Roma Tre University
Ph.D. in Computer Science and Engineering

Model and Domain Independence: an Experience in Model Management and Information Extraction

Daniele Toti

Model and Domain Independence: an Experience in Model Management and Information Extraction

A Thesis presented by
Daniele Toti
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Department of Computer Science and Automation

March 2012

Advisor:
Prof. Paolo Atzeni

Reviewers:
Prof. Elena Baralis
Prof. Douglas Stott Parker Jr.

"The state of mind which enables a man to do work of this kind is akin to that of the religious worshipper or the lover; the daily effort comes from no deliberate intention or program, but straight from the heart."

Albert Einstein, About the Principles of Research

Abstract

Information systems play a crucial role in handling the flow of information generated, exchanged, acquired and stored nowadays. The strife to pursue model- and domain-independent approaches by prescinding from the specificity of their respective instances is a natural way to keep advancing towards more scalable and interoperable information systems, and therefore contributing to the creation of a more cohesive and productive world. In this regard, this dissertation addresses the problem of model and domain independence applied to two critical elements of the whole lifecycle of information management: the modeling and design phase, and the acquisition and storage phase, respectively exemplified here by the areas of Model Management and Information Extraction. Specifically, in this Thesis we first focus on the topic of model-independent schema and data translations, where we show an extension of a model management operator at both the conceptual and the operative level, by introducing the concepts of inheritance and polymorphism within its underlying data dictionary and the rules used to perform the actual translations. Subsequently, we shift our attention towards the automatic discovery of abbreviations from full-text scientific papers, where we propose a domain-independent methodology to identify and resolve acronyms and abbreviations and match them with entities of a given domain, all within a suitable implementing framework. Eventually, we discuss potential future directions of this research, by introducing the concept of semantic similarity among ontologies and the challenge of automatically build them and align them, in order to both extend the proposed methodology and tackle wider areas of knowledge discovery.

Acknowledgments

This is the third time in my academic career that I find myself facing the ultimate task of expressing my gratitude in a Thesis to those who, one way or another, contributed to its production and to the whole path that led me there. And the list of people to thank grew longer every time, with the constant risk of forgetting someone more or less important. Since every journey is different, though, I will once again try to address all whose support, influence and contribution, whether material, physical or both, have made this achievement possible.

To my family, in other words to my mother Laura, my father Angelo and my uncle Sandro, for being always there, no matter what.

To the other half of my heart and my whole self, Eleonora, for providing me with so much strength, comfort and safety, despite having to deal with me through good and bad times.

To friends both old and new, both distant and near, both lost and found, and however frequently I can see them these days: Adriano, Alberto, Stefano, Cristian, Davide, Marco Passariello, Marco Cipriani, Daniele Tommasi, Laura, Daniele Moccia, Emanuele, Federico, Mauro, Alessandro, Eloisa, Francesco, Pietro, Fabio, Claudio, and all those who crossed paths with me so far.

To my long-time advisor Prof. Paolo Atzeni, for having allowed me to undertake this Ph.D. adventure and for all his patience in bearing with me for so long.

To Prof. Fabio Polticelli, whom I was delighted to work with and whose unparalleled cooperation, support and helpfulness accounted for a great deal in making my doctoral graduation come true.

To the great teachers and mentors encountered throughout my student career, starting from Prof. Luca Cabibbo, Prof. Paolo Merialdo and all the teachers in the Database group, moving towards Prof. Sally Stevens from my English school, Prof. Francesco La Gala, Prof. Licia Capodiferro and the

others from my graduation years, and going back to Prof. Elisabetta Diadori, Prof. Rocco De Maria, Prof. Luca Gentile, Prof. Gabriella Beretta, Prof. Pia Cillo, and all the others I forgot to mention: the person I have become is partly thanks to them and their unforgettable lessons on a variety of subjects, life included.

To the outstanding colleagues-now-turned-into-friends met during my Ph.D. years, that is to say Stefano, Pierluigi and Giorgio, for having welcomed me and welcoming me still among their valued, joyfully gay and crazy ranks.

To my grandfather Antonio, Clara, my grandmother Letizia, Mrs. Speranza and Annamaria, who keep watching over me from above and whose praise I hope to always prove myself worthy of.

To my one and only Lord, for granting me all the uncountable gifts I am blessed with every single day of my life.

To all those who believed in me and in my professional choices, for giving me further motivation to go forward; and to those who did not, for eliciting my courage in treading a different, perhaps harder, but more personally rewarding road. We all know research is no walk in the park and few certainties line up the horizon; nevertheless, I am finally beginning to see a glimpse of my future self, and I realize this is the right thing to do. I am not sure whether we can change the world, but at least we can strive to pursue what is right, for both our professional career and personal life. This is and will be my unyielding trademark from now on.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> *"There is surely nothing other than the single purpose of the present moment. A man's whole life is a succession of moment after moment. There will be nothing else to do, and nothing else to pursue. Live being true to the single purpose of the moment."*
>
> Tsunetomo Yamamoto *Hagakure*

1.1 Background and Motivation

Information management, in its broadest sense, is of paramount importance in our present, heavily-computerized and interconnected world. Information is everywhere, in different forms and natures, and is created, communicated, shared, acquired, transformed and stored on a second-to-second basis. Within this context, information systems play a crucial role in carrying out these tasks as a whole, and the more effective they are, the more easily users can benefit from them, by being properly granted access and handling to the information needed.

Formalisms for representing information are however disparate and often highly divergent from one information system to another, making interoperability an ongoing issue to be constantly addressed. Furthermore, the heterogeneous character of the information itself, and the disparate, specific application domains within which it is originated and managed, bring about relevant customization and scalability problems.

This predicament deeply affects the information management process in its entirety and consequently yearns for solutions characterized by a substantial degree of generality, so that it might be possible to devise effective and efficient systems for handling information, regardless of the specific models or domains.

As a matter of fact, striving to pursue model- and domain-independent approaches by prescinding from the specificity of their respective instances is a natural way to keep advancing towards more scalable and interoperable information systems, and therefore contributing to the creation of a more cohesive and productive world.

In this regard, this dissertation addresses the problem of model and domain independence applied to two critical elements of the whole lifecycle of information management: the modeling and design phase, and the acquisition and storage phase, respectively exemplified here by the areas of Model Management and Information Extraction.

Model Management is an innovative approach to meta-data management that offers a higher abstraction layer upon data conceptualization and design than other existing, scope-restricted techniques. The main abstractions used are models (like schemas, interface definitions, and so forth) and formal descriptions of transformations among models to correctly manipulate them, called mappings. A set of purposefully-designed operators is applied to models and mappings as a whole rather than to their individual elements, in order to handle those transformations in a generic manner and consequently simplify the programming of meta-data applications.

Information Extraction, on the other hand, is a type of information retrieval whose main purpose lies in automatically extracting relevant information from a variety of unstructured or semi-structured sources, and providing the extracted information with an appropriate structure in order for it to be proficiently consumed and stored. The ultimate goal of information extraction systems is therefore to bridge the gap between man and machine in terms of text processing and speed reading, by placing clear semantics upon the disordered context of textual information, and consequently opening up novel frontiers for human-computer interactions.

Specifically, in this Thesis we first focus on the topic of model-independent schema and data translations, where we show an extension of a model management operator at both the conceptual and the operative level, by introducing the concepts of inheritance and polymorphism within its underlying data dictionary and the rules used to perform the actual translations.

Subsequently, we shift our attention towards the automatic discovery of abbreviations from full-text papers, where we propose a domain-independent

methodology — and a framework to implement it — to identify and resolve acronyms and abbreviations and match them with entities of a given domain, given the availability of a domain entity repository. We then speculate about a potential extension of this methodology, by introducing the concept of semantic similarity among ontologies (the latter automatically built from core concepts featured in the input papers), in order to both enhance the proposed abbreviation discovery approach and tackle new research areas altogether.

1.2 Contribution

This Thesis is the result of the work carried out within the context of two research projects at Roma Tre University, specifically the MIDST project with the Database Research Group of the Department of Computer Science and Automation, and the PRAISED project with the Theoretical Biology and Biochemistry Laboratory of the Department of Biology. The main contributions of this work can be summarized as follows:

- the restructuring of the underlying data dictionary of the existing MIDST framework at the conceptual level, by introducing generalizations among its constructs, and the consequent extension to the Datalog language, used for the translation rules, by enhancing it with polymorphic features to properly take advantage of the defined generalizations. Syntax and semantics have been defined for the extended language, and a proper algorithm for interpreting polymorphic rules has been contextually developed.

- the definition of (four) PolyDatalog rules to replace several tens of classic Datalog rules within the whole translation processes of the MIDST framework, and an extensive testing phase to assess the correctness and performance of the rewritten translations. This has led to a dramatic decrease in the number of rules required within them and a simultaneous surge in ease, scalability, maintainability, and reusability of the rules themselves and the overall approach, thanks to the inner parametricity (and thus generality) of the PolyDatalog rules.

- the inception, design and development of the PRAISED framework and its core abbreviation discovery methodology, aimed at tackling the problem of automatic information extraction from full-text papers in a domain-independent fashion. Such a methodology is made up of three main

phases, related to (i) identifying abbreviations within a source text, (ii) resolving the identified abbreviations by finding their corresponding explanation, and (iii) matching these explanations with entities of a desired domain, given a domain entity repository. The overall approach overcomes typical drawbacks of similar techniques, by employing: lightweight morphological checks and criteria for the identification and resolution phases, with minimal Natural Language Processing (NLP) overhead, resulting in extremely fast execution times; weak constraints upon the structure of the candidate words, in order to broaden recall with respect to other methods using strong constraints; full-text papers as specific targets around which the whole system has been designed and built, and which differ from their abstract counterparts not just in terms of length, for they feature a far higher level of complexity and "unstructuredness" and a wider range of abbreviation forms, especially in chaotic domains like biomedical publications; a final entity recognition phase, which is indeed lacking in the major abbreviation discovery approaches.

- thorough experimentation with the PRAISED framework on a variety of input sources, ranging from abstract corpora to a full-text corpus and web articles, and a contextual comparison with the major similar approaches (as far as was possible, for the systems differed in both nature and functionality).

1.3 Organization of the Thesis

This Thesis is organized into five main parts with a certain number of chapters each. Figure 1.1 displays the overall structure, highlighting the relationships between parts and chapters and the categories that chapters belonging to different parts fall into, as well as providing the reader with potential reading paths as indicated by the linking arrows.

In *Part I: Background*, Chapter 1 (the present chapter) introduces and motivates the research presented in this Thesis, describing its structure and contents as well.

Part II: An Experience in Model Management is devoted to describe the contributions related to the Model Management area. Specifically, Chapter 2 describes the core elements of Model Management and discusses the MIDST framework implementing one of its operators, within the context of model-independent schema and data translations. Chapter 3 introduces PolyDatalog, an extension to the Datalog language meant to incorporate polymorphic fea-

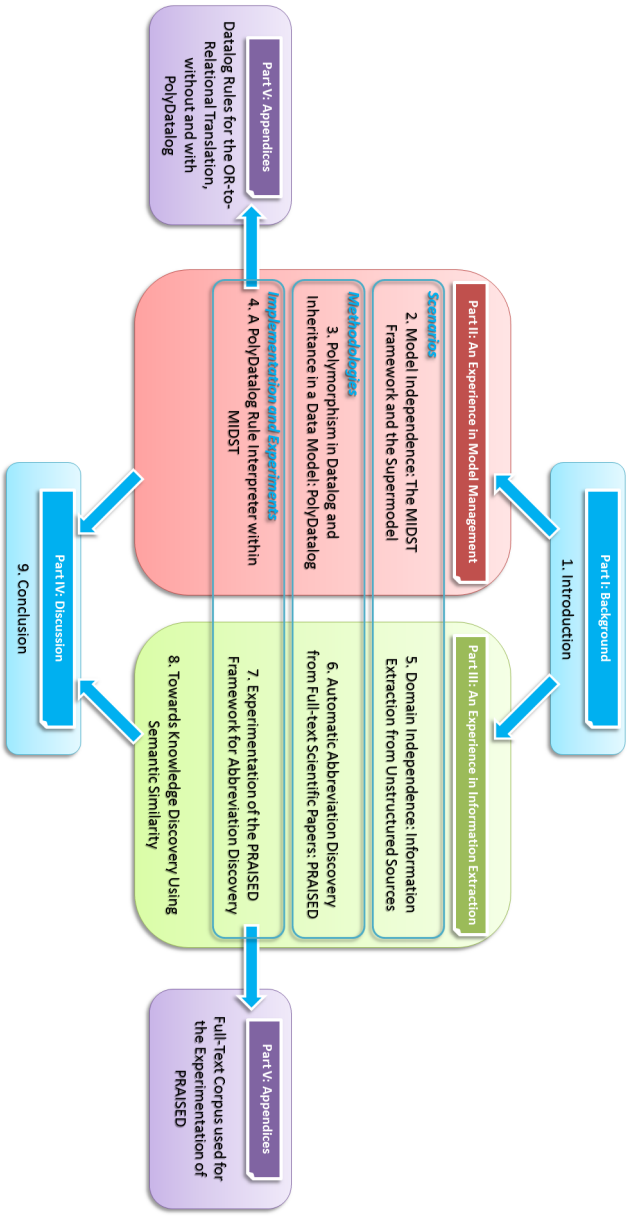

Figure 1.1: Structure of this Thesis, highlighting the main parts it is composed of, the relationships between them and the various chapters, and the categories of the latter. The arrows show potential reading paths to be followed.

tures, in order to take advantage of the generalizations introduced in MIDST's data model. Chapter 4 concludes this part by listing and commenting the experimental results of a PolyDatalog interpreter within the MIDST framework.

Part III: An Experience in Information Extraction focuses instead on the contributions related to the Information Extraction area. Specifically, Chapter 5 raises the problem of extracting information from textual sources, and underlines a complex application domain where such a problem is especially critical. Chapter 6 presents the abbreviation discovery process implemented by the PRAISED system, consisting in a three-phase process: the abbreviation identification phase, the abbreviation resolution phase, and the domain entity recognition phase. Chapter 7 reports the results obtained by the experimentation of PRAISED against a variety of test data, and compares them with the major similar approaches. Chapter 8, finally, proposes an extension to the abbreviation discovery methodology in the PRAISED framework, by relying on semantic similarity computed with the use of automatically built ontologies.

Part IV: Discussion concludes this dissertation, wrapping up its contributions and pointing out potential future directions of research.

Part V: Appendices contains additional material related to Part II and III, like the Datalog and PolyDatalog rules required in a sample translation and the full-text corpus used for the testing phase of the PRAISED framework.

Chapter 2

Model Independence: The MIDST Framework and the Supermodel

"The boy who is going to make a great man, or is going to count in any way in after life, must make up his mind not merely to overcome a thousand obstacles, but to win in spite of a thousand repulses or defeats."

Theodore Roosevelt *The Strenuous Life*

In this chapter, we introduce the core elements of Model Management, and discuss the MIDST framework which implements one of its operators within the context of model-independent schema and data translations. We then proceed to highlight the motivating scenario behind our work described in the next two chapters.

2.1 Model Management

The creation and management of most information systems involve the design, integration and maintenance of complex artifacts, such as application programs, databases, websites, workflow scripts, user interfaces and so forth. In order to effectively perform such tasks, a manipulation of formal descriptions

(models) of these artifacts, i.e. object diagrams, interface definitions, database schemas, XML schemas etc., is inevitably required. This manipulation usually brings up the necessity of carrying out transformations between the data models, the latter requiring in turn an explicit representation of mappings with the purpose of explicitly defining how models relate to one another.

In order to provide a more effective approach for tackling these issues, Bernstein et al. [12, 13] devised a framework called *model management*, with the purpose of supporting development of meta-data intensive applications in different domains.

The foundation of model management lies in fact on the concept of "meta-data", also called "data about data"[1]. Basically, model management is a new approach to meta-data management that offers a higher-level programming interface than other techniques. As a consequence, model management is a generic approach to solve problems of data programmability where precisely-engineered mappings are required.

A first class citizen of this approach is the *model*. A model is a formal description of a metadata artifact. Examples of models include databases and XML schemas, interface specifications, object diagrams, UML-like diagrams, device models, form definitions and ontologies as well. We must underline the fact that, in this work, we use a slightly different terminology with respect to the traditional definitions. In fact, here we use a more database-like terminology, where a *schema* is the description of the structure of the database and a *data model* (also called a model in brief) is a set of constructs that can be used to define schemas. In this regard, examples of models are the Relational model or the Entity-Relatioship model, just to name a few. Instead, Bernstein uses the term "model" for what we call Schema, and "metamodel" for what we call model. Since a higher level is also needed within this framework, we will have a metamodel as well, which he would refer to as *metametamodel*.

The manipulation of schemas usually involves designing transformations between them: formal descriptions of such transformations are called schema mappings or, more simply, mappings. Examples of mappings are SQL views, ontology articulations, mappings between class definitions and relational schemas, mappings between different versions of a model, mappings between a web page and the underlying database, and so on.

[1]Meta-data is a somewhat overloaded concept, for it is often used to reference either structural meta-data, related to the design and conceptual data structures, or descriptive meta-data, which are additional information about actual data content. We refer to the former definition in our dissertation.

The core idea behind model management is to develop a set of algebraic operators that generalize the transformations across various metadata applications. These operators are applied to schemas and mappings as key elements, rather than to their individual elements, and are generic: they can be used for various problems and different metadata artifacts. Below is the list of the most significant model management operators:

- *Match*: it takes two schemas as input and returns a mapping between them. The mapping returned identifies combinations of objects in the input schemas that are either equal or similar to each other, according to externally provided definition of equality and similarity.
- *Compose*: it takes two mappings between schemas as input and returns a mapping combining the two input mappings.
- *Extract*: it takes a schema and a mapping as input and returns the subsets of the schema involved in the mapping.
- *Merge*: it takes two schemas as input and returns a schema corresponding to their "union", along with two mappings between the original schemas and the output schema. In other terms, the Merge operation returns a copy of all the objects of the input schemas, with the exception of those objects from the input schemas that are collapsed into a single object in the output.
- *Diff*: it takes a schema and a mapping as input and returns the subset of the schema that does not participate in the mapping.
- *ModelGen*: it takes a schema, a source model and a target model as input and returns the translation of the source model into the target model.

Model management operators can therefore be used for solving a variety of problems, as in schema evolution, data integration etc., by relying upon effective programs executed by a model management system.

2.2 A Model Management System: MIDST and its Supermodel

The MIDST (*Model-Independent Schema and Data Transformation*) proposal is an implementation of the ModelGen operator, one of the model management

operators [12, 13] discussed in the previous section. More specifically, the goal of ModelGen is to translate schemas from a data model to another: given a (source) data model M_1, a (source) schema S_1 (in data model M_1) and a (target) data model M_2, it produces a schema S_2 in M_2 that corresponds to S_1; for the data level extension, given also a database D_1 over schema S_1, it generates a corresponding database D_2.

Within the context of the MIDST framework, we use the notion of construct to represent and manage different models in a uniform way. Constructs with the "same" meaning in different models are defined in terms of the same generic construct; for example, entity in an ER model and class in an object-oriented model both correspond to the Abstract construct. Here, we assume the availability of a universe of constructs. Each construct has a set of references (which relate its occurrences to other constructs) and boolean properties. Constructs also have names and possibly types (this is the case of lexical elements with a value like attributes of entities in the ER model). Let us comment on the various aspects with respect to the ER model. References are rather intuitive and two examples are enough to explain them: each attribute of an entity (a specific construct) has a reference to the entity (another construct) it belongs to; each relationship (binary here, for the sake of simplicity) has references to the two entities involved. Properties require some explanation: for each attribute of an entity, by using two properties we can specify whether it belongs to the primary key and whether it allows for null values; for each relationship, by using properties we can describe its cardinality and whether it contributes to the identification of an entity or not: two properties tell us whether the participation of the first and second entity is optional or mandatory (that is, whether its minimum cardinality is 0 or 1), two properties tell whether maximum cardinality of the first and second entity is 1 or is unbounded (N, as we usually write), and another property tells us whether the first entity has an external identifier which this relationship contributes to (i.e. whether it is a weak entity).

In this framework, given a set of constructs, the references are always required to build schemas for meaningful models (for example, a relationship without references to entities makes no sense), whereas properties could be restricted in some way (for example, we can think of models where all cardinalities for relationships are allowed and models where many-to-many relationships are not allowed). Therefore, we can consider models to be defined by means of their constructs, each with a condition on its properties.

Having said that, we define the Supermodel as a model that is able to include all the constructs of the universe in the most general form (i.e. with

no restrictions). An appropriate data dictionary stores all of these constructs, along with their correspondences to those of the specific models.

In MIDST, translations are specified in a Datalog variant, where the predicate names are names of constructs and argument names may be OIDs, names, types, names of references and properties. A specific, important feature is the OID-invention, obtained by means of Skolem functors. Without loss of generality, we assume that our rules satisfy the standard safety requirements: all construct fields in their head have to be defined with a constant, a variable, that cannot be left undefined (i.e. it must appear somewhere in the body of the rule) or a Skolem term. We assume the same for arguments of Skolem terms. Besides, our Datalog programs are assumed to be coherent with respect to referential constraints. More precisely, if there is a rule that produces a construct N that refers to a construct N', then there must be another rule that generates a suitable N' which guarantees the satisfaction of the constraint.

