that combines attributes of the identical entities and resolves possible conflicts.

  The threshold filter applies a pre-configured threshold to the similarity score of two entities. Its value is taken from the framework stings therefore its modification is easy.

  The fusion process describes specific behaviour for each of the attributes of the compared entity type. The algorithms heavily depend on the domain of the application; they are implemented separately for each application. Therefore in order to change the behaviour of the component, one needs to implement

the corresponding interface and to write a new piece of code that applies the appropriate strategies for resolving conflicts. Providing configurable implementation for some common strategies is planed as future work on the framework.

⋆ *easy* to modify - threshold filter ⋆ *moderately easy* to modify - fusion component

In order to evaluate formally the maintainability of the IdRF we will "change scenario evaluation" technique proposed by [Bengtsson *et al.* 04]. The authors suggest using three steps for evaluating the effect of applying new scenario on a software system. This process corresponds to the *changes in the requirement* part of ISO 9126 standard definition. The three steps are:

- Identify the affected components
- Determine the effect on the components
- Determine ripple effects

We consider a very general scenario of using the framework - identification of a new type of entities. Any new identification task is a subject of research and development in a context of new domain, project or application. It can be however assisted by reusable software components and procedures implemented in the framework. Thus we will use different identity criteria and various combinations of the existing once. We will measure the maintainability of the framework in terms effort needed for implementation of new application domains.

The process should start from analysing possible data sources and the appropriate data representation. The goal of the schema alignment step would be the objects of identification to be described against an ontology. It appropriate ontology for the chosen domain does not exist, it should be created. Once the object and data sources are prepared for processing, then the IdRF can be used for identity resolution.

### Identify the affected components

The IdRF architecture is component based, thus changes in the framework are actually possible as changes in its components. Applying a new scenario challenges their functionality. We consider a new type of objects for identification and changes in the ontology schema. They will reflect in modification of several components:

- the new ontology should be loaded into the *semantic repository*

- the *pre-filtering component* needs to be configured to apply new candidate selection constrains
- the *evidence collector* should be feeded with a similarity measure function definition and possible with new algorithms for attribute similarity calculation
- the *decision maker's* threshold has to be tuned. If one expects any conflicts in the entity values, a new fusion algorithm should be implemented

### Determine the effect on the components

The maintainability of the framework is determined by the effect of the new scenario on each of the components. The ability to correspond to the new requirements shows the overall capacity of changes of the framework. We have already discussed and evaluated the degree of modification capability of each component independently. The role of the scenario is to show how they combine and what the average maintainability cost for using entire IdRF is.

The effect for each of the components can be defined as follows:

- *semantic repository* - change in the configuration only - *easy*
- *pre-filtering* - change in the selection constrains - *easy*
- *evidence collection* - add and configure new predicates - *easy*
- *decision maker* - implement new fusion procedure - *moderately easy*

Three out of four affected components are *easy* to be modified according to the rating schema we use. The fourth one is *moderately easy* to be modified. Therefore we can conclude that IdRF as a whole is easily maintainable.

### Determine ripple effects

The third step is to determine the ripple effects of the modifications. This is however very difficult to be estimated. Basically every change of one component behaviour may lead to compromising the results of its work. The errors will be then propagated through the other components called in the dataflow. Here we make an optimistic assumption that modifications in components will be made after detailed investigation of the performance at each step. Since all components of the framework are already specified as affected (except the SDCE that is not a subject of modification as discussed above), we can suggest that there would be no ripple effects caused by the new modifications.

### Modification for company profile scenario

In this work we discuss two usecase: Job offers collection and Company profile collection. The first one served as motivation scenario and testing bed for IdRF implementation, while the second one was build over the existing infrastructure modifying the domain specific components of the first usecase. This modification required changes in the configuration of several components.

Initially the preparation starts with schema alignment as discussed in Section 3.1.3. It includes manual definition of mapping between relational database model used by one of the sources to the domain ontology. This step requires basic understanding of the domain and both schemas, and it took not more then few working hours. Once performed has been made available to the corresponding IdRF application as set of *semantic repository* configuration parameters.

The main subject of customisation in this usecase are algorithms used for *pre-filtering* and *evidence collection*. They are implemented as a new predicates (called *OrganizationCombine*) and extend the pool of predicated available to the IdRF application.

As par of the *pre-filtering*, it reflects the requirements for data preparation and selection as described in details in Section 3.2.3 including the corresponding data normalisation procedures. Being used as one of the predicates in the corresponding Class Model definition, it takes part in building restriction keys for candidate selection. Concrete implementation and usage is given as an example in Section 5.3. Data preparation and normalisation algorithms are implemented from scratch as part of the *pre-filtering* customisation, since they does not reply any of the normalisation procedures used in the Job offers collection scenario. The candidate selection, however, is based on simple configuration of the attributes and reuses the result of data normalisation e.g. company name tokenisation. This step requires deep understanding and investigation of the domain in order to obtain the most suitable restrictions and data transformations and took several days of experiments.

The main effort in customising *evidence collection* step is put in choosing the most suitable combination of attributes and similarity measures. A comprehensive explanation of the selection of attributes and definition of algorithms is given in Section 4.1.4. Like in pre-filtering, domain analysis plays major role in the process of setting evidence collection parameters. Once chosen, the new similarity metrics for company names comparison are implemented as part of the new predicate *OrganizationCombine* , while the rest of the attributes are compared using IdRF built-in *StrictSameAttribute* predicate. Final implementation and configuration of the evidence collection step in IdRF application of this usecase took just few days, however we spent a couple of week for a domain research preceding the implementation.

The last component for modification in the IdRF is the *decision maker*. The analysis of what is needs to be changed here is given in Section 4.2.2. The decision threshold is provided as part of the module configuration, while the fusion strategy is implemented from scratch. The total time spent for customising this module is a couple of days.