2.3 Motivating Scenario: Increasing Complexity of the Data Model

Over time, the necessity of properly representing a large number of heterogeneous models, and therefore being able to define even more complex schemas, led to the consequent introduction of a significant quantity of new constructs within our data dictionary.

However, a key observation was the following: many constructs, despite differences in their syntactical structures, were semantically similar or almost identical. For example, attributes of entities and of relationships in the ER model and columns in the relational model showed some similarity: they all represented a lexical value. In these cases, two or more constructs could have been collapsed into a single construct retaining their common semantics and possessing a structure obtained by the union of the structures of the constructs involved. Clearly, constructs obtained this way would have had some optional references, together with some mandatory ones. This observation led to the definition of a more compact (i.e. with a smaller number of constructs) and cohesive (i.e. where a single construct can represent all the concepts sharing the same semantics) Supermodel.

The increasing structural complexity of the data model that came as a result has brought to our attention a certain number of issues concerning the language used for defining the translation rules (i.e. Datalog), as well as the current structure of the dictionary itself: on one hand, the necessary Data-

log rules have kept increasing, while on the other hand their scalability and reusability have consequently dropped. Therefore, it was necessary to take into account a possible refactoring for the dictionary and its constructs, in terms of establishing appropriate hierarchies for those constructs which allow for mutually exclusive references towards other constructs.

Therefore, as the first step of a viable solution to these issues, we have considered the introduction of a generalization for each of the aforementioned constructs, whose parent is the generic construct without any references, and whose children are the different variants for the parent construct. Each of these variants features a set of mandatory references (corresponding to one of the original mutually exclusive subsets of references). Datalog with OID invention, though, is extremely limited when it comes to handling such generalizations. Its syntactical (as well as semantic) constraints are in fact a heavy burden we have to cope with: although — from a theoretical point of view — generalizations should prove valuable in overcoming the problems we have early mentioned, their representation according to the Datalog paradigm would instead produce the opposite effect. This is mainly due to the nature of the language itself: being a logic programming language, it does not include a proper set of features to effectively represent and handle generalizations or hierarchies. That is why we have come to the definition of a new Datalog extension, which we have labelled PolyDatalog: by taking advantage of the introduced generalizations via an appropriate use of polymorphism and inheritance, we will show how it is possible to greatly increase scalability, maintainability and reusability for the translation rules and thus the whole translation processes. Chapter 3 will provide details for our solution.

Chapter 3

Polymorphism in Datalog and Inheritance in a Data Model: PolyDatalog

> *"The true spirit of delight, the exaltation, the sense of being more than Man, which is the touchstone of the highest excellence, is to be found in mathematics as surely as in poetry."*
>
> Bertrand Russell *Mysticism and Logic: And Other Essays*

In this chapter, we thoroughly discuss an extension to the Datalog language that is meant to introduce typical Object-Oriented Paradigm (OOP) features like polymorphism and inheritance, in order to take advantage of the restructured data dictionary (with the introduction of generalizations) mentioned in the previous chapter.

3.1 Related Work

The idea of extending logics and rule-based systems with concepts like polymorphism, typing, and inheritance goes back to the beginning of 80's [40]. Recent approaches [21, 22, 2, 3, 27, 34, 29] adapt theories and methodologies of object-oriented programming and systems, proposing several techniques to

deal with methods, typing, overriding, and multiple inheritance.

Gulog [21, 22] is a deductive object-oriented logic (alternatively, according to its creators, a deductive object-oriented database programming language) with hierarchies, inheritance, overriding, and late binding; every Gulog program can be translated into an equivalent Datalog program with negation ($Datalog^{neg}$), where negated predicates are used to discern applicability of a rule to a class or subclass. Many works proposed extensions of Datalog and provided algorithms to translate their custom Datalog programs into "classic" Datalog with negation. In $Datalog^{meth}$ [2], a deductive object-oriented database query language, Datalog is extended with classes and methods; its programs can be translated into Datalog with negation as well. Selflog is a modular logic programming with non-monotonic inheritance. In [3], moving from SelfLog and $Datalog^{meth}$, Datalog is extended with inheritance (there are explicit precedence rules among classes) with or without overriding; programs can be rewritten in Datalog with an extra-predicate to mark rules and make them applicable only for a certain class or subclass; they propose also a fine-grained form of inheritance for Datalog systems, where specialization of method definitions in subclasses is allowed and, when a local definition is not applicable, a class hierarchy is traversed bottom-up (from subclass to superclass) until a class with an applicable method is reached. $Datalog^{++}$ [27] is an extension of Datalog with classes, objects, signatures, is-a relationships, methods, and inheritance with overriding; $Datalog^{++}$ programs can be rewritten in Datalog with negation. A language with encapsulation of rule-based methods in classes and non-monotonic behavioral inheritance with overriding, conflict resolution, and blocking (two features missing in other languages, according to the authors) is presented in [34]. In f-logic [29], by limiting to topics of interest, there are polymorphic types, classes, and subclasses; it is possible to distinguish between two kinds of inheritance: structural, where subclasses inherit attributes of super-classes, and behavioral, where subclasses inherit methods of super-classes; three methodologies (pointwise, global-method, user-controlled) to manage the overriding with behavioral inheritance are provided.

Our approach differs from the aforementioned proposals. They introduce concepts of object-oriented programming and, in particular, propose overriding of methods for sub-classes, where needed. We have a different goal: neither do we need overriding nor define anything for sub-classes (sub-predicates, in our case). Instead, using object-oriented programming terminology, we define a method (the rule) for the super-class (the polymorphic construct) and, consequently, generate specific methods (other rules) for the sub-classes (children constructs). From this point of view, our work has something in common

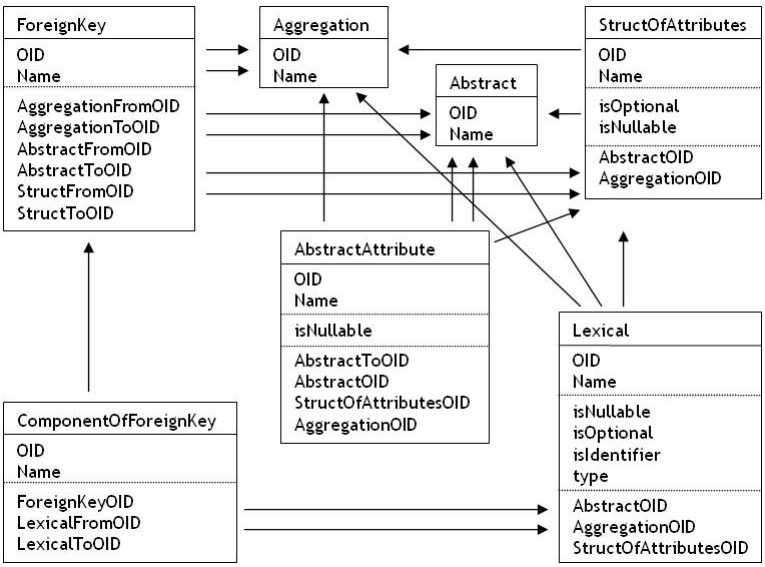

Figure 3.1: A simplified Object-Relational model

with [16] where reusing and modification of rules is allowed by defining "ad hoc" rules for replacing predicate names featured in other rules.

3.2 Birth and definition of PolyDatalog

The main idea behind PolyDatalog is born out of a simple observation. Despite the rising number of Datalog rules necessary for performing more and more complex translations, in fact, a large quantity of such rules feature rather apparent similarities from several points of view. From the syntactical point of view, a huge amount of rules actually share the same syntax, even though it is applied on different constructs, depending on the specific case. And from the semantic point of view, at least as many rules as the aforementioned set, despite being syntactically non-homogenous, share the same purpose, regardless of the constructs involved.

In order to make the following concepts clearer, let us consider a simplified version of the Object-Relational (OR) model (depicted in Figure 3.1) that involves the following constructs of the Supermodel:

- Abstract, representing typed tables;
- Aggregation, representing simple tables;
- StructOfAttributes, representing structured columns;
- AbstractAttribute, representing reference columns from tables (typed or not) or from structured columns, both pointing towards a typed table;
- Lexical, representing columns belonging to tables (typed or not) or to structured columns;
- ForeignKey, representing foreign keys from/to tables (typed or not) or structured columns;
- ComponentOfForeignKey, representing columns involved in a foreign key.

In accordance with the issues concerning mutually exclusive references that we have previously discussed, four constructs within this very model actually allow for this kind of references. Specifically, these constructs are the following:

- Lexical, allowing for three references in mutual exclusion: towards Abstract, Aggregation, or StructOfAttributes;
- StructOfAttributes, allowing for two references in mutual exclusion: towards Abstract or Aggregation;
- AbstractAttribute, allowing for three references in mutual exclusion: towards Abstract, Aggregation or StructOfAttributes;
- ForeignKey, allowing for three references in mutual exclusion: towards Abstract, Aggregation or StructOfAttributes; the presence of a couple of such references (as pointing - from - and pointed - to - construct involved in the foreign key) is mandatory, thus every possible combination of them is permitted.

Within the context of our restructured data dictionary, we were therefore able to "generalize" the constructs mentioned above, each resulting in a hierarchy whose parent is the generic construct without any references and whose children are the specific constructs each with a single reference among their optional ones. This is shown in Figure 3.2.

The rules involving these generalized constructs currently vary just in terms of the different references placed accordingly. Let us stress this via an example,

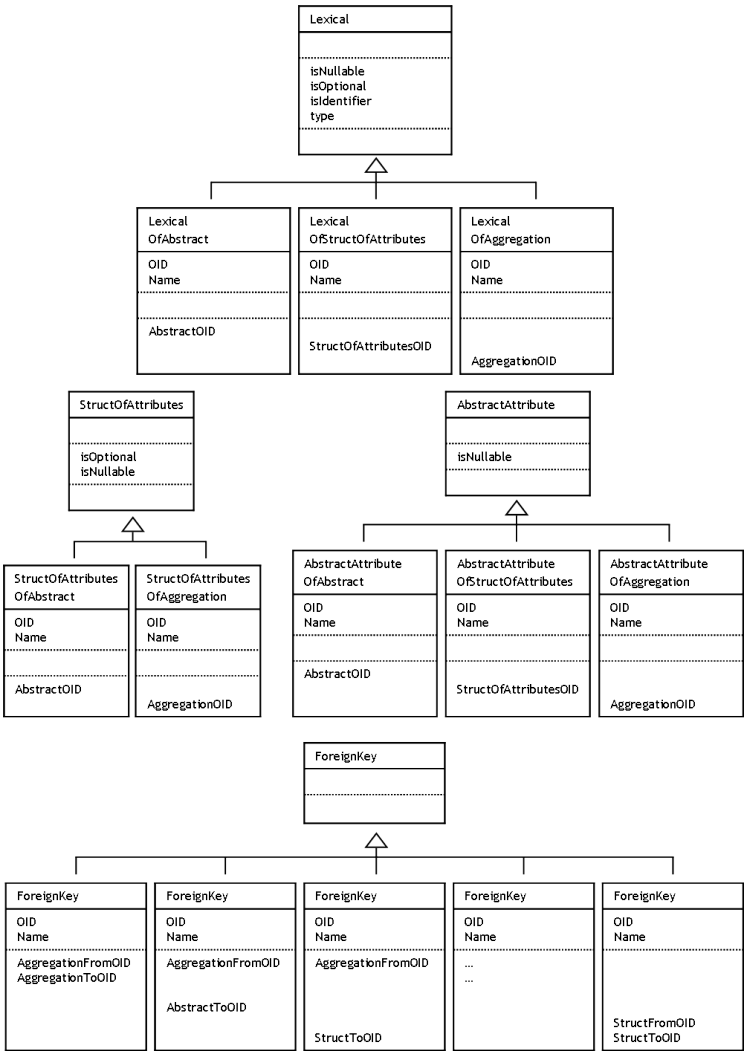

Figure 3.2: Generalizations of Object-Relational model

taken from an actual translation whose source model is the proposed OR model. Should we want, for instance, to translate an OR schema into a Relational one, we would have to perform the following macro-steps:

- Remove typed tables from the schema;
- Remove structured columns from the schema.

It is obvious that these two translation steps will require a certain number of substeps, involving all those constructs related to the typed tables and the structured columns, which will have to be consequently removed as well. In this context, a removal is but a transformation of a certain kind of construct into a different one: for instance, a typed table will have to be turned into a plain table, in order to fit within the destination schema (a Relational one). According to our representation, all the Abstract constructs (representing typed tables) will be turned into Aggregation constructs (representing plain tables). Therefore, all those constructs, whose references point towards a typed table within the source schema, will have to be modified as well, by "changing" their respective references. This is obtained via a Datalog rule for each of these constructs. For example, as we have already said, a Lexical (i.e. a column) allows for three references in mutual exclusion, towards an Aggregation (i.e. a table), an Abstract (i.e. a typed table), and a StructOfAttributes (i.e. a structured column). In the translation process mentioned above, as far as the typed tables removal is concerned, the rules involving Lexical will thus have the following semantics:

1. Copy all those Lexicals pointing towards Aggregation (i.e. leave the columns of plain tables as they are, therefore a simple copy is performed);
2. Turn the Lexicals pointing towards Abstract into Lexicals pointing towards Aggregation (i.e. transform all the columns belonging to a typed table into columns belonging to the plain table resulting from the translation of the original typed table);
3. Copy all those Lexicals pointing towards StructOfAttributes (as in step 1, i.e. leave the columns of StructOfAttributes exactly as they are).

Let us have a quick glance at the rules involved (where the # symbol denotes Skolem functors and we omit non-relevant fields) and double-check the correctness of our point:

LEXICAL (..., aggregationOID: #aggregation_0(aggOID))
←
LEXICAL (..., aggregationOID: aggOID),
AGGREGATION (OID: aggOID);

LEXICAL (..., aggregationOID: #aggregation_1(absOID))
←
LEXICAL (..., abstractOID: absOID),
ABSTRACT (OID: absOID);

LEXICAL (..., structOfAttributesOID: #structOfAttributes_0(structOID))
←
LEXICAL (..., structOfAttributesOID: structOID),
STRUCTOFATTRIBUTES (OID: structOID);

By reasoning on these rules, it may be noticed that, despite some syntactical differences, semantics of the rules involving Lexicals of (i.e. referencing) "something" is always the same whichever that "something" is: "carry over the values of various elements to the target schema, according to the elements they belong to". The idea is that, whenever an analogous transformation of all variants of Lexicals is needed, it is no longer necessary to write a specific rule for each of them, but it is instead possible to write just one polymorphic rule for the root construct of the corresponding generalization; the rule engine will be the one responsible to compile Datalog rules, obtaining this way one specific rule for every single variant of each root construct.

In the general case, a polymorphic rule designed for the transformation of a root construct C, with n children constructs and k different translations for each construct referenced by a child construct, when compiled, will be instantiated kn times, producing a specific Datalog rule for each translation of a construct referenced by a child of the generalization whose parent is C.

A preliminary semantics for a polymorphic rule can thus be the following: given a root construct, i.e. a construct that is parent of a generalization, consider such a generalization; for each of the constructs referenced by its children, establish a correspondence between the referenced construct and its translation(s) within the whole translation process; produce as many rules as needed according to the number of these translations.

3.3 Further details about polymorphism in Datalog

Let's quickly review the basic concepts about polymorphism and then see how they fit in our Datalog extension. According to the object-oriented programming paradigm, polymorphism is the ability of objects belonging to different types to respond to method calls of the same name, each one according to an appropriate type-specific behavior. Specifically:

- an "object", in our approach, is an instantiated Supermodel construct referenced by a construct that is parent of a generalization;
- a "method" is a reference within the Datalog rule specifying the translation for such a construct;
- a "type" is the specific reference (among those in mutual exclusion) possessed by a certain instance of the considered construct (e.g. StructOfAttributes, that is the reference featured by Lexical, being parent of a generalization, when instantiated as Lexical Of StructOfAttributes);
- the "behavior" is the outcome of the specific translation step (e.g.: "translate every Lexical by correctly placing its proper referenced construct, according to the way the latter has been previously translated by the other Datalog rules of the whole translation"), for each of the references featured by the children of a generalization.

3.4 Critical aspects of the PolyDatalog solution

Identifying the translated construct.

First and foremost, the major critical aspect is strictly bound to the PolyDatalog semantics we have earlier sketched. So far, we have been considering the semantics of a classic Datalog rule as the following: "generate the construct expressed by the literal within the rule's head, beginning from the constructs featured as literals within the rule's body". While this certainly holds true for the instances of a polymorphic rule, when instantiating such a rule a critical issue arises: according to the PolyDatalog semantics, we need to know how the constructs referenced by a polymorphic construct have been translated within previous rules in the translation process. Let us go back to our previous example concerning Lexical. Its three referenced constructs in the considered

model, i.e. Abstract, Aggregation and StructOfAttributes, have been respectively turned into Aggregation, Aggregation and StructOfAttributes throughout the whole translation process. But how do we know this? Clearly, by considering those rules where such (three) translations/copies occur. But then again, how do we identify such rules when scanning them? It might be perhaps enough for us to give just a quick glimpse at the various Datalog rules in the simplest cases. However, when dealing with more complex rules, with many constructs in their bodies and a wider range of syntax elements involved, things might get far more complicated. In general, in a Datalog rule, we therefore need to identify the "main", or "translated", construct within the rule's body. This means establishing which construct featured as a literal in its body is the actually translated construct into the construct expressed by the literal in its head. Still referring to our example, as we have stated, Abstract becomes Aggregation. Therefore, there must be a rule, previously examined in the translation process, whose "main" construct is Abstract, and is right there translated into, namely, Aggregation. A rule like the following must thus exist:

AGGREGATION (OID: #AggregationOID_1(absOID), Name: name)
←
ABSTRACT (OID: absOID, Name: name);

Abstract is clearly the translated construct of this rule, becoming Aggregation in the destination schema. Abstract satisfies two paramount requirements: it is featured in the rule's body and its OID argument is found as the argument of the Skolem functor used to generate the head construct's OID.

The aforementioned requirements are necessary but not sufficient in order to correctly identify a main construct within a Datalog rule. In fact, a construct's OID (satisfying every previous requirement) must not be featured as an argument of a Skolem functor used to generate a value for a reference of the head construct. Furthermore, in our Datalog rules, we can refer to constructs belonging to the destination schema as well as to temporary views (used to store partial results): these constructs will never be the translated constructs of the corresponding rule.

By putting together what we have said so far, the algorithm for detecting whether a given construct is the one translated within a given rule will be the following:

Given a construct C and a Datalog rule R:

1. check that C is included within R's body;

2. check that C, found in R's body, is neither an explicit destination construct nor a temporary construct;

3. check that C's OID argument is found within the arguments of the Skolem functor used for generating the head literal's OID; if so, check that it does not also appear within the Skolem functor used for referencing some destination construct in the head literal.

If all these checks succeed, the considered construct is the translated construct of the given rule, and therefore its translation is represented by the construct expressed in the rule's head.

Multiple translations for a single construct.

Strictly bound to the identification process for the main construct of a Datalog rule is the second, most relevant issue we are about to consider. As it results from our previous assertions, when we proceed to check whether a given construct is actually the translated construct of a rule, we are considering it in terms of its name. In other words, we check whether a construct, possessing a certain name, is the main construct within a set of scanned rules. At this high abstraction level, though, a seemingly critical scenario may occur: throughout a whole translation process, a given kind of construct bearing a specific name may have multiple translations, i.e. may result as the main construct of multiple rules. As an example, let us consider the elimination of many-to-many relationships within the ER family: many-to-many relationships have to be translated to entities, while other relationships have to be copied. When compiling a polymorphic rule meant to copy the attributes of relationships, we have to consider both the translations undergone by the relationships. The key point here is that multiple translations depend on the specific features of the constructs involved, expressed by different values of their respective parameters (some constraints on their properties, for instance). Therefore, those parameters, used to discriminate among the different instances of a construct, will have indeed to be included when instantiating the polymorphic rule involving the polymorphic construct which allows such a construct as one of its references.

Multiple polymorphic references within a single literal.

The third critical aspect is related to a particular situation where a polymorphic construct allows for multiple mandatory references among its optional ones (e.g.

ForeignKey). From a superficial view, we could think it is enough for these mandatory references to be handled separately. Actually, this is not the case: having their simultaneous presence as a constraint, we will have to generate, when instantiating the PolyDatalog rule, every possible combination for the allowed references, thus producing a minimum number of n^k rules, where n is the number of constructs featured as children of the considered generalization (whose parent is the polymorphic construct as previously stated), and k is the number of mandatory references to be simultaneously featured within the polymorphic construct. We say minimum under the assumption of a single translation for each of the referenced constructs: since there might be more than one translation, as we have shown in the previous paragraph, such a number is bound to exponentially increase. In fact, by defining m as the sum of the various m_i, i.e. the number of different translations for a construct n_i, the maximum number of rules produced turns out to be m^k. In our representation, we usually end up having $n \leq 3$, $m_i \leq 3$ and $k \leq 2$, therefore producing an amount of rules ranging from 1 and 9^2. It is rather apparent that these particular cases allow for the most effective use of PolyDatalog rules, whereas a single polymorphic rule succeeds in replacing several dozens of classic Datalog rules.