In this section we clarified "maintainability" aspect of evaluation. We showed that it corresponds to the capability of the software to be modified and evaluated the possible degree of modification in a general scenario. Based on the conclusion that the proposed IdRF is relatively easy to be modified, we can advocate the proposed approach and its implementation in the frameworks are **easily maintainable**.

# Chapter 7

# Conclusion and Future work

In this work we presented our approach to the identity resolution problem, which arises wherever heterogeneous sources contain multiple references to one and the same real world object[1]. The main goal of the work was to construct a very general approach to identity resolution that is applicable to structured as well as to textual sources.

We proposed a four-step mechanism that starts with *schema alignment* of incoming data sources, including processing of textual sources to extract relevant information. We outlined the advantages of using a rich semantic knowledge representation language, which is suitable for modelling standard database schemas as well as object oriented data models. The main benefits from using ontologies as a conceptual model are: schema flexibility and the possibility for extension; multidimensional data model that allows tracking of relations; the existence of standardised description and query languages.

As a second step we defined *candidate selection* which includes pre-processing and normalisation of data. It aims at discarding those entities that are clearly very different from those that they are compared to, so they cannot be identity candidates. Our algorithms for restricting the pool of candidates are computationally cheap and filter only clearly non-matching entities. Before applying them the incoming data is parsed and normalised in order to eliminate possible variations in representing the same information. This, however, is possible only to a certain extent, but again saves some time when compared to executing detailed comparison algorithms.

The main evidence for identity of two entities comes from applying *similarity measures*. This is the third step in the process, which compares attribute values of two

[1]This problem is related to "record linkage", however it extends the notion of record to any object that can be formally described.

entities and returns an evaluation score quantifying their similarity. Our approach allows for using custom measures for each type of attribute as well as some of those previously proposed in related research. We also introduced a mechanism for combining the scores of different attributes into a single result using logical operators.

The last step in the identity resolution process is *data fusion* or combining the values from two entities that have been found to be identical into a single object that will be stored back to the knowledge repository. It starts by choosing the entity pair whose similarity is sufficient evidence for identity. Then we outlined several conflict resolution strategies that can be applied on the attribute level. Finally the new values for the enriched entity are stored in the knowledge base. In this way they are made directly available for the next iteration of the resolution process.

We implemented these four steps in an IDentity Resolution Framework and applied them in two use-cases. Our final evaluation showed that this approach allows for obtaining high accuracy in resolving identity. We compared our results to another similar system that has been reported recently in the literature and showed that our approach is more efficient in terms of time and speed when processing large numbers of entities. Finally we investigated maintainability and showed that various modifications are very easy to implement in the chosen framework.

A major contribution in our work is our investigation of the usage of a rich semantic knowledge representation that allows for flexible and unified interpretation during the process. Thus we are not restricted in the type of information that can be processed (although we focussed mainly on information extracted from text). Further we indicated the importance of each of the main stages of the identity process, formalising and enriching previously known techniques in various related domains. We suggested a rule based approach for customisation in each of the steps and introduced logical operators and their interpretation during the process.

## 7.1 Future Work

The identity resolution framework is a software prototype that implements the identity resolution approach described in this work. There are many possibilities for extending and improving its components. So far it does not support the very first stage of the identity resolution process, namely schema matching, and only a basic mapping between database schemas and entity descriptions is supported. Extending the set of predefined predicates which apply different similarity measures will also improve the usability of the platform.

When working with real world objects, there are certain facts that are true only at a certain period of time (e.g. a person holding a position in an organisation), while

others does not change (e.g. birthdate). Although our work is not focussed on dealing with temporal nature of entity attributes, this can be taken into account during attributes value the comparison. Unfortunately RDF and OWL specifications do not yet provide standard mechanism for coding temporal information. Thus one should introduce a custom mechanism for reflecting temporality and customise similarity metrics used for comparison entity attributes accordingly. This we see as a promising direction for future work.

In order to test more scenarios for identification we can foresee two completely different domains where the proposed approach can be implemented:

- News articles, which report large numbers of facts and where various sources provide similar information, presented in many different ways. The big news agencies provide us with large amounts of experimental data.
- Semantic descriptions of web services, which describe the specificity of semantic web services for purposes of facilitating discovery, composition and so on. Automatic composition of services depends on correct recognition of which services achieve similar goals.

Working on news articles is a more general problem than the extraction and resolution of job vacancies because of the wide variety of fact types. The knowledge model would be more complex and the fusion of facts correspondingly more challenging. This domain would be very interesting for our experiments mostly because the likelihood of a large amount of redundancy, with the same event reported by different sources (i.e. news agencies, on-line news papers, etc.). Experiment here could be based on the corpus collected by the News Collector [2]. Another product of Ontotext Lab (the KIM Platform) already implements basic Information Extraction techniques for this domain and could be used as a base for further experiments and evaluations.

We also plan a number of other experiments that will explore domains other than the purely textual.

Another direction for future work is in combining similarity of different entity attributes with contextual information in order to build a total similarity measure. It is often the case that clues to identity are indirect – derived from the context where the fact appears as well as from external sources (e.g. co-occurring entities, web search hits, etc.).

[2] A product of Ontotext Lab, News Collector is a web service that collects and provides instant access to the articles from the top-10 on-line news sources. Available at http://news.ontotext.com or read more at http://www.ontotext.com/products/NewsCollector.html.

In this work we presented a rule-based approach for combining similarity evidence, however reflecting the relevance of this indirect evidence in manual processes can become an extremely difficult and even impossible task. Therefore, in connection with this work we aim at evaluating several machine learning techniques in order to assist setting up the decision process. The expected benefit from using (semi) automatic tuneable decision criteria is decreasing the tuning cost (in relation to the required expertise). The decision maker module chooses the candidate favoured either by the manually produced set or by the learned class model. Hence, the generality of the module is guaranteed while the class models can be easily improved and replaced.