3.5 Syntax and Semantics of PolyDatalog rules

By gathering together all the aforementioned considerations and statements, it is then possible to define a syntax template for a generic PolyDatalog rule, used within a translation process from a schema to another.

The syntax template for a PolyDatalog rule would be as follows:

```
GenericConstruct' (...,
        constructResult[k]OID: #skolemForConstruct[k](cOID[k]))
←
GenericConstruct (...,
        constructOrigin[k]OID: cOID[k]),
ConstructOrigin[k] (OID: cOID[k]);
```

where:

- GenericConstruct is the name of a polymorphic construct that is parent of a generalization; this is not a keyword, but an actual construct which will have to be specified when writing the PolyDatalog rule.

- GenericConstruct' is the name of the construct which GenericConstruct will be turned into, as specified in the rule's head; it might differ from GenericConstruct, but the children of their generalizations must share the same referenced constructs anyway.

- ConstructOrigin[k] is the keyword used to indicate a generic construct referenced by a children of a generalization whose parent is GenericConstruct. This keyword is used to tell the interpreter, each time an instance of this rule is produced, to replace it with one of the referenced constructs GenericConstruct allows for in mutual exclusion. As we have previously stated, there can be more than one such keyword, for a certain polymorphic construct might require more than one mandatory reference simultaneously: that is why the index k is indeed needed.

- constructOrigin[k]OID is the polymorphic reference to ConstructOrigin within the polymorphic literal in the rule's body. As it corresponds to ConstructOrigin, there might be more than one within a literal;

- constructResult, that is only featured within a reference, is the keyword used to indicate the construct which ConstructOrigin has been previously turned into (copied or translated), according to those rules where constructOrigin is identified as their main construct;

- constructResult[k]OID is the reference to ConstructResult within the polymorphic literal in the rule's head. There might be more than one as well;

- skolemForConstruct[k] is the keyword used to indicate the Skolem functor by which the OID for the destination construct has been generated, concerning the translation from ConstructOrigin to ConstructResult. There might be more than one, according to how many different translations such a construct features throughout the whole translation process;

- cOID[k] is the OID argument for the construct referenced by the polymorphic construct. Such an argument will always appear as argument of construct-Origin[k]OID, of the Skolem functor skolemForConstruct[k], and as the OID argument of the construct referenced accordingly for each instance of the polymorphic rule; it might appear more than once within the instantiated rule's body, according to the parameters which need to be specified when generating such an instance; there might be more than

one. It does not need any interpretation, and will be left as it is when the instances of the polymorphic rules are produced;

- the k index is thus so defined: if the polymorphic construct allows for only one reference, k is not needed; instead, if there are more mandatory references simultaneously featured, k will be replaced by an integer value beginning from 1, in every keyword that contains it.

Given this syntax, a more detailed semantics for a PolyDatalog rule is the following:

- given the polymorphic construct GenericConstruct featured within the rule's body and possessing a polymorphic reference constructOriginOID, consider the generalization whose parent is GenericConstruct;
- for each child of such a generalization, check how its referenced constructs have been translated within the previous rules of the whole translation process, and store the constructOrigin-constructResult associations and their related Skolem functor skolemForConstruct;
- generate an instance of the polymorphic rule for each of these associations, by replacing the featured keywords accordingly.

Chapter 4

A PolyDatalog Rule Interpreter within MIDST

"Since we cannot be universal and know all that is to be known of everything, we ought to know a little about everything. For it is far better to know something about everything than to know all about one thing."

Blaise Pascal *Pensées*

In this chapter, we provide the algorithm for interpreting PolyDatalog rules, as a result of the discussion from the previous chapters, and show the experimental results of its implementation within the MIDST framework.

4.1 Algorithm for interpreting PolyDatalog rules

By taking into account all the various considerations and arguments we have discussed in Chapter 3, we come up with the following algorithm to be used by a PolyDatalog interpreter. It takes a translation process as input and refers to a given set of generalizations defined over the (meta)constructs of our data dictionary.

```
polyDatalogInterpreter(P)
 1  for each R of P
 2     if isPolymorphic(R) then {
 3           rootC = getPolymorphicConstruct(R)
 4           refList = getConstructsReferencedByChildren(rootC)
 5           for each ref_i in refList {
 6              ruleSet_i = findRules(ref_i)
 7              IR_i = instantiateRules(ref_i, ruleSet_i)
 8              P = P − R + IR_i
 9           }
10     }
```

This algorithm analyzes every rule of the considered program (line 1). If a polymorphic variable is detected (line 2), it tracks down the corresponding polymorphic construct (line 3) and then obtains the list of its children (line 4), by analyzing the generalizations such a construct is involved in. For each child of a generalization C_i (line 5), it looks out for non polymorphic rules in **P** whose main construct is one of those referred by C_i (line 6). Now it can produce the needed non polymorphic rules (line 7) and replace the polymorphic rule in **P** with these generated rules. (line 8).

The PolyDatalog rule used to replace, for instance, the three rules listed in Section 4.2 will be as follows:

```
Lexical (...,
         constructResultOID: #skolemForConstruct(cOID))
←
Lexical (...,
         constructOriginOID: cOID),
ConstructOrigin (OID: cOID);
```

When processing this rule, the interpreter will look at the generalization having Lexical as its parent, consequently finding out that Abstract, Aggregation and StructOfAttributes are the constructs referenced by its children (see Figure 3.2). By scanning the regular rules whose main construct is one of those three, the interpreter will therefore produce the aforementioned rules, having Abstract been turned into Aggregation, Aggregation into Aggregation,

and StructOfAttributes into StructOfAttributes in the considered translation process from an OR schema to a Relational one.

4.2 Experimental results of PolyDatalog and its interpreter

In this section we will proceed to show the relevant results we managed to achieve thanks to the restructuring of our data model and the consequent introduction of PolyDatalog rules within the schema translation processes. Results were encouraging: we detected far greater benefits in terms of reusability, maintainability and scalability of our solution than what was expected from the theoretical study.

Let us get back to considering the translation earlier proposed whose source was the Object-Relational from OR model to relational, which was meant to remove any typed tables within a given schema.

The outcome of PolyDatalog is rather straightforward: a translation originally made up of 27 rules gets stripped of nearly two-thirds of its rules, while obviously keeping safe both its correctness and completeness. The full list of rules for this translation, without and with PolyDatalog, can be found in the Appendices.

By extending this approach to the whole set of schema translations handled by our system, we have obtained significant advantages: an average 30% of Datalog rules (with peaks of 55%) have been removed by a handful of PolyDatalog rules. The summary of these results is shown in Figure 4.1, while the actual PolyDatalog rules used in all the translations are listed in the Appendices.

We must stress an important point: translations where more than 50% of their rules are removed by a very limited amount of PolyDatalog rules (actually, only 4 of them) are not so rare at all. Actually, a significant number of them gets extraordinary results from the introduction of polymorphic rules: this is usually the case when dealing with source models featuring hierarchies of greater complexity, i.e. with more constructs involved and a larger number of children for each generalization. This happens therefore with models like XML-Schema and OR, the latter having been discussed with practical examples throughout this dissertation. On the other hand, it becomes crystal clear that models with limited and fewer hierarchies get less benefits from the use of polymorphism, as it is shown by our experimental results. Even so, PolyDatalog's inner parametricity in terms of the polymorphic references greatly enhances the schema translations within its range of applicability: first, the

	Number of Initial Rules	Number of Resulting Rules	Number of Replaced Rules	% of Replaced Rules
Entity-Relationship	21	18	3	14.3
Binary Entity-Relationship	83	79	4	4.8
Object (UML Class Diagram)	8	8	0	0
Object-Relational	480	325	155	32.3
Relational	7	7	0	0
XSD	138	88	50	36.2
Supermodel	29	13	16	55.2

Figure 4.1: Experimental results of PolyDatalog and its interpreter

PolyDatalog rules would not change over time even though the construct referenced by the polymorphic one did indeed change; second, only a handful of them have succeeded in removing hundreds of original Datalog rules; and finally, we could anytime define more PolyDatalog rules should the need arise (for instance, when newer and more complex hierarchies are introduced in the data model), all the while getting even larger benefits from our Datalog extension. Besides, as far as performance is concerned, the execution time of the system when performing translations is unaffected by the interpretation of polymorphic rules, since its almost instantaneous processing is preserved even after the introduction of the PolyDatalog interpreter, and no significant user-perceivable slowdown is detected at all.

4.3 Discussion

The restructuring of the data model presented in Chapter 2, carried out by the introduction of hierarchies among its constructs, has been made effective by extending Datalog accordingly, as discussed in Chapter 3. Following the experimentation reported in this chapter, our polymorphic Datalog has ultimately proven dramatically effective within our model management proposal. As a matter of fact, the use of hierarchies has resulted in a major turning point as far as MIDST schema translations are concerned. Thanks to the parametrical structure of the PolyDatalog rules, in fact, more than one-third of the previously necessary rules within our sytem's translation processes have been

removed. Furthermore, reusability and scalability have been greatly enhanced, for PolyDatalog rules are highly reusable and can be in principle successfully applied to future translations as well, whereas new models are conceptually defined and newer, more complex hierarchies are introduced in order to represent them correctly. Such a restructuring, combined with the polymorphic features introduced within Datalog, might prove just as successful in other scenarios as well, where predicates belonging to their data model feature a similar structure and syntactical and semantic similarities can be detected among their rules.

Chapter 5

Domain Independence: Information Extraction from Unstructured Sources

"All courses of action are risky, so prudence is not in avoiding danger (it's impossible), but calculating risk and acting decisively. Make mistakes of ambition and not mistakes of sloth. Develop the strength to do bold things, not the strength to suffer."

Niccolò Machiavelli *The Prince*

In this chapter, we introduce the problem of extracting information from textual sources, and underline a complex application domain where such a problem is especially critical. We then discuss a motivating scenario for the work described in the next chapters.

5.1 Information Extraction

Textual documents, either in their purely unstructured or semi-structured form, are undoubtedly the repository of most of the human knowledge. As the amount of available digital information grows to previously unimaginable levels, an unprecedented number of documents containing essential knowledge, albeit scattered among them, is at people's disposal for a variety of different studies and researches. All of the everyday manual operations involved for

collecting and organizing such a knowledge are consequently getting more and more painstaking and time-consuming, thus yearning for semi-automatic or automatic solutions to help reduce costs in terms of both time and resources employed.

The area of study of Information Extraction has this very goal: providing unstructured or semi-structured texts with a clear, machine-readable structure, thus allowing for automatically extracting relevant information from them. Given the complexity of such a task, only a very specific domain can be usually tackled in an effective way by one information extraction system, which is in fact specifically tailored around it in most of the cases. Reaching a considerable level of domain independence in this area is a challenging objective we believe worth pursuing. In this and the following chapters, we will show how it is possible to devise an information extraction system for abbreviation discovery (a critical problem in the field, which we will further discuss below) by using a very complex domain as a motivating scenario, but with a domain-independent approach capable of tackling any number of different domains altogether.

5.2 The Biological Domain

In the biomedical community, consistent nomenclature of proteins and their corresponding abbreviations (also called short forms, or SF, from now on) is of utmost importance for knowledge dissemination and gene/protein sequence database searching and retrieval. However, there are no generally accepted rules for naming novel proteins and abbreviating the corresponding names, and writing guidelines or suggested best-practices are often ignored and disregarded. There are indeed many ambiguities with different proteins with similar names sharing the same abbreviation. Therefore, in this case an additional layer of complexity has been placed on top of the already unstructured data which any plain text is usually made of. This is a major problem both in the biomedical literature and in sequence databases, generating confusion and errors. Furthermore, genomic initiatives have led to the discovery of a tremendous number of novel proteins and, as a consequence, to an explosion of protein name abbreviations whose correct resolution requires a clean and effective strategy.

5.3 Motivating Scenario: Protein Abbreviations in Biomedical Literature

The aforementioned explosion of protein abbreviations has become a critical issue that can dramatically harm research productivity. In fact, existing databases, repositories, dictionaries and ontologies in the field must be constantly kept up-to-date as new abbreviations are introduced and defined, while fruition of the scientific publications gets harder and harder for the journeyman and the field expert alike. Therefore, such a challenging issue yearns for as orderly and clean a solution as can be provided. Tackling this problem is no easy feat, though, given the data deluge itself and the inner complexity of the biological domain, which features thousands of often ambiguous short names and acronyms for proteins as well as for small molecules, chemicals and so forth. Further complexity is represented by the increasing use of protein names made up of numbers and letters which have little or no reference to the actual structural and functional class of the protein itself (e.g. p53BP2, p53 binding protein 2). Generalist resources such as the web portals *Abbreviations.com* or *Acronymfinder.com*, put to a test with some common protein abbreviations, demonstrate to be unable to address the complexity and the chaotic character of protein abbreviations, and thus more complex approaches are needed to solve this problem. In addition, static resources are inadequate for a field such as protein science, in which the number of known proteins increases almost exponentially over time.

Besides, the impact of abbreviation explosion is hardly limited to proteins, embracing instead the whole biomedical publication world at all levels. The MEDLINE/PubMed repository [43], which is the most comprehensive archive for biomedical papers, has doubled its size throughout the last decade. To cite a few numbers, around 64000 new abbreviations were added to it in 2004 [18], with the total number of abbreviations being estimated at 9 million in 2007 [48]; more than 80% of those abbreviations are ambiguous, each featuring an average of 16 different meanings, with an average of 7 different expanded forms for each meaning [33]. Speaking of proteins only, 533657 entries are currently stored by the latest release (dated 2011-12-14) of the UniProt/SwissProt database, and their corresponding abbreviation collection is almost ten times the size [53, 7]. Figure 5.1 shows the trend representing the growth in the number of protein entries stored in the UniProt/SwissProt database up to the present day.

It is therefore safe to assume that this particular domain is one of the most complex available in this regard. Abbreviation discovery, whose purpose is to

Figure 5.1: Growth in the number of proteins manually reviewed and stored in the UniProt/SwissProt database over the years since its establishment

identify abbreviations (short forms, acronyms etc.) from an input text and match them with their corresponding expanded form in an automatic fashion, is thus an extremely relevant challenge to be faced and addressed. In the next chapter, by successfully addressing the need for abbreviation discovery in the biological world, we will present a domain-independent extraction system which can work in a variety of contexts as effectively as in the biological one.

Chapter 6

Automatic Abbreviation Discovery from Full-Text Scientific Papers: PRAISED

> *"When someone is seeking...it happens quite easily that he only sees the thing that he is seeking; that he is unable to find anything, unable to absorb anything...because he is obsessed with his goal. Seeking means: to have a goal; but finding means: to be free, to be receptive, to have no goal."*
>
> Herman Hesse *Siddhartha*

In this chapter, we present the abbreviation discovery process implemented by the PRAISED system, consisting in a three-phase process. We provide a brief high-level description of the system, then discuss related work in this area, and finally we delve deeper into the detail of our process, by describing each of its core phases.

6.1 Inception of the PRAISED system

All of the considerations discussed in the previous chapter prompted us to try and address the abbreviation resolution problem, which, as we said before, falls into the category of extracting information from unstructured sources, as an aid to the scientific community to properly categorize and classify the

so frequently scattered terms. This is how the PRAISED (Processor for Abbreviation Identification, Disambiguation and Storage) system was born; its first application was indeed oriented towards the biomedical world, where such needs of structure and order are so greatly felt. Even so, the design choices behind its inception allow for absolute generality, for its abbreviation discovery process can be indeed successfully applied to any number of different domains.

The discovery strategy PRAISED implements consists in a three-phase process. First, candidate, domain-independent abbreviations are detected within a full text, via a ranking process based on lexical clues and exclusion rules (*Abbreviation Identification*). Second, abbreviations are matched with their potential explanations (also called LFs - long-forms - from now on) using a series of criteria combined with fitting optimization techniques (*Abbreviation Resolution*), independent of any specific domain as well. And finally, given the availability of a domain-specific repository, an entity recognition (*Domain Entity Recognition*) task is performed on the resulting SF-LF pairs, in order to sort out those actually belonging to the desired domain.

In this regard, it is important to reiterate that even though our process has been originally applied for protein recognition in biological publications, its core design provides for generality and scalability. In fact, the first two steps of the discovery process are not related to the biological (or any other) domain at all, and can be easily compared with other domain-independent approaches. The last step, although "domain-specific", is also free from any ties to a particular domain: underlying domain-specific entity repositories can be plugged in and out without affecting the entity recognition process.

Considering this, we however believe the ability to achieve relevant results on full-text papers — comparable to those obtained by other systems on the simpler abstracts — to be PRAISED's flagship feature, and one of the real novelties of its approach.

6.2 Related Work

Automatically extracting data from unstructured sources has become more and more significant a research subject over the years, due to the increased diffusion and availability of information repositories and archives. The ideal purpose of such an information extraction system is to place a perfectly clear semantic structure upon the retrieved, often messy, data, which usually goes hand-in-hand with the creation of a corresponding relational database for storing the structured information.

In this area, several research groups have proposed a certain number of methodologies for trying to discover acronyms within a source text, ranging from general approaches to more specific techniques. These include the use of regular expressions [45], linguistic cues and pattern-based recognition strategies [51, 55, 31, 42, 46, 57], as well as machine learning algorithms, natural language processing and mixed methods [41, 17, 56].

Earlier approaches [51, 55, 31] limited the identification patterns to all uppercase words or words with at least an uppercase letter, and the resolution patterns to strictly adjacent words whose initial letters only could participate in the abbreviation matching. Others restricted acronyms to parenthesis-enclosed words [45, 46], and placed limits on capital letters and word length [42]. The well-known algorithm proposed in [46] achieved 96% precision and 82% recall on the Medstract Gold Standard Corpus. [42] reached 98% precision and 95% recall on a very small testing set. [17] focused on single words between brackets using dynamic programming to check for an abbreviation explanation to the left of the bracket-enclosed word, scoring 80% precision and 83% recall on the Medstract corpus.

Some subsequent proposals [57, 56] shifted their attention more specifically towards the biomedical world. In detail, [57] used pattern-matching rules based on the correspondence between the initial characters of contiguous words and the abbreviation letters, and obtained an average 95% precision and 70% recall on a small set of biomedical papers. [56], instead, took advantage of the method proposed in [24] to first identify proteins within a text, and then, assuming the identified protein names were all correct, to try and map those names to their corresponding abbreviations. They claimed 98% precision and 96% recall on biomedical abstracts, under however the assumption of correctness of the previous protein name identification step.

More recently, [30] tried to build an abbreviation repository by using a ML approach to extract and resolve SF-LF pairs. The tool they developed (which is still available online, unlike most of the systems we just mentioned) focused nevertheless on paper abstracts, and built its abbreviation archive accordingly. Its resolution rules fall short when a complex full-text paper is provided as its input, showing less accurate results (more on this in Chapter 7. Furthermore, this tool only matches SFs with their corresponding LFs, without trying to resolve the matched LFs as entities of a given domain. A similar tool, based instead on custom patterns, is mentioned in [4], but is not available anymore (at the time of our earlier experiments it was still available, and thus we were able to compare its results on the abstract corpora).

To the best of our knowledge, most experimental results of other proposed

methods seem to be sticking with abstracts as their testing set, where a resolution algorithm usually faces a very limited complexity and, consequently, a far smoother ground. Strong constraints are often placed upon the candidate abbreviations, and/or fixed recognition patterns are employed in the resolution process, resulting in limited recall. The major SF-LF resolution approaches, like [46], [30], [4], also stopped at the LF expansion and did not bother matching those LFs with known entities. Also, performance in terms of execution time is seldom mentioned or not at all, and the overall results are not so easily comparable, for often modified corpora are used to test the resolution algorithms.

Table 6.1: Summary of major approaches for abbreviation discovery

Method	**Approach**	**Tested on**	**Scope**	**NER**[2]
S&H[46]	custom patterns	MEDSTRACT	Abstracts	N/A
ALICE[4]	heuristic patterns	A&T Corpus + MEDSTRACT (original and modified)	Abstracts	N/A
BIOADI[30]	machine-learning	AB3P Corpus + custom corpus	Abstracts	N/A

In comparison, the methodology implemented by PRAISED uses mainly syntactical/lexical clues for finding abbreviation candidates, placing only weak constraints upon the structure of the words, allowing us to achieve high levels of recall in the identification step without resorting to heavier computational approaches like part-of-speech tagging. Besides, our resolution process does not work along fixed recognition patterns, but is instead proximity-based: the search space where abbreviation expansions are looked for is a range of contiguous words, appearing either at the lefthand side or at the righthand side with respect of the abbreviation itself and within the boundary of its sentence, where the probability of finding a long form is usually highest.

6.3 Discovery Process in PRAISED

Let us now illustrate the abbreviation discovery process as implemented in PRAISED. Figure 6.1 shows an overview of the PRAISED discovery process; further details will be discussed in the following paragraphs.

[2]Named Entity Recognition

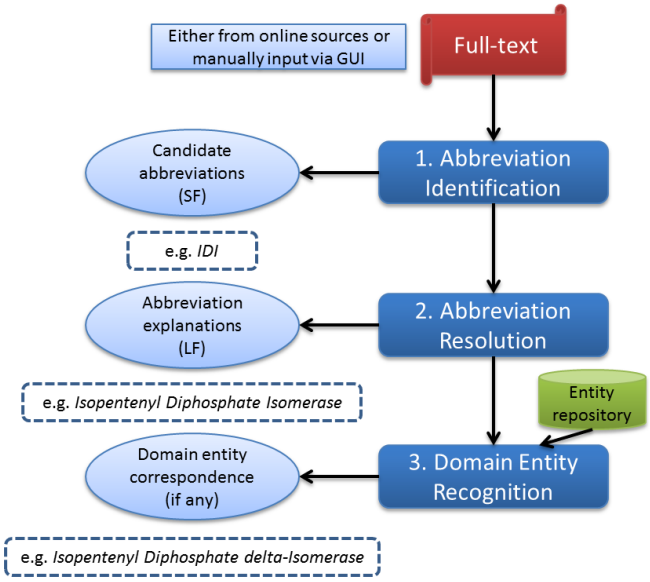

Figure 6.1: Overview of the abbreviation discovery process in PRAISED

System overview

The PRAISED system is a web-based tool designed to take as input a plain text, either pasted by the user (or chosen in the file system) via the system GUI, or automatically retrieved from an online repository in HTML/XML format. At the present time, a crawler is provided for the PubMed Central database [44] and for the news articles of the United States Army website [52]; other crawlers will have to be defined should we want to automatically retrieve texts from different sources. The text passed to the system then undergoes the abbreviation discovery process, up to Phase 2 if an entity repository is not available or if a named entity recognition is not desirable, or the full-fledged execution including the entity recognition as well. Results produced by the system can eventually be reviewed by the user and, if deemed correct, stored in our local abbreviation database. Currently, this is a one-way only data flow: stored results are used only for reports and statistics, but do not intervene in the core discovery algorithm. To phrase it differently, our discovery process has no memory: SFs are identified and matched with LFs and domain entities

(if any) without any prior knowledge about them. Further developments for this approach will be mentioned in Chapter 8.3. In the next paragraphs we will discuss the core discovery process of PRAISED in greater detail, moving along its three phases: the Abbreviation Identification, the Abbreviation Resolution, and the Domain Entity Recognition. The outline of this process and its substeps is shown in Figure 6.2. Let us now illustrate them.

Preliminary actions: Sentence splitting and tokenization

Before the execution of the actual discovery algorithm, the input text is split into sentences, by means of one of Stanford's NLP libraries [49]. It must be underlined that this is a very light "NLP" processing, unbiased by the performance overhead of heavy NLP part-of-speech tagging or parsing, and very efficiently executed by the library. Each sentence is subsequently tokenized and individually passed to Phase 1 of the process as a list of tokens. Current tokenization rules split by blank spaces, leaving however a configurable number of words separated by spaces if enclosed within brackets: the default value for this number is 3.

Phase 1: Abbreviation Identification (AI)

The actual first phase of the process identifies SFs within each of the given sentences from the original text, by using a series of lexical checks for each word considered. Those words scoring higher than a certain threshold will be the resulting candidate SFs retrieved; along with them, a set of contiguous words are stored, to be subsequently used in Phase 2 as the search space for the potential LFs. We therefore use a proximity-based approach, due to the fact that chances of finding the corresponding explanation for a given abbreviation are highest among the words placed immediately before or after it, within the boundary of the sentence they are in. The size s of each set of words (either left or right with respect to the candidate SF) is dependent on the length of the SF itself, resulting in $n + k$, where n is the SF's length and k is a configurable factor. Our experiments have resulted in our current assignment of the value 2 for such a parameter as the best compromise between precision and recall.

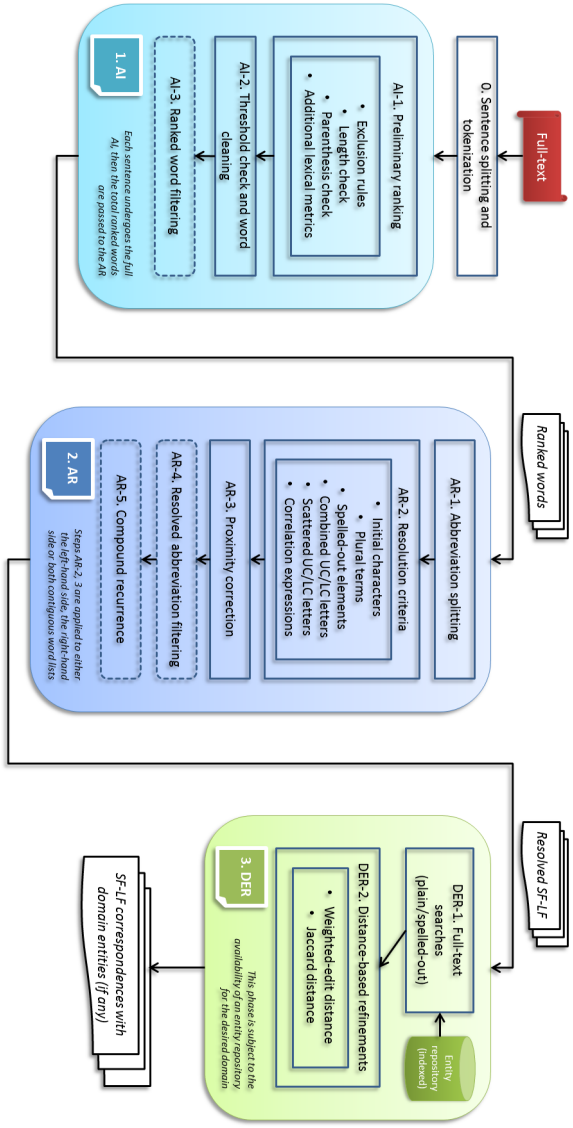

Figure 6.2: A more detailed look at the abbreviation discovery methodology employed by PRAISED. The three main phases building it up are Abbreviation Identification (AI), Abbreviation Resolution (AR), and Domain Entity Recognition (DER). Rectangles represent mandatory steps of the process, whereas rounded dashed lines indicate optional elements.

AI-1: Preliminary ranking

- AI-1.1: Exclusion Rules
  The first task to be undertaken during the identification phase is the exclusion of a certain set of irrelevant words, so that they will not be passed to the actual ranking step. This is done by applying general-purpose rules, which are meant to remove stop-words (*and*, *of*, *or* etc.), a list of known, recurring non-acronym words (*Fig.*, *Table* etc.), and those derived from some known patterns to be excluded (like words containing no letters, or one character-long terms).

- AI-1.2-4: During the last three steps of Phase 1, a number of lexical checks are performed upon each of the remaining words, as we show in Table 6.2 along with the lexical form they are meant to detect and the corresponding default rank increment if such a form is found. These default values have been manually set upon thorough cross-domain experimentation, and so far they have succeeded in providing significant precision and recall values for structured and unstructured domains alike (as with the military and biological domain, which we will discuss later in Chapter 7). Nevertheless, in Chapter 8.3 we will also speculate on a potential and more scalable improvement to this static ranking system.

AI-2: Threshold check and word cleaning

After rank is assigned to all of the considered words, those whose rank is above a threshold (currently set at > 5) will be passed to the subsequent phase of the process, each along the list of the contiguous words found in their source sentence, as we said before. When this happens, candidate SFs are cleared of enclosing brackets (if any) or any other punctuation element that was left.

AI-3: Ranked word filtering

An optional final step can be performed at the end of Phase 1: filtering lexically identical abbreviations, so that only different SFs are present in the resulting set. This is indeed useful when each full-text article is individually processed, for there is indeed a high chance of repeated occurrences for the same abbreviation. On the other hand, when testing our system against artificial corpora (like the collections of paper abstract which we will further discuss in Chapter 7), such a filter could prove disadvantageous, by excluding SF with identical form but with different explanation, as derived from likewise different sources.

Table 6.2: Lexical metrics for step AI-1 (UC = uppercase, LC = lowercase)

Step	**Description**	**Examples**	**Default rank increment**
AI-1.1	Exclusion rules	at, or, and, for etc.	N/A
AI-1.2	Length check	DBMS, A/DACG, etc.	+2 (len<7)/+1 (len<10)
AI-1.3	Parenthesis check	(TCP/IP)	+2 (single)/+6 (both)
AI-1.4	All UC	DAO	+4
AI-1.5	All UC + trailing LC "s"	YACs	+3
AI-1.6	All UC + numbers with at least a letter	1XYZ, A789, 3GL	+4
AI-1.7	All UC + dashes/underscores, at least a letter	A-B-C	+1
AI-1.7.1	More UC than dashes/underscores (if AI-1.7 is true)	RIG-I	+2
AI-1.8	All UC + numbers + dashes/underscores, at least a letter	-H2AX	+2
AI-1.9	More UC than numbers or dashes/underscores	LPAAT1_	+2
AI-1.10	Some LC	QoS	+0
AI-1.10.1	More UC than LC (if AI-1.10 is true)	PPPoE	+3
AI-1.10.2	Initial UC (if AI-1.10 is true)	PPPoA	+1
AI-1.10.2.1	Some numbers (if AI-1.10.2 is true)	IPv6	+3

Phase 2: Abbreviation Resolution (AR)

The second phase of the process is responsible for trying to match a candidate abbreviation with its potential explanation among its contiguous words previously stored. Let us now review this phase, starting from preliminary steps, and then moving onto the series of resolution criteria to be applied for identifying the LF of a given SF. Eventually, we will discuss a handful of optimization techniques, one mandatory and two optional, to be performed after the application of the resolution criteria.

AR-1: Abbreviation splitting

Prior to proceeding to the application of the resolution criteria, an abbreviation — coming from the candidate SF list produced by Phase 1 — is split into a number of subelements, which roughly correspond to each of its characters. We say "roughly", for letters are individually split, while contiguous digits are treated as a single unit. For instance, the elements resulting from *Cyp33* will be *C*, *y*, *p* and *33*.

The purpose of the subsequent resolution phase is to match each of these subelements with a term among the contiguous words previously stored in Phase 1. The resulting match ratio m_r will therefore be computed as $(E_m/E) * 100$, where E_m is the number of matched subelements of the abbreviation and E is the total number of its subelements.

The search space within which these matches are to be looked for by default will first be the abbreviation's previous words, properly tokenized by dashes or other relevant connectors. Empirical tests have shown us that the likelihood of an abbreviation being explained is greater among the words that immediately precede it, while it is lower among the words that immediately follow it. If and only if we get $m_r \leq 50$ at the end of all the passes, the whole resolution process is repeated by taking into account the abbreviation's next words as the new search space. The system can nonetheless be configured via GUI in order for it to scan the rightmost contiguous words first. In both cases, at the end of the two scans (either left-then-right or right-then-left), the threshold for deeming a resolution successful is $m_r > 50$ as well: SF-LF pairs under said threshold will not appear among the results of Phase 2.

AR-2: Resolution criteria

Resolution criteria used for resolving a SF are listed in Table 6.3, and will now be explained.

- AR-2.1: Initial characters
  The first criterion is perhaps the simplest and at the same time the most effective: we try to match the subelements of the SF with terms having those elements as their initial characters. Most SFs like *FWG (Financial Working Group)* are thus correctly resolved in this fashion.

- AR-2.2: Plural terms
  Many abbreviations are cited in the scientific papers as plural nouns, and are consequently explained as such. With this criterion, we try to fully match SF ending with an "s" with the corresponding plural noun in their explanation.

- AR-2.3: Spelled-out elements
  The explanation of an abbreviation can also contain spelled-out elements, like cardinal or ordinal numbers, Roman numbers and Greek letters, which might appear in any number of positions (usually at the beginning or at the end of the explanation). *Third generation language (3GL)* is a significant example of these particular cases. This criterion matches those spelled-out terms with corresponding subelements of the given SF.

- AR-2.4: Combined uppercase/lowercase letters
  An interesting subset of SFs, especially frequent in the biological domain, is structured in a way that a multi-letter prefix is used instead of a single initial character. This is the case of abbreviations like *glutamate receptor (GluR)*, *lidocaine (Lid)*, *murine leukemia virus (MuLV)*, or *vasodilator-stimulated phosphoprotein (VA-SP)*. This criterion checks whether unmatched letters, either lowercase (as in the former three examples) or uppercase (as in the latter example) can be combined with previously resolved subelements (usually uppercase letters) to form the prefix of some word within the search space, generally already matched with another abbreviation subelement; a successful detection of such prefixes will increase the overall match ratio accordingly.

- AR-2.5: Scattered uppercase/lowercase letters
  This "last-resort" criterion checks whether there are some unresolved uppercase or lowercase letters of the considered SF which might be found scattered within the word matched with their previously resolved element; for instance, *allatostatin receptor (AlstR)*, which has some unmatched lowercase letters in the middle (*st*). As in the earlier criteria, match ratio will be increased accordingly if such correspondences are detected. Being

this a broader criterion, by default it is performed on one condition: the given SF must be enclosed in plain brackets, thus having a higher chance of being an actual abbreviation. This is indeed a compromise between indiscriminately applying this criterion (thus bringing about an increased number of wrong matches) and not using it at all (resulting in a loss of recall for several SFs falling under the corresponding categories). It is however possible to configure its usage so that it is applied under any circumstances.

- AR-2.6: Correlation expressions
  There can be cases where a LF, or part of it, does not match any subelements of the corresponding SF, for no lexical bond can be traced back among some or all the abbreviation elements and the explanation terms. This is especially true when the LF refers to another abbreviation or term, or more generally when correlation expressions like *as known as*, *also called* etc. are used to link morphologically uncorrelated words. When the other criteria listed so far have failed to produce a match ratio ≥ 50, this criterion is applied in order to detect these correlations between a SF and its explanation. If a correlation is found, the match ratio undergoes an increase proportional to the proximity of the correlation expression to the abbreviation itself, with a maximum value of 51 (so that, in the worst case, it might still end up above the 50 threshold we have set).

Table 6.3: Resolution criteria for step AR-2

Step	**Description**	**Examples**
AR-2.1	Contiguous words	**FWG**: **F**inancial **W**orking **G**roup
AR-2.2	Plural terms	NLR**s**: (NOD)-like receptor**s**
AR-2.3	Spelled-out elements	**3**GL: **third** generation language
AR-2.4	Combined LC letters	**Cy**p33: **cy**clophilin 33
AR-2.5	Combined UC letters	**VA**SP: **va**sodilator-stimulated phosphoprotein
AR-2.6	Scattered LC letters	Als**t**R: allato**st**atin receptor
AR-2.7	Scattered UC letters	M**B**CS: Mem**b**rane-Curvature Sensing
AR-2.8	Correlation expressions	A-10 **is also known as** Thunderbolt II

AR-3: Proximity correction

Correctness of the matched terms can decrease when dealing with a large search space, usually resulting from very long SFs. Having a high number of words among which to search for matches, the chance of matching the wrong term via the main syntactical criteria discussed so far is bound to increase. In order to adjust the matching precision, a proximity correction is performed after the last resolution criterion. Basically, this step tries to detect resolved elements "distant" to their next element of the SF, in terms of the position of their matched word among the search space. If such a "distant" element is found, it looks for another word whose proximity to the next matched word is higher, and tries to match this word with the considered element, employing the basic matching method used as the first resolution criterion. If a match is established, the element's previously matched word is replaced with the new-found, nearer one. We have detected an average 30% increase of correctness for long abbreviations after applying this proximity correction. Match ratio is unaffected by such a process.

AR-4: Resolved abbreviation filtering

Symmetrically to Phase 1, Phase 2 also features an optional filtering step, where identical SFs might be filtered out, leaving only the one with the highest match ratio. Such an optional filter can be further customized with finer-grained rules, in order to leave a certain number of resolved SF-LF pairs sharing the same SF, and whose match ratio is above a certain level (obviously higher than the standard threshold). This filter helps remove incorrect duplicates when processing a full-text article; on the other hand, it is not advisable to be used when scanning a large corpus, full of disparate SF-LF pairs which might indeed share their SF, but have a different (and correctly resolved) LF altogether.

AR-5: Compound recurrence

Within a paper, it is frequently possible to come across partially unresolvable abbreviations whose unmatched, usually contiguous elements are however part of another abbreviation earlier defined in the text. An optional resolution step was therefore designed to check whether a series of subelements (dubbed a "compound") of a perfectly resolved SF (a "parent" abbreviation) are detected inside other, partially or poorly resolved SFs within a single article. If found, those matching compounds are updated by matching them with the resolved words already associated to the one belonging to the parent abbreviation. In

the end, match ratio is updated accordingly. The potential ambiguity for SFs containing an exact compound of another uncorrelated SF within the same paper is kept to a minimum under our assumption of a perfectly resolved parent abbreviation, along with the lower bound placed on the compound's length (currently ≥ 2). This optional step, just like the one mentioned in the previous paragraph, finds its greater usefulness when processing full-text documents, whereas it might produce poorer results when dealing with large corpora, each with thousands of SFs to be processed at once (and, consequently, with many potential "compounds" which could be erroneously detected).

Results from Phase 2

After the application of the aforementioned resolution criteria, results are produced in terms of a list of successfully matched SF-LF pairs featuring a match ratio above our set threshold (50). We must underline the fact that showing the potential LF as made up of the various matched words lined up together does not always provide us with a 100% correctly retrieved explanation. There might have been words in-between that could not be explicitly matched via the criteria we had employed. For instance, *PACSINs*, resolved as *protein casein substrates in neurons*, would be incomplete in comparison with its correct explanation *protein kinase C and casein kinase 2 substrates in neurons* as found in the original input text. That is why the final product of our resolution phase is the original sequence of words literally reconstructed starting from the delimiting matched words of the abbreviation. This ensures us greater accuracy for SF-LF pairs correctly resolved (even though initially missing some in-between words), while it does not significantly alter the result for those pairs that are incorrectly resolved to begin with.

Phase 3: Domain Entity Recognition (DER)

The third and last phase of the process has the purpose of discriminating those resolved abbreviations that actually correspond to known entities of a given domain. As we said earlier, this is the actual and only domain-specific step of our whole process. Despite that, there is no particular tie to a domain or the other: what makes this phase domain-specific is the entity repository used accordingly. There is no theoretical limit to the kind of repository that can be used: in principle, any database or other data source will do, as long as it can be imported into our system (this will of course require a specific importer module

for the chosen source). The gist of this phase is the following: basically, by using as input the original sequence of words building up the explanation of a resolved abbreviation, we try to tell whether these words correspond to a known entity of the desired domain, via a dictionary-based matching step. First, an indexing is performed upon the entity repository, and a subsequent search is carried out in order to match our input with one of the records within the available data bank. The result is a list of candidate entities, each with a certain score: those scoring higher are more likely to be the actual correspondences with the LFs matched with the identified SFs in the previous two phases (a score of 100 means a perfect match). Refinements are then performed, by considering the string similarity between the input words and the candidate entities, so that the score of those with a greater proximity to the input is increased or maximized. Further details of this phase are described below.

DER-1: Full-text searches

After an index is built upon the underlying entity repository, we proceed to search within it for relevant matches. We use the Apache Lucene library [6] for fulfilling both the indexing and the search task. For each record, we decide to store within our index both the entity name "as-is" and its potentially expanded form in terms of its spelled-out elements, like plain numbers (e.g. 1 -> one, first) Roman numbers (e.g. V -> five, fifth) and Greek letters (e.g. α -> alpha).

The actual search is performed against the index thus created, in the shape of two queries: one will look for a match between the input words and the index fields representing the plain entity names, and the other will do the same between the spelled-out versions of the input words and the spelled-out versions of each entity name. Case-insensitiveness, stop-word elimination, word stemming (i.e. bringing a word back to its lemma form), and other optimizations related to the number and position of the single words within the input text, are all left to Lucene. This search mechanism will assign a score > 0 to those entities somehow matching the input words; we apply here a threshold so that only the most relevant matches are returned as candidate entities. We must underline that the two queries we mentioned are executed as parallel searches, and as such there might be duplicates between their two result sets: these duplicates are merged by adding up their respective score, so that they could climb some positions in the final ranking (after all, if a candidate appears in both result sets, it is likely to be more relevant than others).

DER-2: Distance-based refinements

After the search step is completed, a certain number of the results obtained are already as expected: for those LFs that matched with an entity, the first element of the result list should be the entity itself.