# Appendix A

# Standard Predicates in IdRF

This appendix contains the corresponding java code of some of the system predicates that are implemented in IdRF. They can be grouped in correspond to the logical operators :

- system predicates corresponding to the logical operators as presented in Chapter 3:
  - AndPredicate, NotPredicate, OrPredicate and ImplicationPredicate
- subsidiary system predicates used in class model (see Section 5.2):
  - ConstPredicate, SuperPredicate and VariablePredicate
- example of a full power core predicate:
  - SameAliasPredicate

---

```
 /** General interface for implementing a particular
  * similarity measure
  */
public interface Predicate {

    /** the algorithm for measuring the similarity of two entities
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return similarity score
     */
```

```
    public double eval(EvalContext context, EntityDescription descr1,
                                            EntityDescription descr2);

    /**
     *  selects attribute values for queering for similar entities
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */

    public Condition prepareQuery(
      EvalContext context, String targetVar, EntityDescription descr,
                                                boolean isImportant);
}
```

---

```
package com.ontotext.idrf.sdce.predicate;

import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;
import com.ontotext.kim.client.entity.EntityDescription;

public class AndPredicate implements Predicate {

  Predicate m1, m2;

    /**
     * system predicate corresponding to the logical operator 'and'
     * @param m1 - predicate 1
     * @param m2 - predicate 2
     */
    public AndPredicate(Predicate m1, Predicate m2) {
        this.m1 = m1;
        this.m2 = m2;
    }
     /**
```

```
     * combines similarity score from predicate 1 with score from
     * predicate 2 as d1*d2
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return similarity score
     */
  public double eval(EvalContext context, EntityDescription descr1,
                                          EntityDescription descr2){
        double d1 = m1.eval(context, descr1, descr2);
        double d2 = m2.eval(context, descr1, descr2);
        return d1 * d2;
  }
    /**
     * combines query restrictions from predicate 1 and predicate 2
     * in AndCondition
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
  public Condition prepareQuery(EvalContext context, String targetVar,
                        EntityDescription descr, boolean isImportant){

    Condition cond1 =
        m1.prepareQuery(context, targetVar, descr, isImportant);
    Condition cond2 =
        m2.prepareQuery(context, targetVar, descr, isImportant);
    if ( cond1 != null && cond2 != null )
        return new AndCondition(cond1, cond2);

    if ( cond1 != null ) return cond1;
    if ( cond2 != null ) return cond2;
    return null;
  }
}
```

```
package com.ontotext.idrf.sdce.predicate;

import com.ontotext.kim.client.query.*;
import com.ontotext.kim.client.entity.*;
import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;

public class NotPredicate implements Predicate {
  Predicate m;

    /**
     * system predicate corresponding to the logical operator 'not'
     * @param m - predicate
     */
 public NotPredicate(Predicate m){
   this.m = m;
 }
    /**
     * reduces the similarity score from the predicate as
     * (1-predicate score)
     * @param context - the context where the predicate is called
     * @param descr1  - source entity description
     * @param descr2  - target entity description
     * @return similarity measure
     */
 public double eval(EvalContext context, EntityDescription descr1,
                                          EntityDescription descr2){
   return (1 - m.eval(context,descr1,descr2));
 }

    /**
     * neglects the condition of the predicate
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr1
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
 public Condition prepareQuery(EvalContext context, String targetVar,
```

```
                    EntityDescription descr1, boolean isImportant){
   Condition condition =
               m.prepareQuery(context, targetVar, descr1, false);
   return new NotCondition(condition);
 }
}
```

---

```
package com.ontotext.idrf.sdce.predicate;
import com.ontotext.kim.client.query.*;
import com.ontotext.kim.client.entity.*;
import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;

public class OrPredicate implements Predicate {
  Predicate m1, m2;

   /** system predicate corresponding to the logical operator 'or'
     * @param m1 - predicate 1
     * @param m2 - predicate 2
     */
  public OrPredicate(Predicate m1, Predicate m2){
    this.m1 = m1;
    this.m2 = m2;
  }
    /**combines similarity score from predicate 1 with score from
     * predicate 2 as (d1 + d2 - d1*d2)
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return similarity score
     */
  public double eval(EvalContext context, EntityDescription descr1,
                                          EntityDescription descr2){
    double d1 = m1.eval(context,descr1,descr2);
    double d2 = m2.eval(context,descr1,descr2);
    return (d1 + d2 - d1*d2);
  }
```

```
    /* combines query restrictions from predicate 1 and predicate 2
     * in OrCondition
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
  public Condition prepareQuery(EvalContext context, String targetVar,
                          EntityDescription descr, boolean isImportant)
  {
    Condition cond1 =
            m1.prepareQuery(context, targetVar, descr, false);
    Condition cond2 =
            m2.prepareQuery(context, targetVar, descr, false);
    return new OrCondition(cond1,cond2);
  }
}
```

---

```
package com.ontotext.idrf.sdce.predicate;
import com.ontotext.kim.client.query.*;
import com.ontotext.kim.client.entity.*;
import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;

public class ImplicationPredicate implements Predicate {
  Predicate m1, m2;

    /**system predicate corresponding to the logical operator '=>'
     * @param m1 - predicate 1
     * @param m2 - predicate 2
     */
  public ImplicationPredicate(Predicate m1, Predicate m2)
  {
    this.m1 = m1;
    this.m2 = m2;
  }
    /**combines similarity score from predicate 1 with score from
```

```
     * predicate 2 as (1 - d1 + d1*d2)
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return similarity measure
     */
  public double eval(EvalContext context, EntityDescription descr1,
                                          EntityDescription descr2){
    double d1 = m1.eval(context,descr1,descr2);
    double d2 = m2.eval(context,descr1,descr2);
    return (1 - d1 + d1*d2);
  }

    /**combines query conditions from predicate 1 and predicate 2
     * as "not(condition1) or condition2"
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
  public Condition prepareQuery(EvalContext context, String targetVar,
                          EntityDescription descr, boolean isImportant)
  {
    Condition cond1 =
            m1.prepareQuery(context, targetVar, descr, false);
    Condition cond2 =
            m2.prepareQuery(context, targetVar, descr, false);
    return new OrCondition(new NotCondition(cond1), cond2);
  }
}
```