However, for determined input texts, an unsatisfactory situation can occur. This is due to the fact that Lucene tends to assign the same score value to any term that includes the same number of input words. Let us clarify this with an example, taken from the experimentation against biological papers. Given the protein abbreviation *GST*, which after the Abbreviation Resolution phase has been correctly resolved as *Glutathione S-transferase*, we want to find out whether this explanation is a protein or not, given a properly indexed protein repository at our disposal. The queries Lucene performs in this case might provide us with a list of high-scoring protein name candidates, which are in fact all the variations of this protein as featured in the repository (e.g. *Glutathione S-transferase APIC, Glutathione S-transferase alpha-1* etc.). This is all well and good, for at this point it becomes clear that we are actually dealing with a protein. Despite that, all of the candidates returned containing the three input words *Glutathione*, *S* and *transferase*, will have been assigned the same score by Lucene: therefore, all of them, including the perfect match that is indeed *Glutathione S-transferase*, will be returned with their score in a tie. Judging from this, any entity matching exactly the input words considered may not necessarily appear as first in the candidate list, ordered by decreasing score, as provided by Lucene.

In order to make up for this particular behavior, some refinements are found to be necessary: we need a post-processing step to be carried out after the execution of Luce-ne's search. What we actually do is using the notion of string similarity for adjusting the scores of those candidates that indeed represented the most accurate matches with the given LFs. Specifically, we make use of LingPipe's implementation of the weighted edit distance and Jaccard distance [32], thanks to which we check how much the input words and the candidate list returned by Lucene are similar.

We perform these two distance checks in the listed order, each aimed to encompass a different family of cases: respectively, those terms sharing the higher number of consecutive characters (letters, digits, and so forth) and overall length, and those having entire tokens in common, regardless of their individual position. We have manually established distance ranges, so that for a distance value falling inside the given range, the corresponding score of the considered candidate entity is adjusted accordingly. Each of the distance checks

has also been manually assigned a specific weight, in order to fine-tune the final results in terms of both precision and recall. This allows us to compensate Lucene's imprecisions and produce more accurate results.

This matching phase, as we have said so far, is of course based upon the correspondence between an LF and an entity in terms of the latter's name. There might however be several entities in a domain repository sharing the same name, even though differing by other attributes. When this turns out to be case, instead of a one-to-one correspondence, the system will provide a list of matching entities with the given LF.

Chapter 7

Experimentation of the PRAISED Framework for Abbreviation Discovery

"If you wish to be great, begin from the least. If you are thinking to construct some mighty fabric in height, first think of the foundation of humility. And how great soever a mass of building you may wish and design to place above it, the greater the building is to be, the deeper you have to dig his foundation."

Saint Augustine *Sermon 19:2 on the New Testament*

In this chapter, we discuss the results of our abbreviation discovery methodology as implemented in the PRAISED system, derived from an experimentation phase consisting of disparate testing sets. We also compare our solution to the major available approaches in the same area, and finally speculate about potential improvements for our system.

7.1 Experimental Results for PRAISED

Let us discuss how PRAISED fared throughout the experimentation phase we carried out. For this purpose, several testing sets were used to assess the actual capabilities of our system.

Abstract corpora

The first stage of the experimentation, whose purpose was to compare PRAISED with its major competitors, was performed against four annotated and freely available abstract corpora, some of which earlier used by the research groups that designed those very systems: the Medstract Gold Standard [36], the AB3P [1], the BioText [15] and the A&T [5] Corpus. These corpora were made up of a certain amount of paper abstracts extracted from the PubMed online repository, featuring several SF-LF pairs mostly derived from the biomedical world. Further details on the abstract corpora can be found in Table 7.1.

Table 7.1: Details on the abstract corpora used for the experimentation of PRAISED

Name	Articles	SF-LF pairs	Words
MEDSTRACT	400	409[3]	∼78000
AB3P	1250	1221	∼227000
BioText	1000	956	∼244000
A&T	993[4]	1095	∼205000

In order for a fair comparison to take place, we provided PRAISED and the three systems selected for the purpose (Schwartz and Hearst [46], ALICE [4] and BIOADI [30].) with these corpora as input, evaluating our own system up to Phase 2 (thus without the domain entity recognition, which the other systems were not capable of performing). For ALICE and BIOADI, we used their respective online tools (one of which, ALICE's, is apparently not available anymore at the time of this writing); as far as S&H was concerned, we used the implementation found in [47].

Given the following definitions:

$$Precision = \frac{\#\text{ SF-LF pairs resolved}}{\#\text{ SF-LF pairs retrieved}}$$

$$Recall = \frac{\#\text{ SF-LF pairs resolved}}{\#\text{ SF-LF pairs total}}$$

$$f\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

[3] Out of the original 414, 5 were incorrectly annotated, and thus excluded from the count

[4] Out of the original 1000, 7 articles could not be retrieved by their listed PubMedIds

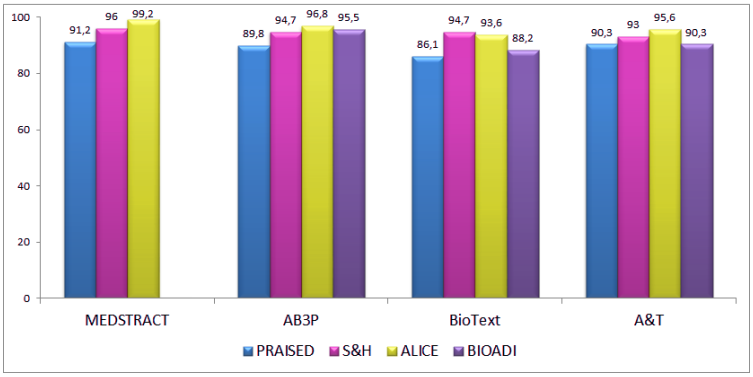

Figure 7.1: Comparison of precision on the abstract corpora between PRAISED and the major approaches

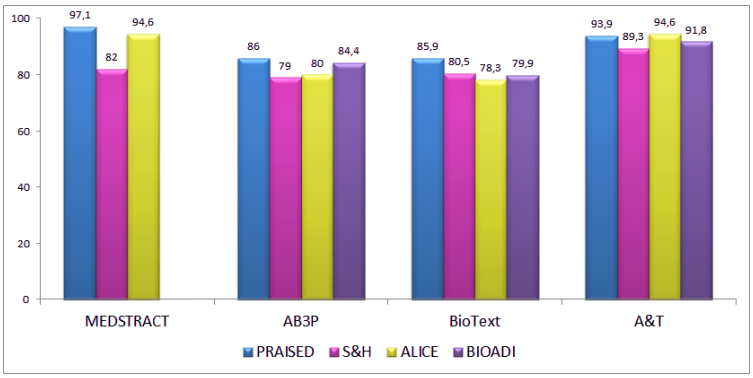

Figure 7.2: Comparison of recall on the abstract corpora between PRAISED and the major approaches

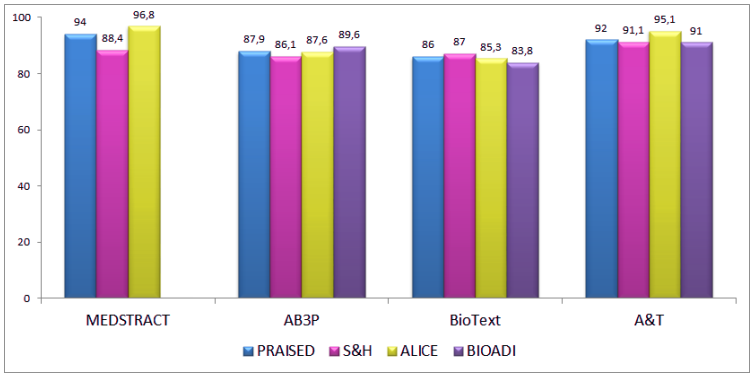

Figure 7.3: Comparison of f-measure on the abstract corpora between PRAISED and the major approaches

the results we obtained are summarized in Table 7.2. Results in terms of recall on the abstract corpora, as compared with those of the other systems, were higher than the competitors in three cases out of four, by 2.4-5.4 points; our system was outperformed in the fourth case by 0.7 points. Precision stayed slightly below the competitors', while the overall f-measure reached significant levels, thanks to the higher recall values. The comparison between PRAISED's results and those of the other systems are conveniently shown in Figure 7.1, 7.2 and 7.3. For this test, the optional steps of Phase 1 (Ranked word filtering) and Phase 2 (Resolved Abbreviation Filtering and Compound Recurrence) were excluded from our system configuration. As far as recall was concerned, out of those abbreviations we could not detect (i.e. not identified in Phase 1), an average 7.5% were single-character abbreviations (specifically excluded by our lexical checks), and an average 10% were bracket-enclosed abbreviations which contained more than two blank spaces (i.e. made up by four words or more). At the same time, there were a number of abbreviations, not originally annotated by the curators of the corpora, which our system successfully identified and resolved, as it is shown in Table 7.3.

All these elements notwithstanding, it is important to reiterate that such a comparison, although interesting, is somewhat artificial in its very nature, since the competing systems lack in functionality (i.e. NER) and capability

[5]The BIOADI web tool generated an error while trying to process the corpus, therefore we could not get any results from it

Table 7.2: Results of PRAISED against the abstract corpora, compared to the major similar approaches

	MEDSTRACT			AB3P			BioText			A&T		
System	**P(%)**	**(R%)**	**F**	**P(%)**	**R(%)**	**F**	**P(%)**	**R(%)**	**F**	**P(%)**	**R(%)**	**F**
PRAISED	91.2	97.1	94	89.8	86	87.9	86.1	85.9	84.3	90.3	93.9	92
S&H	96	82	88.4	94.7	79	86.1	94.7	80.5	87	93	89.3	91.1
BIOADI	N/A	N/A	N/A[5]	95.5	84.4	89.6	88.2	79.9	83.8	90.3	91.8	91
ALICE	99.2	94.6	96.8	96.8	80	87.6	93.6	78.3	85.3	95.6	94.6	95.1

Table 7.3: Additional abbreviations identified and resolved by PRAISED within the abstract corpora

Corpus	SF-LF pairs	Correct
MEDSTRACT	73	69 (94.5%)
AB3P	26	19 (73%)
BioText	22	22 (100%)
A&T	12	10 (83.3%)

(i.e. application to full-texts), and thus can hardly compare with PRAISED in its entirety. In spite of that, results were satisfactory nonetheless in both recall and f-measure, and the lower precision levels mainly derived from our broader, full-text-oriented lexical techniques.

Full-text biological corpus

The subsequent test was carried out against a really challenging set: a corpus of 130 papers, mainly belonging to biochemistry, molecular biology and cell biology, of the past twenty years (1990-2010). Table 7.5 displays some further details, while the list of all the annotated papers and the protein abbreviations featured within them can be found in the Appendices. As we said earlier in the Introduction, the annotation of these papers was manually performed, and only protein abbreviations were hand-picked to be annotated. The configuration used for each paper included all the optional elements of the discovery process, which was tested in full with the availability of the UniProt/SwissProt database [53, 7] for the entity recognition phase. Table 7.4 breaks down the results in terms of precision, recall and f-measure obtained during this test for each major step of the process, and needs further commenting.

As we can see, Phase 1 of the abbreviation discovery process resulted in 94.5% precision and 97.3% recall (f-measure: 95.9), while the process up to Phase 2 scored 90.7% precision and 83.3% recall (f-measure: 86.8). This means that we were not able to correctly identify only 2.7% of the annotated protein abbreviations, with 5.5% of false positives; we could not instead resolve ∼17% of the annotated SFs, while less than 10% of those resolved proved to be incorrect. Incidentally, here, as in the subsequent tests, we deem an abbreviation resolvable if its explanation is to be found within the same paper it is identified in, thus excluding those abbreviations that are only mentioned but not actually explained.

Table 7.4: Results of PRAISED against the full-text corpus

Phase 1			Phase 1 & 2			Phase 3			Full process		
P(%)	**R(%)**	**F**	**P(%)**	**R(%)**	**F**	**P(%)**	**R(%)**	**F**	**P(%)**	**R(%)**	**F**
94.5	97.3	95.9	90.7	83.3	86.8	79.5	91.4	85	89.1	75.6	82.9

Table 7.5: Details on the biological full-text corpus used for the experimentation of PRAISED

Articles	**Protein SFs annotated**	**Words**
130	1325	∼650000

Some of the SFs we could not properly resolve bore no apparent lexical bond with their corresponding explanation, as in *Ftr1, Membrane permease*, or a very loose one, as in *BRCA1, carboxy terminal domain*; others, instead, had their LF at a distance of tens or even hundreds of words, therefore escaping our proximity-based resolution process. As far as domain entity recognition is concerned, its results in terms of precision and recall (considered in isolation, therefore taking as a reference the results from Phase 2, and not all the SF-LF pair which had to be identified and resolved in the first place) were respectively 79.5% and 91.4% (f-measure: 85). The results from the whole discovery process (all the three phases) were finally 89.1% precision, 75.6% recall, with a f-measure of 82.9. Judging from this, several conclusions can be drawn.

First, the lower precision in Phase 3 is explained by the presence of a score of non-protein SF-LF pairs containing several terms appearing in as many protein names featured within the domain entity repository we used. Consequently, several of those explanations generated non-empty candidate lists of entities. Speaking of recall, instead, the 9% lost in Phase 3 was caused by the presence of malformed or misspelled/contracted protein names.

Concerning the whole abbreviation discovery, results point out to Phase 2 as the one whose recall level more decisively affects the entire process, due to the cases mentioned above and the inner complexity of the biological corpus itself. On the other hand, Phase 1 achieves very prominent results, succeeding in identifying almost all the abbreviations from the full-texts, with a stable precision rate. Potential improvements in the discovery process will be discussed in Chapter 8.

As a counterexample of what we stated so far concerning the fundamental characteristics of PRAISED, we recently tried to process those same full-text papers of our corpus with the competing systems earlier shown. At the time of this final experiment, ALICE could not be used anymore, since its web tool was discontinued. We thus began testing the BIOADI web tool, which succeeded in processing less than 30% of the corpus, due to problems in dealing with coding for symbols and Greek letters; against the papers it managed to accept, it yielded ∼51% recall and ∼76% precision as far as protein abbrevia-

tions were concerned (and of course, with no entity recognition), a significantly lower performance with respect to PRAISED. Regarding the S&H algorithm, it displayed an even poorer performance against the entire corpus, with recall at ∼36% and precision at ∼75%.

Military-related corpus

The final test of our experimentation phase took into account a whole different domain, whose requirements are at the opposite end of the spectrum with respect to the biological one. This was done in order to assess the system's capabilities in dealing as proficiently as when applied to the biological texts. That is why we proceeded to identify and resolve abbreviations from military-related news articles, as found in the United States Army official website [52]. This is a quite significant example of an extremely well-structured domain, as opposed to the largely ambiguous and unstandardized biomedical papers. We did not have at our disposal a related military entity repository, so testing was possible only up to Phase 2; for this purpose, we used a rather comprehensive military abbreviation list (which can be found in [37]) as the annotation source for the SFs to be found in the website articles. The set of articles we chose are all those dated September 2011 (see Table 7.6 for details).

Table 7.6: Details on the military-related corpus used for the experimentation of PRAISED

Articles	**Abbreviations (total/% resolvable)**	**Words**
1215	3645/16.7%	∼600000

Since the SFs in those articles had not been previously annotated by anyone, we proceeded along the following course: we first scanned each article, detecting which of the abbreviations featured in the general list actually appeared in it, and used the resulting subset as the basis for computing recall (comprising however both resolvable and irresolvable SFs, since we could not know if their explanation was present or not at that time). We thus applied the abbreviation discovery process of PRAISED, using the expansions in the military abbreviation list as the basis for computing resolution precision. Out of the 3645 abbreviations detected beforehand, PRAISED apparently managed to resolve 15.8%, with 89% precision. Such an unrealistic recall value prompted us to a further analysis: by taking as a sample 1/10 of the considered articles,

we discovered that no more than 16.7% of the abbreviations featured in them were actually resolvable (i.e. their expansion was there). This statistically-significant value let us recompute recall as 91.2%, resulting in a f-measure of 90.1; the truly lost recall (∼9%)) was mainly due to morphologically uncorrelated SF-LF pairs (e.g. *AN/GRQ-27, Goldwing*), much like what happened with the biological corpus. These results are summarized in Table 7.7, and prove PRAISED's capabilities in dealing with similarly structured contexts. At the same time, they made us realize the impact of so many unexplained abbreviations in domains like the one tackled, as well as underlining the need of more advanced correlation criteria for loose SF-LF pairs. All of this led to ideas for future improvements, which we will discuss in Chapter 8.

Table 7.7: Results of PRAISED against the military-related corpus

P(%)	**R(%)**	**F**
89	94.8	91.8

Execution time

The execution time for PRAISED's abbreviation discovery process is one of its major strengths. An average of 150000 words per minute can be processed by the system on a i7 720QM with a medium load from other tasks. This makes elaborating an individual paper an almost instantaneous action: for example, a 20-page-long paper consisting of ∼8000 words is processed in about 3 seconds. A thorough comparison with the other three approaches mentioned throughout this paper is, however, hardly feasible in this regard. First off, the ALICE system, as we already said, has become unavailable as of lately, and unfortunately we had not evaluated its execution time earlier on during our experiments. BIOADI yielded an average performance of 31000 words per minute; however, it must be recalled that the BIOADI execution corresponds to only Phase 1 and 2 of PRAISED. On the other hand, factors such as network latency and server load might play a negative role in the overall BIOADI performance. Regarding the S&H algorithm, its time performance is hardly comparable to any full-fledged system like BIOADI or PRAISED, which involves a complex architecture including an underlying database, a GUI, and so forth.

7.2 Discussion

The effectiveness of the discovery process implemented by PRAISED was shown during our experimentation phase, both when facing collections of abstracts and full-text corpora, each yielding obviously different results, according to the degree of complexity detectable in the source texts.

There is, however, always room for improvements.

First of all, as far as the abbreviation identification is concerned, scalability might indeed be increased, via modifying the current ranking system it adopts. In fact, by moving from statically-defined rank values to dynamically learned ones (by means of an appropriate training set for the desired domain), it will be possible for the system to scale for an even wider range of different domains, which might feature specific forms of abbreviations in greater numbers than what our current rank assignment considers.

Moreover, for improving recall in complex full-texts, we believe it is paramount to try and link together apparently unrelated SFs and LFs, especially where there is no clear lexical bond between them or when they are found in parts of the text very distant from each other. In this sense, the application of a specific annotated ontology could help the detection of complex interrelationships among lexical terms, provided that our system properly takes advantage of it, by introducing a semantic approach next to the mainly syntactical-based methods it currently features.

Furthermore, our methodology might be extended from our current paper-by-paper basis to a corpus-based approach, in order to deal with cross-references among the scientific articles. Many abbreviations, in fact, as we have especially seen with the military-related articles, might be only cited but not explained within a given text, and thus cannot be treated as resolvable SF-LF pairs, whereas no definition is explicitly given for them within the context of their source article or paper. Even more, there are situations (some of them preliminarily tackled via our correlation expressions in Phase 2) where the explanation of an abbreviation includes other abbreviations as well, often not defined within that context.

Finally, we might indeed provide our system with some "memory", by letting it take advantage of previously resolved and stored abbreviations for resolving the next, most of all when the latter could not be resolved otherwise (be it for the lack of an explanation or for the failure of the resolution criteria).

We will try and address some of these potential improvements and extensions in the next chapter.

Chapter 8

Towards Knowledge Discovery using Semantic Similarity

"All our knowledge begins with sense, proceeds thence to understanding, and ends with reason, beyond which nothing higher can be discovered in the human mind for elaborating the matter of intuition and subjecting it to the highest unity of thought."

Immanuel Kant *Critique of Pure Reason*

In this chapter, we propose a methodology to automatically detect characterizing knowledge from semi-structured scientific papers, in order to sort them out according to the subjects they discuss, and therefore allow for speed-reading and cataloging activities. This methodology is born as an extension of PRAISED, in order to overcome its current limits and achieve even greater results in terms of abbreviation discovery capabilities. Nevertheless, it must be stressed out that such a strategy, despite being applied for the purpose of expanding our system, is in principle relevant for a wider range of applications, and might be used to enhance other knowledge extraction systems as well.

8.1 Moving towards a semantic approach

The abbreviation discovery process implemented by PRAISED, as we have seen in the previous chapters, mainly proceeded on a paper-by-paper basis, by deeming an abbreviation resolvable if and only if its explanation could be found

within the same paper the former was featured in. Besides, a proximity-based scan is used to check for abbreviation explanations, thus potentially failing in resolving abbreviations whose explanation is mentioned in a whole different section of the considered paper. By computing semantic similarity among papers from a given domain, we will show how it is possible to shift from a paper-by-paper to a corpus-based approach, as we have briefly mentioned in the Discussion section of the previous chapter, so that abbreviations found in a determined text might be resolved by drawing on a different paper sharing similar subjects with the first.

In fact, those unresolvable (with a paper-by-paper basis) abbreviations may have their expanded form listed within another paper, and actually identifiable by the system via its proximity-based strategy: the more similar the papers' topics, the higher the chance of encountering the same abbreviations. By taking advantage of the mutual semantic similarity among papers, we believe it is possible to resolve several of those "unresolvable" abbreviations as well.