---

```
package com.ontotext.idrf.sdce.predicate;

import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.Condition;
import com.ontotext.kim.client.entity.EntityDescription;
```

```
public class ConstPredicate implements Predicate {
  double value;

    /**
     * system predicated that models using constants
     * @param value - value of the constant
     */
 public ConstPredicate(double value)
 {
   this.value = value;
 }

    /**
     * does not calculate any similarity measure but returns
     * constant value
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return constant value
     */
 public double eval(EvalContext context, EntityDescription descr1,
                                         EntityDescription descr2){
   return value;
 }

    /**
     * an empty method
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return null
     */
 public Condition prepareQuery(EvalContext context, String targetVar,
                      EntityDescription descr, boolean isImportant){
   return null;
 }
}
```

```
package com.ontotext.idrf.sdce.predicate;

import com.ontotext.kim.client.query.*;
import com.ontotext.kim.client.entity.*;
import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;


public class SuperPredicate implements Predicate {

    /**
     * extracts and executes the similarity measure corresponding
     * to the super class of the entities.
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return  similarity score
     */
  public double eval(EvalContext context, EntityDescription descr1,
                                          EntityDescription descr2){
  double dist;
  if (!context.decClassLevel()) return 0;
  dist = context.getCurrentPredicate().eval(context, descr1, descr2);

  if (!context.incClassLevel()) return 0;
  return dist;
  }

    /**
     * builds query condition corresponding to the super class of
     * the entities.
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
  public Condition prepareQuery(EvalContext context, String targetVar,
                        EntityDescription descr, boolean isImportant)
```

```
  {
    if (!context.decClassLevel())
      return null;

    Condition condition = context.getCurrentPredicate().
                prepareQuery(context, targetVar, descr, isImportant);

    if (!context.incClassLevel())
      return null;

    return condition;
  }
}
```

---

```
package com.ontotext.idrf.sdce.predicate;

import com.ontotext.kim.client.query.*;
import com.ontotext.kim.client.entity.*;
import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;

public class VariablePredicate implements Predicate {
  String varName;
  Predicate varMetric;
  Predicate subMetric;

    /**
     * system predicate that defines complex predicate as variable
     * and executes the referred predicate
     * @param varName
     * @param varMetric
     * @param subMetric
     */
  public VariablePredicate(String varName, Predicate varMetric,
                                           Predicate subMetric){
    this.varName   = varName;
    this.varMetric = varMetric;
```

```
  this.subMetric = subMetric;
}

  /**
   * calculates the similarity score of the referred predicate
   * @param context - the context where the predicate is called
   * @param descr1 - source entity description
   * @param descr2 - target entity description
   * @return similarity score
   */
public double eval(EvalContext context, EntityDescription descr1,
                                        EntityDescription descr2){
  double dist, varVal;

  varVal = varMetric.eval(context, descr1, descr2);

  context.pushVariableValue(varName, new Double(varVal));
  dist = subMetric.eval(context, descr1, descr2);
  context.popVariableValue();

  return dist;
}

  /**
   * builds a query condition of the referred predicate
   * @param context - the context where the predicate is called
   * @param targetVar - associated variable in the query
   * @param descr1 - source entity description
   * @param isImportant - false value indicates optional condition
   * @return query condition
   */
public Condition prepareQuery(EvalContext context, String targetVar,
                     EntityDescription descr1, boolean isImportant)
{
  Condition varCondition = varMetric.prepareQuery(context, targetVar,
                                                  descr1, isImportant);
  context.pushVariableValue(varName, varCondition);
  Condition condition = subMetric.prepareQuery(context, targetVar,
                                               descr1, isImportant);
```

```
    context.popVariableValue();
    return condition;
  }
}
```

---

```
package com.ontotext.idrf.sdce.predicate;

import java.util.*;

import com.ontotext.kim.client.entity.EntityDescription;

import com.ontotext.idrf.sdce.impl.*;
import com.ontotext.idrf.kb.querymodel.*;
import org.openrdf.model.Literal;
import org.openrdf.model.URI;



public class SameAliasPredicate implements Predicate {
     URI m_attrURI;

    /**
     * evaluates equality of a given entity attribute
     * @param attrURI - uri of the entity attribute
     */
    public SameAliasPredicate( URI attrURI) {
        m_attrURI = attrURI;
    }

    /**
     * compares equality of the corresponding values of
     * a given attribute in both entity descriptions
     * @param context - the context where the predicate is called
     * @param descr1 - source entity description
     * @param descr2 - target entity description
     * @return 0 if the attribute values differ;
     *  1 if the attribute values are equal
```

```
     */
    public double eval(EvalContext context, EntityDescription descr1,
                                            EntityDescription descr2){
        Iterator attrIter1 = descr1.getLabels().iterator();
        if (attrIter1 == null)
            return 0.0;

        while (attrIter1.hasNext()) {
            Object value1 = attrIter1.next();

            Iterator attrIter2 = descr2.getLabels().iterator();
            if (attrIter2 == null)
                return 0.0;

            while (attrIter2.hasNext()) {
                Object value2 = attrIter2.next();
                if (value1.equals(value2))
                    return 1.0;
            }
        }

        return 0.0;
    }



/**
     * builds strict equality condition for using the corresponding
     * value of the given attribute
     * @param context - the context where the predicate is called
     * @param targetVar - associated variable in the query
     * @param descr - source entity description
     * @param isImportant - false value indicates optional condition
     * @return query condition
     */
    public Condition prepareQuery(EvalContext context, String targetVar,
                          EntityDescription descr, boolean isImportant){
        Condition condition = null;
        PathPattern pattern = context.
```

```
                addPathPattern(targetVar, m_attrURI, null, !isImportant);
        for (Literal value : descr.getLabels()) {
            EqCondition eq_condition =
               new EqCondition(pattern.getObjectVar(), m_attrURI, value);

            if (condition == null) condition = eq_condition;
            else
             condition = new OrCondition(condition, eq_condition);
        }
        return condition;
    }
}
```

# Bibliography


[ACE04]
*Annotation Guidelines for Entity Detection and Tracking (EDT)*, Feb 2004. Available at http://www.ldc.upenn.edu/Projects/ACE/.