The strategy we discuss is made up of several substeps, involving tasks like terminology extraction, terminology disambiguation according to context, computation of semantic distance and automatic ontology building/learning and matching/alignment. Regarding such research areas, literature is vast and several potential approaches have been devised. A survey on ontology building methodologies is shown in [50]. Relevant proposals for ontology learning can be found in [54] and [20], although either they require human intervention or they show practical limitations even when dealing with simpler and shorter texts.

As far as ontology matching is concerned, [23] presents a thorough classification of ontology alignment methodologies and tools. One such tool is COMA [35], based on a decade-old experience in the field and largely used by the scientific community, which we will further mention in the Implementation paragraph.

8.2 Knowledge Detection Strategy

In this section we will provide the details for our knowledge detection methodology, as composed by the three main steps further discussed below.

Step 1: Finding characterizing elements in a paper

The first step of the strategy takes as input a corpus of full-text papers, all sensibly belonging to a certain known domain, and produces an ontology for

each of the papers making up the corpus: the ontology will represent the characterizing concepts detected from the text. This step, which involves several computationally-heavy tasks, is performed only once and one paper at a time, as a prerequisite for the subsequent steps, which are instead executed on demand.

Terminology extraction, disambiguation and classification

In order to discover the characterizing elements of the considered paper, relevant terms must be extracted from the source text. This is a natural language processing task, where techniques like part-of-speech (POS) tagging, stemming and lemmatization play a central role. A sample text excerpt can be found in Figure 8.1, taken from one of the Wikipedia entries for "Java", and will be used as an example throughout this discussion. We will refer to it as Paper 1.

> Java is a programming language originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.

Figure 8.1: A text excerpt to be processed: Paper 1

The operation pipeline is as follows:

- **POS tagging**. The source text is tagged in order to identify the POS elements within it; since we are dealing with a semi-structured source, structure elements like paper title, section titles etc. are separately tagged, and will be given a higher weight in the subsequent classification phase. A summary of NLP categories used for tagging can be found in Figure 8.2 (further information can be found in [**?**] and [**?**]). The output of the POS tagging for Paper 1 is shown in Figure 8.3.

- **Candidate characterizing term selection and aggregation**. After the tagging is done, nouns, proper nouns and their combination with adjectives (from the NLP terminology, NN, NNP, NNS, NNPS along with adjacent JJ) are selected as preliminary candidate characterizing terms. Adjacent nouns are combined as well to represent a single compound term.

CC	conjunction, coordinating	**PRP$**	pronoun, possessive
CD	numeral, cardinal	**RB**	adverb
DT	determiner	**RBR**	adverb, comparative
EX	existential there	**RBS**	adverb, superlative
FW	foreign word	**RP**	particle
IN	preposition or conjunction, subordinating	**RRB**	right round bracket
JJ	adjective or numeral, ordinal	**SYM**	symbol
JJR	adjective, comparative	**TO**	"to" as preposition or infinitive marker
JJS	adjective, superlative	**UH**	interjection
LRB	left round bracket	**VB**	verb, base form
LS	list item marker	**VBD**	verb, past tense
MD	modal auxiliary	**VBG**	verb, present participle or gerund
NN	noun, common, singular or mass	**VBN**	verb, past participle
NNP	noun, proper, singular	**VBP**	verb, present tense, not 3rd person singular
NNPS	noun, proper, plural	**VBZ**	verb, present tense, 3rd person singular
NNS	noun, common, plural	**WDT**	WH-determiner
PDT	pre-determiner	**WP**	WH-pronoun
POS	genitive marker	**WP$**	WH-pronoun, possessive
PRP	pronoun, personal	**WRB**	Wh-adverb

Figure 8.2: POS tagging categories

Java/NN is/VBZ a/DT programming/NN language/NN originally/RB developed/VBN by/IN James/NNP Gosling/NNP at/IN Sun/NNP Microsystems/NNP (/LRB which/WDT is/VBZ now/RB a/DT subsidiary/NN of/IN Oracle/NNP Corporation/NNP)/RRB and/CC released/VBN in/IN 1995/CD as/IN a/DT core/NN component/NN of/IN Sun/NNP Microsystems/NNPS '/POS Java/NN platform/NN ./. The/DT language/NN derives/VBZ much/JJ of/IN its/PRP$ syntax/NN from/IN C/NNP and/CC C/NNP +/FW +/FW but/CC has/VBZ a/DT simpler/JJR object/NN model/NN and/CC fewer/JJR low-level/JJ facilities/NNS ./. Java/NN applications/NNS are/VBP typically/RB compiled/VBN to/TO bytecode/VB (/LRB class/NN file/NN)/RRB that/WDT can/MD run/VB on/IN any/DT Java/NNP Virtual/NNP Machine/NNP (/LRB JVM/NNP)/RRB regardless/RB of/IN computer/NN architecture/NN ./.

Figure 8.3: Result of POS tagging on the sample text

- **Lemmatization**. Candidate terms from the previous selection are brought back to their lemma form, in order to ease the disambiguation to follow. The resulting candidate term list of Paper 1 after selection, aggregation and lemmatization is shown in Figure 8.4.

java	syntax
programming language	C
James Gosling	C++
Sun Microsystems	object model
subsidiary	low-level facility
Oracle Corporation	java application
core component	class file
Sun Microsystems	Java Virtual Machine
java platform	JVM
language	computer architecture

Figure 8.4: Candidate characterizing terms after selection, aggregation and lemmatization

- **Terminology disambiguation and filtering**. Candidates thus identified are disambiguated according to words adjacent or next to them, so that polysemous terms are correctly associated with their actual meaning with respect of the lexical context they are placed in. This activity requires an external knowledge base, like WordNet or Wikipedia. In our example, some potentially ambiguous terms are disambiguated and correctly associated to their actual meaning, as we can see in Figure 8.5.

java [Java, programming language]	syntax
programming language	C [C, programming language]
James Gosling	C++ [C++, programming language]
Sun Microsystems	object model
subsidiary	low-level facility
Oracle Corporation	java application
core component	class file [Java class file]
Sun Microsystems	Java Virtual Machine
java platform [Java, software platform]	JVM
language	computer architecture

Figure 8.5: Candidate characterizing terms disambiguated according to their context

- **Terminology classification**. Once the context information has been correctly associated with the candidate terms, they are given a score, or weight, based on the term frequency (TF); only those scoring higher than a set threshold will be actually selected. Terms derived from the paper structure, as we said earlier, will be given priority, so that their score could end up above the aforementioned threshold nonetheless. In the case of our sample text, due to its short length the majority of the detected terms will share the same score, for they appear only once in the text (with a few exceptions); the threshold setting should thus take into account the text's length, so that the classification procedure could work well with differently-sized input papers.

- **Abbreviation identification**. Along with the term list produced so far, a list of the abbreviations featured within the considered paper is also compiled and stored, by taking advantage of Phase 1 of PRAISED's discovery process. In Paper 1, only the term *JVM* is recognized as an abbreviation, and thus stored separately from the characterizing terms earlier produced (even though also featured among them).

Ontology building

The previous, NLP-based processing is able to come up with a collection of terms, each weighted according to the frequency with which it appears within the text and associated with its actual meaning. The next challenging task lies in tying those terms together, by identifying relationships among them. This is what is meant with ontology learning or building, and comes down to tracing back, in an automatic fashion, explicit (or somewhat implicit) ties among terms.

In this regard, a pattern-matching strategy might be employed to discover a certain number of interrelationships from the lexical contexts of the considered terms. Obviously, this kind of automatic detection could not claim completeness: only a selected number of relationships might be inferred in this fashion (e.g. is-a, equivalence, part-whole, property/relation etc.). On the other hand, by restricting the range of relationships to a similar subset, the semantic comparison to be performed in Step 2 can be smoothed and produce more effective results.

Let us review this process by considering our example. In Figure 8.6 we see the resulting ontology automatically built from the source text categorized in Step 1. Relationships correlating the characterizing terms of the input text

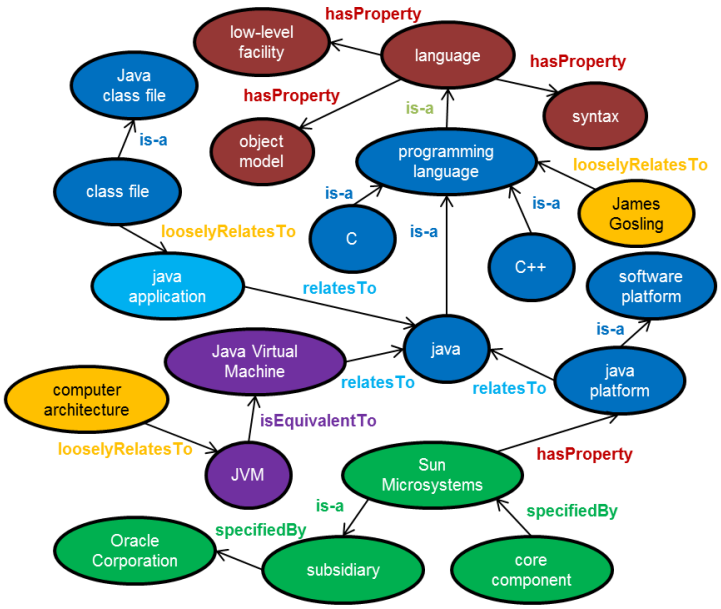

Figure 8.6: Ontology automatically built from the characterizing terms obtained in Step 1 for Paper 1

are built in the following order and with the following strategies (each step corresponding to a different relationship color in the figure; the background color of an element underlines the substep in which that element first appears in the ontology):

- is-a relationships derived from terminology disambiguation (blue color);
- equivalence relationships, via equivalence patterns (following parenthesis-enclosed explanation, correlation expressions like “as known as” etc.) (purple color);
- is-a relationships from lexical patterns (verb “to be” + article, relative connectors + verb “to be”, “such as” etc.), and specification relationships (from “of” connectors) (green color);

- part-of relationships from expressions like "is made of" etc., and property-owning relationships from verb "to have", possessive adjectives and similar expressions (brown color);
- is-a relationships derived from lexical inclusion of characterizing terms (as in "programming language", which is a "language") (olive color);
- general relationships from terms sharing an adjective/a specifying element to the actual term shared (as in "Java Virtual Machine", "java platform", "java application", which are all related to "java") (light blue color);
- loose relationships for tying terms either isolated or yet to be correlated, according to proximity and appearance in the same sentence (the uncorrelated "computer architecture" to "class file", or the isolated "class file" to "java application") (yellow color).

Some imprecisions can be noticed in the automatic building. For instance, "class file", earlier disambiguated as "java class file", ends up being a subclass of "java class file", while it should be the other way around; also, a significant term like James Gosling, the Java creator, gets loosely tied to "programming language" instead of "java", due to the inability of inferring a specific relationship between it and the term "java". There are of course margins of improvement for the automatic ontology building process based on lexical patterns.

In the end, though, the results of Step 1 will be as many ontologies as the papers processed, along with an abbreviation list for each of them. This way, a full-text corpus can be automatically categorized with semantic information for the papers it includes.

Step 2: Computing semantic similarity

Once the paper corpus has been properly semantically categorized, it is possible to proceed with the on-demand steps. The second step takes place whenever a paper, scanned by PRAISED for abbreviations, ends up featuring an abbreviation "without" its corresponding explanation. This may happen for two reasons: either the abbreviation explanation escapes the proximity-based approach implemented by PRAISED (ending up in a different sentence or a different section of the document altogether), or it is simply not present within that very paper. In order to try and resolve such an abbreviation nevertheless, the paper corpus must be taken into account: we need to identify those papers

bearing the highest similarity with the original text in terms of the subjects discussed. For this purpose, the ontologies created in Step 1 must be purposefully compared.

A Java Virtual Machine is a piece of software that is implemented on non-virtual hardware and on standard operating systems. A JVM provides an environment in which Java bytecode can be executed, enabling such features as automated exception handling, which provides "root-cause" debugging information for every software error (exception).

Figure 8.7: A text excerpt featuring an unresolvable abbreviation: Paper 2

Java Virtual Machine	java bytecode
piece	feature
software [Computer software]	automated exception handling
non-virtual hardware	root-cause debugging information
standard operating system	software error [Software bug]
JVM	exception
environment	

Figure 8.8: Characterizing terms for Paper 2

Let us consider the text in Figure 8.7, taken from the Wikipedia entry for "Java Virtual Machine", which we will refer to as Paper 2. Such an excerpt features an abbreviation, *JVM*, which escapes PRAISED's proximity approach for its abbreviation resolution phase: thus, it cannot be resolved by the system. This text, whose characterizing terms are listed in Figure 8.8, turns into the ontology in Figure 8.9 after Step 1. Incidentally, such an ontology features an island of terms isolated from the rest of the structure: this might not be allowed in certain formalizations.

Ontology alignment

The process of comparing ontologies with each other is usually defined as ontology matching or alignment. This is another non-trivial phase and an open research problem as well, comprising an extremely broad range of methods and techniques.

Since we strive to detect papers discussing similar topics as our ultimate goal, it is paramount to try and identify terms which are both alike and related to all or some of the same concepts as well. A pair-wise comparison between

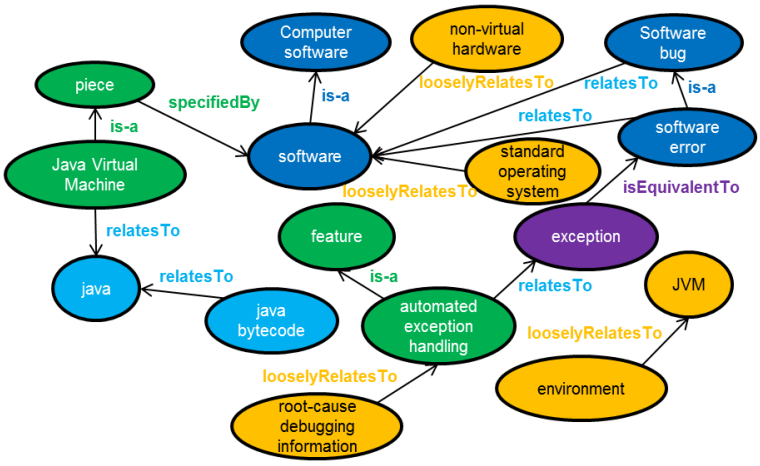

Figure 8.9: Resulting ontology for Paper 2

elements of the ontologies, with the relationships tying them together, could produce a significant result in our case. In order to compute such a semantic similarity, fitting solutions can rely first onto the lexical level, by checking the mutual distance among the terms of the source and target ontologies, in terms of the Jaccard, weighted edit distance and similar metrics; and/or, they could work on the ontological structure, and compute a comparison in terms of the kinds of relationships defined between elements from the considered ontologies. Furthermore, these local strategies could also be combined to perform more advanced, "global" matchings, in order to further refine the alignment process and come up with a more accurate level of similarity between the ontologies and thus the papers themselves.

Other, more advanced strategies might be considered for this task. By exploiting existing knowledge bases, like WordNet, Wikipedia and the like, we might be able to identify terms belonging to the same context, even though apparently distant with respect of the other matching criteria, for they largely differ by their lexical form or the relationships connecting them to other terms.

The result of this matching is meant to return a similarity score, in order to rank papers from the corpus on the basis of their similarity with the original paper. Those papers deemed most similar will be the candidates where unresolvable abbreviations from the original paper could be indeed found. During

this phase, the abbreviation lists from Step 1 will be also taken into consideration: the amount of abbreviations shared between papers will affect the final similarity score, which will be increased accordingly.

In the example proposed, the ontology from Paper 2 is compared to the other papers in the corpus. In Figure 8.10, we can see the comparison between Paper 2 and Paper 1, where similarly colored elements represent several kinds of similarities existing among them (with no intention of being an exhaustive set). Red color means a perfect similarity, both in terms of lexicon and structure; brown color means perfect string similarity; yellow color means partial lexicon and structure similarity; cyan color means structure similarity only; green color means string similarity only. To reiterate, other similarities could actually be detected in addition to those we have highlighted, either according to the above criteria (as with "java classfile" and "java platform" in Paper 1, all baring a high proximity with all the terms in Paper 2 featuring the word "java") or by applying other matching techniques (like the use of knowledge bases).

Based on a properly configured scoring system, it is safe to assume that these two ontologies, sharing several similarities at various levels, will be eventually assigned a high similarity score.

Step 3: Tracing back abbreviation explanations

As soon as we are done with the critical tasks in Step 1 and 2, the final step is simply a matter of applying the PRAISED process to the papers most similar to the one considered, where chances are higher to find the explanation of the originally unresolvable abbreviations from the source text.

From our example, Paper 1, once assigned a high similarity score with respect of Paper 2, will be a fit candidate to be used as the search space for the unresolved abbreviation (*JVM*) found in Paper 2. Indeed, as it can also be seen by checking its ontology only, the explanation of such an abbreviation is actually featured in Paper 1 (and appears in its abbreviation list as well), and will be successfully matched with its corresponding abbreviation by PRAISED's resolution process, this way resolving the original unresolved abbreviation from Paper 2.

8.3 Implementation

The methodology we have described in the previous paragraph can be implemented in a variety of ways.

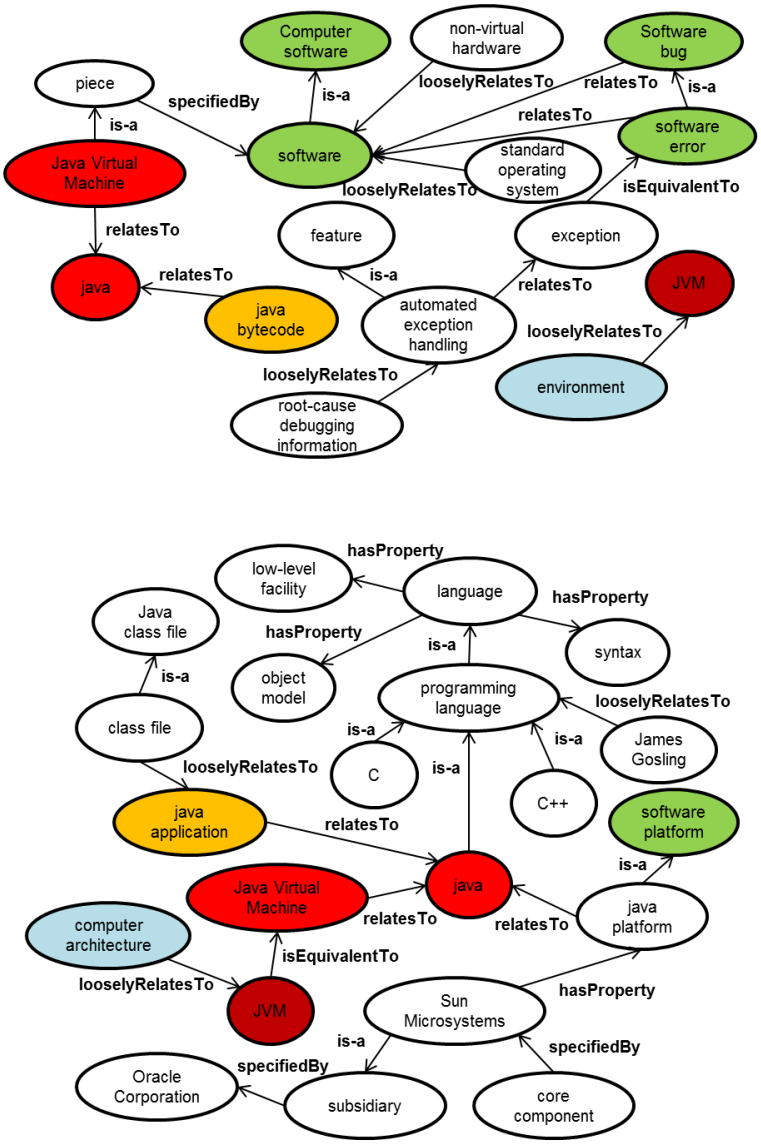

Figure 8.10: Comparison of ontologies between Paper 1 and Paper 2

For Step 1, the Stanford NLP tools [49] could be used to carry out POS tagging and lemmatization, while terminology disambiguation might be performed by taking advantage of an extensive knowledge base like YAGO2 [26], or tools like Wikipedia Miner[38]. Term frequency can be easily calculated via an appropriate algorithm, whereas abbreviation identification can be easily handled by PRAISED - Phase 1. The ontology building process is perhaps the most critical part, and a handful of options can be evaluated for its successful execution. One strategy might lie in a custom implementation, by exploiting the strategy shown in Section 8.2, and/or taking advantage of other patterns, for instance those listed in [19]. Otherwise, a tool like Text2Onto [20] might be able to carry out the given task altogether, even though its accuracy and effectiveness show several limits from our preliminary analysis and tests.

As far as Step 2 is concerned, ontology matching is another open research area, as we mentioned earlier. For the purpose of our system, several alternatives could be considered, ranging from an instance-based, linguistic matching to a schema-based terminological one, depending on the inner characteristics of the automatically-built ontologies from Step 1 (and thus on the method earlier selected for ontology learning). A state-of-the-art ontology matching system is COMA [35], developed by a team with a ten-year-old experience (and soon to be extended to version 3.0), and is a very qualified candidate for the needed task. In the event of opting for a custom or combined solution, libraries like Lingpipe [32] can help compute a number of term distances (Jaccard, weighted edit etc.), while Apache Lucene [6] could take care of indexing and full-text searches should the need arise.