[Arenas *et al.* 99]
M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '99)*, pages 68–79, New York, NY, USA, 1999. ACM.

[ARPA 93]
Advanced Research Projects Agency. *Proc. of the Fifth Message Understanding Conference (MUC-5)*. Morgan Kaufmann, California, 1993.

[Aswani *et al.* 06]
N. Aswani, K. Bontcheva, and H. Cunningham. Mining information for instance unification. In *Proc. of the International Semantic Web Conference (ISWC'06)*, 2006.

[Bagga & Baldwin 98]
A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In C. Boitet and P. Whitelock, editors, *Proc. of the Thirty-Sixth Annual Meeting of the ACL and Seventeenth International Conference on Computational Linguistics*, pages 79–85, San Francisco, California, 1998. Morgan Kaufmann Publishers.

[Bagga & Biermann 00]
A. Bagga and A. Biermann. A methodology for cross-document coreference. In *Proc. of the Fifth Joint Conference on Information Sciences*, pages 207–210, 2000.

[Baxter *et al.* ]
R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking

methods for record linkage. In *Proc. of the 2003 ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation.*

[Beckett 04]
D. Beckett, editor. *RDF/XML Syntax Specification (Revised).* W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[Bengtsson *et al.* 04]
P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet. Architecture-level modifiability analysis (a l m a). *Journal of Systems and Software*, 69(1-2):129–147, 2004.

[Benjelloun *et al.* 09]
O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, 2009.

[Berners-Lee 06]
T. Berners-Lee. Linked data - design issues, Retrieved July 23, 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[Berners-Lee *et al.* 09]
T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax, January 2005, Retrieved June 14, 2009. http://tools.ietf.org/html/rfc3986.

[Bilenko & Mooney 03]
M. Bilenko and R. J. Mooney. Employing trainable string similarity metrics for information integration. In *Proc. of the IJCAI-2003 Workshop on Information Integration on the Web*, pages 67–72, Acapulco, Mexico, August 2003.

[Biron & Malhotra 01]
P. V. Biron and A. Malhotra, editors. *ML Schema Part 2: Datatypes.* W3C Recommendation, 2 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[Bizer *et al.* 09]
C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[Bleiholder & Naumann 06]
J. Bleiholder and F. Naumann. Conflict handling strategies in an inte-

grated information system. In *In Proc. of the International Workshop on Information Integration on the Web(IIWeb)*, 2006.

[Bontcheva *et al.* 09]
K. Bontcheva, B. Davis, A. Funk, Y. Li, and T. Wang. Human language technologies. In J. Davies, M. Grobelnik, , and D. Mladenic, editors, *Semantic Knowledge Management. Integrating Ontology Management, Knowledge Discovery, and Human Language Technologies*, pages 37–49. Springer Berlin Heidelberg, 2009.

[Bray *et al.* 00]
T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, editors. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation, 6 October 2000. http://www.w3.org/TR/2000/REC-xml-20001006/.

[Brickley & Guha 04]
D. Brickley and R. V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[Castano *et al.* 08]
S. Castano, A. Ferrara, S. Montanelli, and D. Lorusso. Instance matching for ontology population. In *Proc. of the Sixteenth Italian Symposium on Advanced Database Systems (SEBD)*, 2008.

[Chinchor 98]
N. Chinchor. Overview of muc-7. In *Proc. of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[Christen 07]
P. Christen. Towards parameter-free blocking for scalable record linkage. Technical report, TR-CS-07-03, Australian National University, Department of Computer Science (DCS), 2007.

[Christen 08]
P. Christen. Febrl - an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*, pages 1065–1068, New York, NY, USA, 2008. ACM.

[Cimiano *et al.* 03]
P. Cimiano, S.Staab, and J. Tane. Automatic acquisition of taxonomies from text: Fca meets nlp. In *Proc. of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining*, pages 10–17, Cavtat-Dubrovnik, Croatia, 2003.

[Cohen & Richman 02]
W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*, pages 475–480, New York, NY, USA, 2002. ACM.

[Cohen 00]
W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Trans. Inf. Syst.*, 18(3):288–321, 2000.

[Cohen *et al.* 03]
W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string metrics for matching names and records. In *Proc. of the Workshop on Data Cleaning, Record Linkage, and Object Consolidation (KDD-2003)*, Washington DC, August 2003.

[Collier & Takeuchi 02]
N. Collier and K. Takeuchi. Pia-core: Semantic annotation through example-based learning. In *Proc. of the Third International Conference on Language Resources and Evaluation (LREC2002)*, Las Palmas, Spain, 2002.

[Collier *et al.* 03]
N. Collier, K. Takeuchi, and A. Kawazoe. Open ontology forge: An environment for text mining in a semantic web world. In *Proc. of the International Workshop on Semantic Web Foundations and Application Technologies*, Nara, Japan, March 2003.

[Cunningham 05]
H. Cunningham. Information Extraction, Automatic. *Encyclopedia of Language and Linguistics, 2nd Edition*, pages 665–677, 2005.

[Cunningham *et al.* 02]
H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. Gate a framework and graphical development environment for robust nlp tools and applications. In *Proc. of the Anniversary Meeting of the Association for Computational Linguistics (ACL 02)*, Philadelphia, US, 2002.