Finally, Step 3 only requires the abbreviation resolution process implemented by PRAISED, applied to the papers that scored higher in similarity to the source text.

Conclusion

"There is nothing outside of yourself that can ever enable you to get better, stronger, richer, quicker, or smarter. Everything is within. Everything exists. Seek nothing outside of yourself."

Musashi Miyamoto *The Book of Five Rings*

In this dissertation we have faced the problem of model and domain independence as applied to two critical areas of the information management lifecycle: the modeling and design phase, and the acquisition and storage phase, respectively represented by relevant experiences in Model Management and Information Extraction.

Within the context of model-independent schema and data translations, we have shown how to extend a model management operator as implemented in the MIDST framework, by restructuring its underlying data dictionary via the introduction of hierarchies among its constructs at a conceptual level. We have consequently enhanced the Datalog language used for performing the various steps of the schema translations, by providing it with polymorphic features in order to take full advantage of the restructured data dictionary. Thus, PolyDatalog was born, and its syntax and semantics have been properly defined, together with an algorithm to interpret its rules, which has been later implemented in the MIDST framework. As a result, translations have been rewritten with the insertion of PolyDatalog rules, the latter replacing hundreds of classic Datalog rules, and dramatically increasing the scalability, reusability and maintainability of the overall approach, thanks to their inner parametricity and generality.

As far as Information Extraction is concerned, we have explored the issue of automatically extracting information from scientific texts in a domain-

independent fashion, focusing our efforts on the matter of abbreviation discovery, whose need is especially felt in disordered and chaotic domains like the biomedical literature. In this regard, we have conceived a methodology specifically designed to extract abbreviations from full-text papers, possessing paramount traits such as: (i) a fundamental attention to the complexity of full-texts, as opposed to the far simpler abstracts more generally used as testing ground in this field; (ii) a light-weight, no heavy NLP-based approach yielding very fast execution times, for an improved user experience; (iii) an entity recognition phase, in order for the abbreviation expansions to be matched with entities of a given domain, given the availability of a suitable domain entity repository; and (iv), last but not least, a core domain-independent nature, so that it might be successfully applied to any number of disparate domains, regardless of their different inner characteristics and complexity. This methodology has been subsequently implemented, giving birth to the PRAISED framework, which has been then tested against several texts, ranging from known abstract corpora (mainly used in order for a comparison with the major existing approaches to actually take place), to a full-text corpus of biomedical papers, up to a corpus of military-related web articles. Significant results have been obtained from this experimentation phase, and all of the flagship characteristics of the methodology have found their fitting place and displayed their real-world effectiveness on the task.

Future Work

We believe researching model-independent and domain-independent solutions to be the correct and essential path, in order to keep advancing towards more scalable and interoperable information systems for a variety of contexts and domains, and therefore bringing about a more cohesive and productive world.

Along these guidelines, we have begun exploring the area of knowledge discovery from unstructured information, as we have introduced in Chapter 8, as a crucial piece that might allow for significant leaps in the overall advancement of information management. To succeed in automatizing as much as possible tasks like ontology building and ontology alingment are fundamental steps towards such a long-term goal. In other words, a progressively improved exploitation of semantic similarity and related techniques, aimed at clustering scattered knowledge and thus providing relevant enhancements in its fruition, are the directions we deem worth to follow from now on.

As a matter of fact, we picture a reality ten years from now with significant

advancements in human-computer interaction, thanks to systematic efforts in building even more intelligent, semantic-based systems for a fruitful discovery and flow of human knowledge as a whole.

Appendices

Datalog Rules for the OR-to-Relational Translation, without and with PolyDatalog

This appendix lists the actual rules used for the example mentioned in Chapter 4 (the translation from an Object-Relational schema to a Relational one), first as they appear before using PolyDatalog, and then as they result from the introduction of PolyDatalog rules. These PolyDatalog rules are explicitly shown at the end of the appendix.

OR-to-Relational translation rules without PolyDatalog

- copy Aggregations
- copy Lexicals of Aggregations;
- turn Abstracts into Aggregations;
- turn Lexicals of Abstracts into Lexicals of Aggregations;
- copy StructOfAttributes of Aggregations;
- turn StructOfAttributes of Abstracts into StructOfAttributes of Aggregations;
- copy Lexicals of StructOfAttributes;
- create key Lexicals of Aggregations for those generated from Abstracts;
- create Lexicals of Aggregations for those that have AbstractAttributes to define ForeignKeys;

- create Lexicals of Aggregations for those generated from Abstract that have AbstractAttributes to define Foreign Key;
- create Lexicals of StructOfAttributes for those that have AbstractAttributes to define ForeignKeys;
- create ForeignKeys for each AbstractAttribute of Abstract;
- create ForeignKeys for each AbstractAttribute of Aggregation;
- create ForeignKeys for each AbstractAttribute of StructOfAttributes;
- create ComponentsOfForeignKeys for each AbstractAttribute of Abstract;
- create ComponentsOfForeignKeys for each AbstractAttribute of Aggregation;
- create ComponentsOf ForeignKeys for each AbstractAttribute of StructOfAttributes;
- copy ForeignKeys from Aggregation to Aggregation;
- copy ForeignKeys from StructOfAttributes to StructOfAttributes;
- copy ForeignKeys from Aggregation to StructOfAttributes;
- copy ForeignKeys from StructOfAttributes to Aggregation;
- turn ForeignKeys from Abstract to Abstract into ForeignKeys from Aggregation to Aggregation;
- turn ForeignKeys from Abstract to Aggregation into ForeignKeys from Aggregation to Aggregation;
- turn ForeignKeys from Aggregation to Abstract into ForeignKeys from Aggregation to Aggregation;
- turn ForeignKeys from Abstract to StructOfAttributes into ForeignKeys from Aggregation to StructofAttributes;
- turn ForeignKeys from StructOfAttributes into Abstract into ForeignKeys from StructOfAttributes to Aggregation;
- copy ComponentsOfForeignKeys.

OR-to-Relational translation rules with PolyDatalog

- copy Aggregations;
- turn Abstracts into Aggregations;
- transform StructOfAttributes;
- copy Lexicals;
- create key Lexicals of Aggregations for those generated from Abstracts;
- create Lexicals of Aggregations to define ForeignKeys;
- create ForeignKeys for each AbstractAttribute;
- create ComponentsOfForeignKeys for each AbstractAttribute;
- transform ForeignKeys;
- copy ComponentsOfForeignKeys.

PolyDatalog rules

Copy/Transform Lexicals

```
Lexical (...,
        constructResultOID: #skolemForConstruct(cOID))
←
Lexical (...,
        constructOriginOID: cOID),
ConstructOrigin (OID: cOID);
```

Copy/Transform StructOfAttributes

```
StructOfAttributes (...,
        constructResultOID: #skolemForConstruct(cOID))
←
StructOfAttributes (...,
        constructOriginOID: cOID),
ConstructOrigin (OID: cOID);
```

Copy/Transform AbstractAttributes

```
AbstractAttribute (...,
        constructResultOID: #skolemForConstruct(cOID),
        abstractToOID: #AbstractOID_0(absToOID))
←
AbstractAttribute (...,
        constructOriginOID: cOID),
Abstract (...,
        OID: absToOID),
ConstructOrigin (OID: cOID);
```

Copy/Transform ForeignKeys

```
ForeignKey (...,
        constructResultFromOID: #skolemForConstruct1(cOID1),
        constructResultToOID: #skolemForConstruct2(cOID2))
←
ForeignKey (...,
        constructOriginFromOID: cOID1,
        constructOriginToOID: cOID2),
ConstructOrigin1 (OID: cOID1),
ConstructOrigin2 (OID: cOID2);
```

Full-Text Corpus used for the Experimentation of PRAISED

This appendix lists the bibliographic references of the papers building up the full-text corpus used in the experimentation of the PRAISED system, as well as the protein abbreviations featured within them. It must be noted that multiple occurrences of the same abbreviation are due to the presence of the abbreviation in more than one paper of the full-text corpus, and that a single abbreviation can be undefined in one paper and defined in others, and/or defined in different ways in different papers.

Bibliographic references

- Antioxidants & Redox Signaling 7 (2005) 964-972
- Archives of Biochemistry and Biophysics 362 (1999) 67-78
- Archives of Biochemistry and Biophysics 428 (2004) 22-31
- Archives of Biochemistry and Biophysics 444 (2005) 15-26
- Archives of Biochemistry and Biophysics 498 (2010) 83-88
- Biochemical and Biophysical Research Communications 282 (2001) 904-909
- Biochemical and Biophysical Research Communications 303 (2003) 771-776
- Biochemistry 1997, 36, 341-346
- Biochemistry 1998, 37, 5394-5406

- Biochemistry 2002, 41, 5963-5967
- Biochemistry 2003, 42, 3464-3473
- Biochemistry 2004, 43, 2829-2839
- Biochemistry 2004, 43, 3289-3300
- Biochemistry 2004, 43, 3979-3986
- Biochemistry 2005, 44, 10914-10925
- Biochemistry 2005, 44, 14725-14731
- Biochemistry 2007, 46, 6097-6108
- Biochimica et Biophysica Acta 1685 (2004) 8-13
- Biochimica et Biophysica Acta 1757 (2006) 90-105
- Biochimica et Biophysica Acta 1767 (2007) 79-87
- Biochimica et Biophysica Acta 1791 (2009) 679-683
- Biogerontology 3: 161-173, 2002
- Biol. Chem. 383, 1667 - 1676, 2002
- Bioorganic & Medicinal Chemistry 11 (2003) 21-29
- Biophysical Chemistry 101 -102 (2002) 145-153
- Biophysical Journal 86 (2004) 3855-3862
- Blood (2006) 108, 2946-2949
- Blood (2006) 108, 353-361
- BMC Biology 2007, 5:17
- Cell 111, 733-745, 2002
- Cell 123, 1213-1226, 2005
- Cell 97, 471-480, 1999
- Cell Metabolism 7, 508-519, 2008

- Cell, 121, 1059-1069, 2005
- Cell. Mol. Life Sci. 57 (2000) 1970-1977
- Cell. Mol. Life Sci. 59 (2002) 1413-1427
- Cellular Microbiology (2006) 8, 1059-1069
- Current Biology 11 (2001) R399-R401
- Current Enzyme Inhibition, 2005, 1, 85-95
- Current Opinion in Cell Biology 2002, 14:88-103
- Current Opinion in Structural Biology 2004, 14:447-453.
- Current Opinion in Structural Biology 2004, 14:765-774
- EMBO 19 (2000) 5661-5671
- EMBO reports 9 (2008) 157-163
- Environ. Sci. Technol. 2005, 39, 5378-5384
- Eur. J. Biochem. 264, (1999) 271-275,
- Experimental Cell Research 315 (2009) 119-126
- Experimental Gerontology 39 (2004) 1475-1484
- Expert Rev. Proteomics 1, (2004) 89-100
- FASEB J. 14, 231-241 (2000)
- FASEB J. 15, 1303-1305 (2001)
- FEBS Journal 272 (2005) 1727-1738
- FEBS Letters 499 (2001) 256-261
- FEBS Letters 513 (2002) 45-52
- FEBS Letters 564 (2004) 225-228
- Inorg. Chem. 1998, 37, 4030-4039
- Inorganica Chim Acta. 2005, 358, 2933-2942

- International Journal of Biochemistry & Cell Biology 33 (2001) 940-959
- J. Cell. Mol. Med. 8, 2004, 201-212
- J. Med. Chem. 2006, 49, 3800-3808
- J. Med. Chem. 2006, 49, 7754-7765
- J. Mol. Biol. (2002) 317, 41-72
- J. Mol. Biol. (2002) 324, 105-121
- J. Mol. Biol. (2003) 328, 505-515
- J. Mol. Biol. (2004) 338, 103-114
- J. Mol. Biol. (2005) 347, 565-581
- J. Mol. Biol. (2005) 350, 987-996
- J. Mol. Biol. (2007) 371, 1038-1046
- J. Neurochem. 67, 2155-2163 (1996)
- J. Peptide Res., 2003, 61, 202-212.
- J. Peptide Res., 67, 2155-2163 (1996).
- J. Phys. Chem. B 2004, 108, 12990-12998
- J. Phys. Chem. B 2005, 109, 19929-19935
- Journal of Biological Chemistry 271, 18379-18386, 1996
- Journal of Biological Chemistry 275, 19906-19912, 2000
- Journal of Biological Chemistry 275, 27940-27946, 2000
- Journal of Biological Chemistry 277, 17209-17216, 2002
- Journal of Biological Chemistry 277, 39937-39943, 2002
- Journal of Biological Chemistry 279, 31842-31853, 2004
- Journal of Biological Chemistry 279, 31873-31882, 2004
- Journal Of Biological Chemistry 281, 14241-14249, 2006

- Journal Of Biological Chemistry 281, 36477-36481, 2006
- Journal of Biological Chemistry 282, 1072-1079, 2007
- Journal of Biological Chemistry 282, 13592-13600, 2007
- Journal of Cell Science 117, (2004) 2631-2639
- Journal of Experimental Biology 203 (2000) 841-856
- Journal of Lipid Research 50, 2009, 1653-1662
- Journal of Molecular Graphics and Modelling 19, (2001) 146-149
- Journal of Neurochemistry, 2003, 85, 610-621
- Mitochondrion 10 (2010) 83-93
- Mol. Biol. Evol. 18(2):120-131. 2001
- Mol. BioSyst., 2005, 1, 79-84
- Molecular Biology of the Cell 17, 163-177, 2006
- Nature 402 (1999) 656-660
- Nature 409 (2001) 198-201
- Nature 438 (2005) 1040-1044
- Nature 450 (2007) 1201-1206
- Nature 454 (2008) 1123-1127
- Nature 454 (2008) 1127-1132
- Nature Structural and Molecular Biology 12 (2005) 582-588
- Nature, 389 (1997) 753-758
- Neurobiology of Aging 21 (2000) 455-462
- Neurology 2004;63:1912-1917
- Nucleic Acids Research, 2004, 32, D129-D133
- Photosynthesis Research (2005) 84: 153-159

- Photosynthesis Research 77: 35-43, 2003
- Physiol Rev 84:41-68, 2004.
- PNAS, 1999, 96, 2042-2047
- PNAS, 2001, 98, 7760-7764
- PNAS, 2002, 99, 1264-1269
- PNAS, 2002, 99, 3505-3510
- PNAS, 2003, 100, 9750-9755
- PNAS, 2005, 102, 15459-15464
- PNAS, 2005, 102, 8955-8960
- PNAS, 2006, 103, 12999-13003
- PNAS, 2006, 103, 1810-1815
- Proteomics 2003, 3, 1154-1161
- Science 298 (2002) 1793-1796
- Science 303 (2004) 1831-1838
- Science, 286 (1999) 304-306
- Toxicon 42 (2003) 391-398

Abbreviation list

4-MD-2 undefined
ABC7/Atm1p Membrane protein believed to be responsible for iron export from mitochondria
ABCA1 ATP-binding cassette transporter
ABCG5 undefined
ABCG5 undefined
ABCG8 undefined
ABCG8 undefined
Abl Abelson tyrosine kinase
Abl undefined
ACAT2 undefined
ActA undefined
ActA undefined
ActA undefined
ActA undefined
ActA undefined
AE33 undefined
AFP alpha-fetoprotein
AGAO A. Globiformis CuAO
AHNAK undefined
Akt undefined
Akt undefined
Alb Plasma albumin
AlstR Allatostatin receptor
AlstR1 undefined
AP2 undefined
AP2 undefined
AP2 undefined
AP2 undefined
Apaf-1 Apoptotic protease activating factor
APC antigen presenting cell
apoA1 undefined
Apo-AGAO Apo A. Globiformis CuAO
apoE undefined
apoL-I apolipoprotein L-I
APP amyloid precursor protein
APPs Acute phase proteins
Arp2/3 undefined
ArsR undefined
Ash undefined
Aso1 Polyamine oxidase from C. bidinii
ATP7A Menkes disease protein
ATP7B Wilson disease gene product
ATPase undefined
Axcyt c' Alcaligenes xylosodans cytochrome c'
BACH1 undefined
Bem1 undefined
BmrR undefined
bRC Reaction center of photosynthetic purple bacteria
BRCA1 undefined
BRCT BRCA1 carboxy terminal domain
BRCT2 undefined
Brn-2 undefined
BSA undefined
BSA undefined
BSAO Bovine serum amine oxidase
BsHemAt Haem based aerotaxis Transducer sensor domain of B. subtilis GCS
BtuB Vitamin B12 transporter protein
C/EBP alpha undefined
C/EBPβ member of the C/EBP transcription factor family
C/EBPϵ CCAT enhancer binding protein-epsilon
C/EPBϵ-ER undefined
C1R undefined
C1S undefined
CAII Human carbonic anhydrase

CAM-SLR Carasius somatostatin-like receptor
CB1 Central cannabinoid receptor heterotrimeric GTP-binding protein
Cbp1p Corticosteroid binding protein
c-cbl Cellular homologue of Casitas B lineage lymphoma proto- oncogene product
CCP Complement control protein
CCS undefined
CD116/CD18 Surface antigens
CD14 undefined
CD163 Macrophage cell surface receptor
CD163 undefined
CD163 undefined
CD163 Scavenger receptor CD163
CD163 Scavenger receptor CD163
CD22 Surface antigen
CD2BP2 CD2 binding protein
CD36 undefined
CD44 Raft associated hyaluronate transporter
CD6 undefined
Cdc25 undefined
Cdc42 undefined
CED-3-like C. elegans protein-like
ChEs Cholinesterases
C-jun undefined
CK Creatine kinase
c-kit undefined
CoaR undefined
COMMD1/MURR1 undefined
CooA undefined
Cox Cytochrome c oxidase
Cp Ceruloplasmin
Cp Ceruloplasmin
Cp Ceruloplasmin
CP43 undefined
CP43 undefined
CP43 undefined
CP47 undefined
CP47 undefined
CP47 undefined
Crk undefined
CS Citrate synthase
CS Citrate synthase
Cu,Zn SOD Copper, zinc superoxide dismutase
CuAOs Copper containing amine oxidases
CueO Bacterial cupreous oxidase
CueR undefined
Cyt b559 Cytochrome b559
D1 undefined
D1 undefined
D1 undefined
D2 undefined
D2 undefined
DAP Drostatin
DCT1 Divalent cation transporter
Dcytb Duodenal cytochrome b
deoxyMb undefined
Diff undefined
DISC Death inducing signalling complex
Dlar undefined
DMT1 undefined
DMT1 undefined
DMT1/DCT1/Nramp2 Divalent metal transporter
DNA Pol V undefined
Dorsal undefined
DpsA undefined
DT40 undefined
DtxR Difteria toxin repressor
ECAO E. coli CuAO

EDH1 undefined
EDH2 undefined
EDH3 undefined
EDH4 undefined
EGFP undefined
EHBP1 undefined
EHD EPS15 homology domain
eMAP undefined
Ena Drosophila-Enabled protein
Ena Enabled adapter protein
Ena undefined
Ena undefined
Ena/VASP Enabled/vasodilator-stimulated phosphoprotein
ENA/VASP undefined
Endo-H Endoglycosidase
Eps8 Epidermal growth factor receptor substrate
ER Estrogen receptor
ERK Extracellular signal-related kinase
ERK1 undefined
ERK2 undefined
ERP60 undefined
ERP72 undefined
Ess1/Pin1 Peptidyl prolyl cis/trans isomerise
EVH1 Ena/Vasp homology 1
EVH1 Enabled/Vasodilator stimulated phosphoprotein homology 1
EVH1 Enabled/VASP homology 1
EVH1 Enabled/VASP homology domain 1
EVH1/2 Ena/VASP homology 1/2
EVH2 undefined
EVL Enabled/vasodilator-stimulated phosphoprotein-like protein
EVl ENA/VASP like protein
EVL Ena/VASP-like
FAAH Fatty acid amide hydrolase
FE65 undefined
FEN2 Plasma membrane H+-pantothenate transporter
Fet3 undefined
Fet3p undefined
Fet3p undefined
Fet4 undefined
FixL undefined
FMS1 Polyamine oxidase from S. cerevisiae
Fpn Ferroportin
FPR Formyl peptide receptor
FPRL1 FPR related lipoxin A4 receptor
FRE1, 2 Ferrireductase
FRS2 Fibroblast growth factor receptor substrate 2
FSH Follicle-stimulating hormone
Ftr1 Membrane permease
Ftr1p undefined
Fur Ferric uptake regulator
Fyb/SLAP Fyn-binding and SLP-76 associated protein
Fyb/SLAP T cell signalling Fyn binding protein/SLP-76-associated protein
Fyn undefined
Gamma-GCS Gamma-glutamyl cysteine synthetase
Gamma-GTP Gamma-glutamyl transpeptidase
Gap1 GTPase-activating protein
GAPDH Glyceraldheyde-3-phosphate dehydrogenaseundefined
GAPDH undefined
GAPDH undefined
GCN5 undefined
GCS Globin coupled sensor
G-CSF granulocyte colony stimulating factor