[Dalvi *et al.* 09]
N. Dalvi, R. Kumar, B. Pang, and A. Tomkins. Matching reviews to objects using a language model. In *Proc. of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP '09)*, pages 609–618, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[Davies & Weeks 04]
J. Davies and R. Weeks. Quizrdf: Search technology for the semantic web. In *Proc. of the 37th Hawaii International Conference on System Sciences (HICSS-37 2004)*, Big Island, HI, USA, January 2004.

[Davis & Goadrich 06]
J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proc. of the 23rd International Conference on Machine Learning (ICML '06)*, pages 233–240, New York, NY, USA, 2006. ACM.

[Declerck *et al.* 03]
T. Declerck, H. Cunningham, H. Saggion, J. Kuper, D. Reidsma, and P. Wittenburg. Mumis – advanced information extraction for multimedia indexing and searching. In *Proc. of the 4th European Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2003)*, London, UK, 2003.

[Dill *et al.* 03]
S. Dill, N. Eiron, D. Gibson, D. Gruhl, and R. Guha. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proc. of The Twelfth International World Wide Web Conference (WWW 2003)*, Budapest, Hungary, 2003.

[Domingue *et al.* 04]
J. Domingue, M. Dzbor, and E. Motta. Magpie: Supporting browsing and navigation on the semantic web. In N. Nunes and C. Rich, editors, *Proc. of the ACM Conference on Intelligent User Interfaces (IUI)*, page 191 197, 2004.

[EAG95]
Evaluation of natural language processing systems. Technical Report EAG-EWG-PR.2, EAGLES, September 1995.

[Elfeky *et al.* 02]
M. Elfeky, V. Verykios, and A. Elmagarmid. Tailor: A record linkage tool box. In *Proc. of the 18th International Conference on Data Engineering (ICDE '02)*, page 17, Washington, DC, USA, 2002. IEEE Computer Society.

[Elmagarmid *et al.* 07]
A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record. detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, 2007.

[Enderton 72]
H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, 1972.

[Etzioni *et al.* 04]
O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yate. Web-scale information extraction in knowitall. In *Proc. of the Thirteenth International Wirld Wide Web Conference (WWW 2004)*, New York City, 2004.

[Feldman *et al.* 05]
S. Feldman, J. Duhl, J. R. Marobella, and A. Crawford. The hidden cost of information work. Technical Report White paper, IDC, March 2005.

[Fernandez *et al.* 06]
N. Fernandez, J. M. Blazquez, J. A. Fisteus, L. Sanchez, M. Sintek, A. Bernardi, M. Fuentes, A. Marrara, and Z. Ben-Ashe. News: Bringing semantic web technologies into news agencies. In *Proc. of the International Semantic Web Conference (ISWC'06)*, 2006.

[Fiscus *et al.* 98]
J. G. Fiscus, G. Doddington, J. S. Garofolo, and A. Martin. Nist's 1998 topic detection and tracking evaluation (tdt2). In *Proc. of the DARPA Broadcast News Workshop*, Virginia, US, 1998.

[Funk *et al.* 07]
A. Funk, D. Maynard, H. Saggion, and K. Bontcheva. Ontological integration of information extraction from multiple sources. In *Proc. of the International Workshop on Multi-source, Multi-lingual Information Extraction and Summarisaton*, 2007.

[Gal *et al.* 04]
A. Gal, G. A. Modica, and H. M. Jamil. Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources. In *Proc. of the 20th International Conference on Data Engineering*, page 853. IEEE Computer Society, 2004.

[Genesereth & Nilsson 87]
M. R. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[Giuliano & Gliozzo 08]
C. Giuliano and A. Gliozzo. Instance-based ontology population exploiting

named-entity substitution. In *COLING '08: Proceedings of the 22nd International Conference on Computational Linguistics*, pages 265–272, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[Giunchiglia *et al.* 04]
F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *Proc. of the 1st European Semantic Web Symposium (ESWC'04*, pages 61–75, 2004.

[Goiser & Christen 06]
K. Goiser and P. Christen. Towards automated record linkage. In *Proc. of the fifth Australasian Conference on Data mining and analystics (AusDM '06)*, pages 23–31, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[Gooi & Allan 04]
C. J. Gooi and J. Allan. Cross-document coreference on a large scale corpus. In *Proc. of the Human Language Technology Conference*, Boston, 2004.

[Götz & Suhre 04]
T. Götz and O. Suhre. Design and implementation of the uima common analysis system. *IBM Syst. J.*, 43(3):476–489, 2004.

[Grishman & Sundheim ]
R. Grishman and B. Sundheim. Message understanding conference 6. a brief history. In *Proc. of International Conference on Computational Linguistics (COLING-96)*, pages 466–471.

[Grishman 97]
R. Grishman. Information extraction: Techniques and challenges. In *Proc. of the International Summer School on Information Extraction (SCIE '97)*, pages 10–27, London, UK, 1997. Springer-Verlag.

[Gruber 95]
T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies.*, 43(5-6):907–928, 1995.

[Gu *et al.* 03]
L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. Technical report, CSIRO Mathematical and Information Sciences, 2003.

[Guha & McCool ]
R. Guha and R. McCool, editors. *The TAP knowledge base.* http://tap.stanford.edu/.

[Hahn & Schnattinger 98]
U. Hahn and K. Schnattinger. Towards text knowledge engineering. In *Proc. of 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 524–531, Menlo Park, CA, 1998. MIT Press.

[haiDo & Rahm 02]
H. hai Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB '02)*, pages 610–621, 2002.

[Hamming 50]
R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 26(2):147–160, 1950.

[Handschuh *et al.* 02]
S. Handschuh, S. Staab, and F. Ciravegna. S-cream semi-automatic creation of metadata. In A. Gomez-Perez, editor, *Proc. of the 13th International Conference on Knowledge Engineering and Management (EKAW 2002)*. Springer Verlag, 2002.

[Hassell *et al.* 06]
J. Hassell, B. Aleman-Meza, and I. B. Arpinar. Ontology-driven automatic entity disambiguation in unstructured text. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *Proc. of the International Semantic Web Conference (ISWC'06)*, volume 4273 of *LNCS*, pages 44–57. Springer, 2006.

[Hernandez *et al.* 95]
M. Hernandez, M. A. Hern'andez, and S. Stolfo. The merge/purge problem for large databases. In *In Proc. of the ACM SIGMOD Conference*, pages 127–138, 1995.

[Hirschman 98]
L. Hirschman. The evolution of evaluation: Lessons from the message understanding conferences. pages 281–305, 1998.

[Iosif & Ygge 01]
V. Iosif and F. Ygge. *On-To-Knowledge: EnerSearch Virtual Organisation Case Study, Deliverable 28, ONToKnowledge project.* December 2001. Available at http://www.ontoknowledge.org.

[Jaffri *et al.* 07]
A. Jaffri, H. Glaser, and I. Millard. Uri identity management for semantic web data integration and linkage. In *Proc. of the 3rd International Workshop On Scalable Semantic Web Knowledge Base Systems*, Portugal, 2007.

[Jones & Galliers 96]
K. S. Jones and J. Galliers, editors. *Evaluating Natural Language Processing Systems,*. Lecture Notes in Artificial Intelligence 1083, Springer-Verlag, 1996.

[Kiryakov *et al.* 05]
A. Kiryakov, D. Ognyanov, and D. Manov. Owlim a pragmatic semantic repository for owl. In *Proc. of International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005) atWISE)*, New York City, USA, 2005.

[Kiryakov *et al.* 09]
A. Kiryakov, Z. Tashev, D. Ognyanoff, R. Velkov, V. Momtchev, B. Balev, and I. Peikov. Validation goals and metrics for the larkc platform. Technical report, LarKC project deliverable D5.5.2. http://www.larkc.eu/deliverables/, 2009.

[Klein *et al.* 07]
M. C. Klein, P. Mika, and S. Schlobach. Approximate instance unification using roughowl: Querying with similarity in openacademia. In *Proc. of the Workshop on Uncertainty Reasoning for the Semantic Web (URSW)*, 2007.

[Klyne & Carroll 04]
G. Klyne and J. J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[Köpcke & Rahm 10]
H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.

[Kounev & Buchmann 03]
S. Kounev and A. Buchmann. Performance modeling and evaluation of large-scale j2ee applications. In *Proc. 29th International Computer Measurement Groupt (CMG) Conference*, 2003.

[Kousha & Thelwall 07]
K. Kousha and M. Thelwall. Google scholar citations and google web-url citations. a multi-discipline exploratory analysis. *Journal of the American Society for Information Science and Technology*, 58(7):1055–1065, 2007.

[Lawrence *et al.* 99]
S. Lawrence, C. L. Giles, and K. D. Bollacker. Autonomous citation matching. In *Proc. of the Third Annual Conference on Autonomous Agents (AGENTS '99)*, pages 392–393, New York, NY, USA, 1999. ACM.

[Lehnert *et al.* 92]

W. Lehnert, D. Fisher, J. McCarthy, E. Riloff, and S. Soderland. university of massachusetts: Muc-4 test results and analysis. In *Proc. of the Fourth Message Understanding Conference (MUC-4)*, pages 151–158, 1992.

[Leit *et al.* 07]

L. Leit ao, P. Calado, and M. Weis. Structure-based inference of xml similarity for fuzzy duplicate detection. In *Proc. of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM '07)*, pages 293–302, 2007.

[Levenshtein 66]

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[Maedche *et al.* 02]

A. Maedche, G. Neumann, and S. Staab. Bootstrapping an ontologybased information extraction system. 2002.

[Maynard 05]

D. Maynard. Benchmarking ontology-based annotation tools for the semantic web. In *Proc. of the Fourth All Hands Meeting (AHM 2005)*, Nottingham, UK, September 2005.

[Maynard *et al.* 05]

D. Maynard, M. Yankova, A. Kourakis, and A. Kokossis. Ontology-based information extraction for market monitoring and technology watch. In *Proc. of ESWC Workshop "End User Apects of the Semantic Web"*, Heraklion, Crete, 2005.

[Maynard *et al.* 06]

D. Maynard, W. Peters, and Y.Li. Metrics for evaluation of ontology-based information extraction. In *Proc. of WWW 2006 Workshop on "Evaluation of Ontologies for the Web" (EON 2006)*, Edinburgh, Scotland, 2006.

[Maynard *et al.* 08]

D. Maynard, Y. Li, and W. Peters. Nlp techniques for term extraction and ontology population. In *Proceeding of the 2008 conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, pages 107–127, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[McCallum & Wellner 03]

A. McCallum and B. Wellner. Object consolidation by graph partitioning

with a conditionally-trained distance metric. In *Proc. of the ACM KDD-2003 Workshop on DataCleaning, Record Linkage and Object Consolidation*, Washington DC, August 2003.

[McCallum *et al.* 00]
A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[McKoewn & Radev 95]
K. McKoewn and D. R. Radev. Generating summaries of multiple news articles. In *Proc. of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 74–82, Seattle, WA, August 1995.

[Miller 95]
G. A. Miller. Wordnet: a lexical database for english. pages 39 – 41, November 1995.

[Milo & Zohar 98]
T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of the 24rd International Conference on Very Large Data Bases (VLDB '98)*, pages 122–133, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[Monge & Elkan 97]
A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. of the SIGMOD-1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, Tuscon, AZ, May 1997.

[Needleman & Wunsch 70]
S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[Neiling & Muller 01]
M. Neiling and M. Muller. The good into the pot, the bad into the crop. preselection of record pairs for database fusion. In *In Proc. of the First International Workshop on Database, Documents, and Information Fusion*, 2001.

[Neumann *et al.* 97]
G. Neumann, R. Backofen, J. Baur, M. Becker, and C. Braun. An information extraction core system for real world german text processing, 1997.

[Nikolov 06]
A. Nikolov. Fusing automatically extracted annotations for the semantic web. Technical Report kmi-06-12, Knowledge Media Institute, The Open University, UK, July 2006.

[Noy & Musen 01]
N. Noy and M. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proc. of IJCAI 2001 workshop on ontology and information sharing*, pages 63–70, 2001.

[Optimor 07]
M. B. Optimor. Top 100 most powerful brands. Technical Report Available April 23rd in the Financial Times, Millard Brown Optimor, April 2007.

[Philips 90]
L. Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):39–44, December 1990. Accessible at `http://www.cuj.com/documents/s=8038/cuj0006philips/`.

[Philips 00]
L. Philips. The double metaphone search algorithm. *C/C++ Users J.*, 18(6):38–43, 2000.

[Popov *et al.* 04]
B. Popov, A. Kiryakov, D. Ognyanoff, D. Manov, and A. Kirilov. Kim - a semantic platform for information extaction and retrieval. 2004.

[Popov *et al.* 08]
B. Popov, A. Kiryakov, I. Kitchukov, K. Angelov, and D. Kozhuharov. Co-occurrence and ranking of entities based on semantic annotation. 2008.

[Radev 00]
D. R. Radev. A common theory of information fusion form multiple text sources step one: Cross-document structure. In *Proc. of the First Workshop on Discourse and Dialogue (SIGdial)*, pages 74–83, Somerset, NJ, May 2000.

[Radev *et al.* 00]
D. R. Radev, H. Jing, and M. Budzikiwska. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user study. In *Proc. of the ANLP/ANNCL Workshop on Summarization*, pages 23–29, Seattle, WA, May 2000.

[Saggion 07]
H. Saggion. Shef: Semantic tagging and summarization techniques applied to cross-document coreference. In *Proc. of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*, 2007.

[Saggion *et al.* 03]
H. Saggion, J. Kuper, T. Declerck, D. Reidsma, and H. Cunningham. Intelligent multimedia indexing and retrieval through multi-source information extraction and merging. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI 2003)*, Acapulco, Mexico, 2003.

[Salton *et al.* 75]
G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[Sassone 87]
P. Sassone. Cost-benefit methodology for office systems. *ACM Transactions on Office Information Systems*, 5(3):273–289, 1987.

[Sintek *et al.* 01]
M. Sintek, M. Junker, L. van Elst, and A. Abecker. Using information extraction rules for extending domain ontologies - position statement. In *Proc. of IJCAI'2001 Working Notes of theWorkshop on Ontology Learning*, Seattle, Washington, USA, August 2001.

[Smith & Waterman 81]
T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, March 1981.

[Soderland 97]
S. Soderland. Learning to extract text-based information from the world wide web. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 251–254, 1997.

[Taft 70]
R. L. Taft. Aname search techniques. Technical report, Albany, New York, 1970.

[Tao 03]
C. Tao. Schema matching and data extraction over html tables. Unpublished M.Sc. thesis, 2003.

[Tejada *et al.* 01]
S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Inf. Syst.*, 26(8):607–633, 2001.

[Tejada *et al.* 02]

S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*, pages 350–359, New York, NY, USA, 2002. ACM.

[Terziev *et al.* 05]

I. Terziev, A. Kiryakov, and D. Mano. Base upper-level ontology (bulo) guidance. Technical Report Deliverable 1.8.1, SEKT project, UK, July 2005.

[Thor & Rahm 07]

A. Thor and E. Rahm. Moma - a mapping-based object matching system. In *Proc. of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR 2007)*, page 7. 247-258, 2007.

[Valarakos *et al.* 04]

A. G. Valarakos, G. Paliouras, V. Karkaletsis, and G. Vouros. Enhancing ontological knowledge through ontology population and enrichment. In *Engineering Knowledge in the Age of the SemanticWeb*, pages 144–156. Springer Berlin Heidelberg, 2004.

[Vargas-Vera *et al.* 02]

M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. Mnm: Ontology driven semi-automatic and automatic support for semantic markup. In A. Gomez-Perez, editor, *Proc. of the 13th International Conference on Knowledge Engineering and Management (EKAW 2002)*. Springer Verlag, 2002.

[Vernica & Li 09]

R. Vernica and C. Li. Efficient top-k algorithms for fuzzy search in string collections. In *Proc. of the First International Workshop on Keyword Search on Structured Data (KEYS '09)*, pages 9–14, New York, NY, USA, 2009. ACM.

[Vernica *et al.* 10]

R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *Proc. of the 2010 ACM SIGMOD/PODS Conference*. ACM, 2010.

[Volz *et al.* 09]

J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In A. Bernstein, D. R. Karger, T. Heath,

L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *Proc. of the International Semantic Web Conference (ISWC 2009)*, volume 5823, chapter 41, pages 650–665. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Welty & Murdock 06]
C. Welty and J. W. Murdock. Toward knowledge acquisition from information extractio. In *Proc. of the International Semantic Web Conference (ISWC'06)*, volume 4273 of *LNCS*. Springer, 2006.

[Wilks 97]
Y. Wilks. Information extraction as a core language technology. In *Proc. of International Summer School SCIE-97*, 1997.

[Wilson 05]
D. R. Wilson. Name standardization for genealogical record linkage. In *Proc. of the Family History Technology Workshop (FHTW'05)*, 2005.

[Winkler 95]
W. E. Winkler. Matching and record linkage. pages 355–384, 1995.

[Witten *et al.* 94]
I. H. Witten, T. C. Bell, and A. Moffat. *Managing Gigabytes: Compressing and Indexing Documents and Images*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[Yang *et al.* 06]
K. Yang, J. Jiang, H. Lee, and J. Ho. Extracting citation relationships from web documents for author disambiguation. Technical Report TR-IIS-06-017, Institute of Information Science, Academia Sinica Taipei Taiwan, December 2006.