G-CSF granulocyte-colony-stimulating factor
GDNF Glial cell line-derived neurotrophic factor
Gel Gelatinase
GFP Green fluorescent protein
GFP undefined
GH Growth hormone
GHRH Growth-hormone-releasing hormone
GHS-R G-protein coupled receptor
GIRK1 undefined
GIRK1 G protein-gated inwardly rectifying potassium channel
Glu-C undefined
Glut4 Insuline Responsive Glucose transporter
GLUT-4 Glucose transporter
GM130 undefined
GNBP Gram negative binding protein
GNBP-1 undefined
GNBP-3 undefined
Gp340 undefined
GPCRs G-protein coupled receptors
G-protein GTP binding protein
GPx Glutathione peroxidise
GR Glutathione reductase
Grb2 Growth factor receptor-bound 2
Grb2 undefined
GST Glutathione transferase
GST Glutathione transferase
GST Gluthatione transferase
GTPase undefined
GTPase undefined
GTPase undefined
HasA undefined
Hb Hemoglobin
Hb Hemoglobin
Hb Hemoglobin
Hb Hemoglobin
Hb Hemoglobin
Hb undefined
Hck undefined
HCS70 undefined
HCS73 undefined
HDL High density lipoprotein
HFE undefined
HFE undefined
HIV Tat undefined
HIV-1 RT HIV-1 reverse transcriptase
HIV-RT HIV reverse transcriptase
HLA-H undefined
HMG CoA reductase undefined
HMG-CoA reductase undefined
HMGCR HMG-CoA reductase
HMGCR undefined
HO-1 Heme oxygenase
Holo-AGAO holo A. Globiformis CuAO
Homer undefined
Homer undefined
Hp 1 undefined
Hp 1 Variant of the Hp gene
Hp 1-1 major phenotype of Hp
Hp 1-1 undefined
Hp 2 undefined
Hp 2 Variant of the Hp gene
Hp 2-1 major phenotype of Hp
Hp 2-1 undefined
Hp 2-2 major phenotype of Hp
Hp 2-2 undefined
Hp Haptoglobin
Hp Haptoglobin
Hp Haptoglobin
Hp Haptoglobin
Hp Haptoglobin
HP Hephaestin
Hp Human hephaestin
Hp undefined

Hp1 undefined
Hp1-1 Major phenotypic form of haptoglobin
Hp1-1 undefined
Hp2 undefined
Hp2-1 Major phenotypic form of haptoglobin
Hp2-1 undefined
Hp2-2 Major phenotypic form of haptoglobin
Hp2-2 undefined
HpA0 undefined
HPAO H. polymorpha CuAO
Hpr Haptoglobin related protein
Hpr Haptoglobin related protein
Hs Haemosiderin
HS7C undefined
HSF-1 Heat shock transcription factor 1
HSP1 undefined
HSP10 undefined
Hsp16.3 undefined
Hsp26 undefined
HSP27 undefined
HSP60 undefined
HSP60 undefined
HSP60 undefined
HSP70 Heat shock protein
HSP70 undefined
HSP70 undefined
HSP70 undefined
HSP70 undefined
HSP90 undefined
HSP90 undefined
HSPA8 undefined
hTIIα Human topoisomerase IIα
hTIIβ Human topoisomerase IIβ
HVA High voltage activated Ca++ channels
ICE-like Interleukin-1β converting enzyme-like
IdeR undefined
IgA1 proteases undefined
IgG undefined
IL-6 ploinflammatory cytokine Il-6
iNOS NO synthase
IP3Rs Inositol-1,4,5-trisphosphate receptors
IREF2 undefined
IREG1 undefined
IREG1 undefined
Ireg1 Ferroportin 1
IRP1 Iron regulatory proteins
IRP1 undefined
IRP2 Iron regulatory proteins
IRP2 undefined
IRPs 1 and 2 Iron regulatory proteins
IRSp53 undefined
IscA undefined
IscS undefined
IscU undefined
Itk undefined
JH/JHs Juvenile hormone(s)
KBD undefined
L undefined
Lamp 1 undefined
Lamp 2 undefined
Lamp Lysosome associated membrane protein
LasR undefined
LasR-LBD undefined
LDL Low density lipoprotein
LDL undefined
LDLR LDL receptor
LDLs Low density lipoproteins
LEKTI undefined
Lf Lactoferrin
LFA-1 lymphocyte function-associated

antigen 1
LFA-1 undefined
LfN N-terminal half-molecule of human lactoferrin
LH Luteinizing hormone
LPRs Low-density lipoprotein receptor related proteins
LPS undefined
LRP5 undefined
LRP6 undefined
LSD1 Histone lysine specific demetilase
LuxI/LuxR undefined
LuxR undefined
LVA Low voltage activated Ca++ channels
LXR Liver X receptor
Lyn undefined
Lys Lysozime
M undefined
MAE2 Malonamidase
MAO A undefined
MAO B Monoamine oxidase B
MAOs Monoamine oxidases
MAP2 Microtubule associated protein 2
MaPgb M. Acetivorans protoglobin
MAPK Mitogen activated protein kinase
MAPK Mitogen-activated protein kinase
MAPKAPK2 undefined
MARCO undefined
Mb Myoglobin
Mb(s) Myoglobin(s)
MbCO undefined
Mb-Xe undefined
MDC1 undefined
MENA Mammalian Ena
Mena Mammalian enabled
Mena Mammalian enabled adapter protein (Ena)
Mena undefined
MerR undefined
MerR undefined
MetAPs Methionine aminopeptidases
MFT Mitochondrial iron importer
MFT undefined
mGluR Metabotropic glutamate receptor
mGluRs metabotropic glutamate receptors
MHC undefined
MLE Muconate lactonizing enzyme
MntR undefined
MPO myeloperoxidase
MPR Mannose 6-phosphate receptor
MR Mandalate racemase
MRE11 undefined
MT metallothionein
MT Metallothioneins
MT1 undefined
MTP undefined
MTP1 Metal transporter protein
Mtp1 Metal transporter protein
MTP1 Ferroportin 1
Myc undefined
Myo32 undefined
Myo32 undefined
Myo32 undefined
NafY undefined
Nav 1.8 Tetrodoxin resistant sodium channel
NBS1 undefined
Nck undefined
Nck undefined
NCP Non collagen protein
Nedd-4 undefined

NEX4 C. elegans annexin
NFBD1 undefined
NF-E2 undefined
NF-kB undefined
NF-kB undefined
NF-kB undefined
NF-kB Nuclear Factor kB
NF-KB undefined
NGAL specific granule protein
NifS undefined
NifU undefined
NikR Nickel uptake regulator
NiSOD undefined
NK Neurokinin-like receptors
N-Mena undefined
Nod1 undefined
Nod2 undefined
NPC1 Niemann-Pick C1
NPC1 undefined
NPC1 undefined
NPC1 undefined
NPC1l1 Niemann-Pick C1-like 1
NPC1L1 Nieman-Pick C1 like 1 intestinal sterol transporter
NPC1L1 Nieman-Pick C1-like 1
NPC2 undefined
Npw38 undefined
NPY-like undefined
Nramp2 undefined
Nramp2 undefined
NS3 undefined
NUMB undefined
Ovo ovotransferrin
OxyR undefined
P130 cas undefined
P34 cdc2 undefined
P38 MAPK undefined
p38 undefined
P53BP2 p53 binding protein
PAOs Polyamine oxidases
PARP1 undefined
PbrR undefined
PEBP2/CBF undefined
Pgb Protoglobin
PGLYRP-1 undefined
PGLYRP-2 N-acetylmuramoyl-L-alanine amidase
PGLYRP-3 undefined
PGLYRPs undefined
PGRPI-alpha undefined
PGRPI-beta undefined
PGRPI-L undefined
PGRPI-LB undefined
PGRPI-LC undefined
PGRPI-LE undefined
PGRPI-S undefined
PGRPI-SA undefined
PGRP-L Long PGRPs
PGRP-LC undefined
PGRP-LE undefined
PGRPs Peptidoglycan recognition proteins
PGRPs Peptidoglycan recognition proteins
PGRP-S Short PGRPs
PGRP-S1 Drosophila PGRP
PGRP-SA Drosophila PGRPs
PGRP-SD Drosophila PGRP
PH Pleckstrin homology domain
PI3K Phosphatidyl inositol 3-kinase
PIKK Phosphoinositide-3-kinase-related protein kinase
Pin 1 undefined
PKC Protein kinase C
Plc gamma undefined
Plc gamma undefined
PmxB Polymyxin B
PNGase F undefined

PNGase-F peptide N-glycosidase F
PPAR Peroxisome proliferator activated receptor
PPLO P. Pastoris CuAO
PQBP-1 undefined
PRL Prolactin
Prrp proline-rich RNA-binding protein
PS Photosystem
PSAO Pea CuAO
PsbA undefined
PsbB undefined
PsbC undefined
PsbE undefined
PsbF undefined
PsbH undefined
PsbJ undefined
PsbK undefined
PsbN undefined
PsbO undefined
PsbU undefined
PsbV Cytochrome c550
PsbV undefined
PsbZ undefined
PSI Photosystem I
PSI Photosystem I
PSI Photosystem I
PSII Photosystem II
PSII Photosystem II
PSII Photosystem II
PSII Photosystem II
PSII Photosystem II
PSII Photosystem II
PSTPIP undefined
PTP1B Protein tyrosine phosphatise 1B
Rab11 undefined
Rab11 undefined
Rab11a undefined
Rab11Fip2 undefined
Rab4 undefined
Rab7 undefined
Rac undefined
Rad50 undefined
Raf undefined
Raf1 undefined
Ran undefined
Ran undefined
Ran undefined
RanBP1 undefined
Ras undefined
Rb21 undefined
Rccyt c' Rhodobacter capsulatus cytochrome c'
RET undefined
RhoA undefined
RT Reverse transcriptase
RTK undefined
RTK Receptor tyrosine kinase
RXR Retinoid X receptor
RyRs Ryanodine receptors
SAA Serum amyloid A
SCAP undefined
Scap undefined
SDH Succinate dehydrogenase
SdiA undefined
Sema6A-1 semaphorin 6A-1
Sema6A-1 Semaphorin 6A-1
SERCA undefined
SFR1 undefined
SFR1 undefined
SFR1 undefined
sGC Soluble Guanylate Cyclase
Shank undefined
SHP-2 Src homology 2 domain containing protein tyrosine phosphatise 2
SHSPs Small heat shock proteins
SM22 undefined
SMF1 Yeast manganese transporter

Smf1 undefined
SmtB undefined
SOD Superoxide dismutase
SOD Superoxide dismutase
SOD Superoxide dismutase
SOD1 undefined
SOD2 undefined
Sos undefined
SoxR undefined
SP Serine protease
Spa(AIM) undefined
Spreads Sprouty related proteins with an EVH1 domain
Spred sprouty-related protein with EVH1 domain
Spred-3 undefined
Sprouty undefined
Sprouty2 undefined
Sprouty3 undefined
SR-A Scavenger receptor
SR-AI undefined
SR-B undefined
SR-B1 Scavenger receptor class B type 1
Src undefined
Src undefined
SREBP-2 undefined
SST Somatostatin
SSTR Somatostatin receptor
SSTR2 undefined
SSTR23 undefined
STAT3 Signal transducer and activator of transcription 3
TASK-1 undefined
TCR T-cell receptor
TCTP/HRF undefined
TESK1 Testis-specific protein kinase-1
Tf Transferrin
TfR Transferrin receptor
TfR undefined
TfR1 Transferrin receptor 1
TfR2 Transferrin receptor 2
TIM Triosephosphate isomerise
TIP60 Histone acetyltransferase
TLF1 Trypanosome lytic factor-1
TLR2 undefined
TLR4 undefined
TLRs Toll like receptors
TNF undefined
TNF Tumor necrosis factor
TNF Tumor necrosis factor
TNFalpha Tumor necrosis factor alpha
TNF-α Tumor necrosis factor-α
Toll undefined
tPA Tissue plasminogen activator
TraR undefined
TRC8 undefined
TRH Thyrotropin-releasing hormone
TrkA undefined
TRPS Tryptophan synthase
TRPV5 undefined
TRPV6 undefined
TsH Thyroid-stimulating hormone
Tsk undefined
VAP-1 Vascular adhesion protein-1
VASP undefined
VASP Vasodilator stimulated phosphoprotein
VASP Vasodilator stimulated phosphoprotein
VASP Vasodilator-stimulated phosphoprotein
VASP vasodilator-stimulated phosphoprotein
VESL undefined
Vesl undefined

Vesl undefined
VP2 undefined
WASP Wiskott-Aldrich syndrome
WASP Wiskott-Aldrich syndrome protein
WASP Wiskott-Aldrich syndrome protein
WASP Wiskott-Aldrich syndrome protein
WH1 WASP homology 1
WIP WASP interacting protein
YAP Yes associated protein
Yes undefined
Yfh1p Frataxin homolog
ZntR undefined
ZO-1 undefined
γ-GT γ-Glutamil transpeptidase
γ-GT γ-Glutamil transpeptidase
ω-Aga IVA ω-Agatoxin IVA
ω-CTx GVIA ω-Conotoxin GVIA
ω-CTx MVIIC ω-Conotoxin MVIIC

Bibliography

[1] AB3P Corpus. http://www.ncbi.nlm.nih.gov/CBBresearch/Wilbur/

[2] S. Abiteboul, G. Lausen, H. Uphoff, and E. Waller. Methods and rules. *SIGMOD Rec.*, 22(2):32–41, 1993.

[3] F. N. Afrati, I. Karali, and T. Mitakos. On inheritance in object oriented datalog. In *IADT*, pages 280–289, 1998.

[4] H. Ao and T. Takagi. Alice: an algorithm to extract abbreviations from medline. In *J. Am. Med. Inform. Assoc.*, 12, pages 576-586, 2005.

[5] A&T Corpus. http://3.uvdb.dbcls.jp/ALICE/corpus/download.html

[6] Apache Lucene Library. http://lucene.apache.org

[7] R. Apweiler, A. Bairoch et al. UniProt: the Universal Protein knowledgebase. In *Nucleic Acids Research*, 32, 2004.

[8] P. Atzeni, P. Cappellari, and P. A. Bernstein. Model-independent schema and data translation. In *EDBT*, pages 368–385. Springer, 2006.

[9] P. Atzeni, P. Cappellari, R. Torlone, P. A. Bernstein, and G. Gianforme. Model-independent schema translation. *VLDB J.*, 17(6):1347–1370, 2008.

[10] P. Atzeni and G. Gianforme. Inheritance and polymorphism in datalog: an experience in model management. In *Information and Knowledge Bases XX*, pages 354–358, 2009.

[11] P. Atzeni, G. Gianforme, and P. Cappellari. Reasoning on data models in schema translation. In *FoIKS*, pages 158–177, 2008.

[12] P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, pages 209–220, 2003.

[13] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12. ACM, 2007.

[14] P. A. Bernstein, S. Melnik, and P. Mork. Interactive schema translation with instance-level mappings. In *VLDB*, pages 1283–1286, 2005.

[15] BioText Corpus. http://biotext.berkeley.edu/data.html

[16] A. J. Bonner and T. Imielinski. Reusing and modifying rulebases by predicate substitution. *J. Comput. Syst. Sci.*, 54(1):136–166, 1997.

[17] J. T. Chang, H. Schtze and R.B. Altman. Creating an Online Dictionary of Abbreviations from MEDLINE. In *Journal of American Medical Informatics Association (JAMIA)*, 9(6), pages 612-620, 2002.

[18] J. T. Chang and H. Schtze. Abbreviations in Biomedical Text. In *Text Mining for Biology and Biomedicine*, 2006.

[19] E. Charniak and M. Berland. Finding parts in very large corpora. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 57-64, 1999.

[20] P. Cimiano and J. Vlker. Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems*, pages 227-238, 2005.

[21] G. Dobbie and R. W. Topor. A model for sets and multiple inheritance in deductive object-oriented systems. In *DOOD*, pages 473–488, 1993.

[22] G. Dobbie and R. W. Topor. Representing inheritance and overriding in datalog. *Computers and Artificial Intelligence*, 13:133–158, 1994.

[23] J. Euzenat, T.L. Bach, J. Barrasa, P. Bouquet, J.D. Bo, R. Dieng, M. Ehrig, M. Hauswirth, M. Jarrar, R. Lara, D. Maynard, A. Napoli, G. Stamou, H. Stuckenschmidt, P. Shvaiko, S. Tessaris, S.V. Acker and I. Zaihrayeu. D2.2.3: State of the art on ontology alignment. In *Knowledgeweb*, 2004.

[24] K. Fukuda, T. Tsunoda, A. Tamura, T. Takagi. Toward Information Extraction: Identifying protein names from biological papers. In *PSB1998*, pages 705-716, 1998.

[25] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project - back and forth between theory and practice. In *PODS*, pages 1–12, 2004.

[26] J. Hoffart, F. Suchanek, K. Berberich and G. Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. In *Research Report MPI-I-2010-5.007, Max-Planck-Institut fr Informatik*, 2010.

[27] H. M. Jamil. Implementing abstract objects with inheritance in datalog$^{\text{neg}}$. In *VLDB*, pages 56–65, 1997.

[28] D. Kensche, C. Quix, M. A. Chatti, and M. Jarke. Gerome: A generic role based metamodel for model management. *J. Data Semantics*, 8:82–117, 2007.

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

[30] C. Kuo, M. HT Ling, K.T. Lin and C.N. Hsu. BIOADI: a machine learning approach to identifying abbreviations and definitions in biological literature. In *BMC Bioinformatics*, vol. 10, 2009.

[31] L. Larkey, P. Ogilvie, A. Price and B. Tamilio. Acrophile: An Automated Acronym Extractor and Server. In *Proceedings of the ACM Digital Libraries conference*, pages. 205-214, 2000.

[32] Lingpipe. http://alias-i.com/lingpipe/

[33] H. Liu, A.R. Aronson and C. Friedman. A Study of Abbreviations in MEDLINE Abstracts. In *AMIA, Annual Symposium Proceedings*, 2002.

[34] M. Liu, G. Dobbie, and T. W. Ling. A logical foundation for deductive object-oriented databases. *ACM Trans. Database Syst.*, 27(1):117–151, 2002.

[35] S. Massmann, S. Raunich, D. Aumueller, P. Arnold and E. Rahm. Evolution of the COMA Match System. In *The Sixth International Workshop on Ontology Matching (OM-2011)*, 2011.

[36] MEDSTRACT Gold Standard Corpus. http://www.medstract.org/index.php?f=gold-standard

[37] Military Acronyms, Initialisms and Abbreviations. http://www.fas.org/news/reference/lexicon/acronym.htm

[38] D. Milne and I.H. Witten. An open-source toolkit for mining Wikipedia by: D. Milne, and I.H. Witten. In *Proceedings of New Zealand Computer Science Research Student Conference, (NZCSRSC)*, Vol. 9, 2009.

[39] P. Mork, P. A. Bernstein, and S. Melnik. Teaching a schema translator to produce O/R views. In *ER*, pages 102–119. Springer, 2007.

[40] A. Mycroft and R. A. O'Keefe. A polymorphic type system for PROLOG. *Artif. Intell.*, 23(3):295–307, 1984.

[41] D. Nadeau and P.D. Turney. A Supervised Learning Approach to Acronym Identification. In *18th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI*, 2005.

[42] Y. Park and R.J. Byrd. Hybrid Text Mining for Finding Abbreviations and Their Definitions. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001.

[43] PubMed website. http://www.ncbi.nlm.nih.gov/pubmed/

[44] PubMed Central website. http://www.ncbi.nlm.nih.gov/pmc/

[45] J. Pustejovsky, J. Castao, B. Cochran, M. Kotecki, M. Morrell and A. Rumshisky. Automatic Extraction of Acronym-meaning Pairs from MEDLINE Databases. In *MEDINFO*, 2001.

[46] A. Schwartz and M. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical texts. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, 2003.

[47] S&H implementation. http://biotext.berkeley.edu/code/abbrev/ExtractAbbrev.java

[48] S. Sohn, D. C. Comeau, W. Kim, W. J. Wilbur. Abbreviation definition identification based on automatic precision estimates. In *BMC Bioinformatics*, 9, 2008.

[49] The Stanford NLP Group. http://nlp.stanford.edu/index.shtml

[50] R. Subhashini and J. Akilandeswari. A survey on ontology construction methodologies. In *International Journal of Enterprise Computing and Business Systems (Online)*, Vol. 1, No. 1, 2011.

[51] K. Taghva and J. Gilbreth. Recognizing acronyms and their definitions. In *International journal on Document Analysis and Recognition*, pages 191-198, 1999.

[52] The Official Homepage of the United States Army. http://www.army.mil

[53] The UniProt Database. http://www.uniprot.org

[54] Y. Wang, J. Vlker and P. Haase. Towards Semi-automatic Ontology Building Supported by Large-scale Knowledge Acquisition. In *AAAI Fall Symposium On Semantic Web for Collaborative Knowledge Acquisition*, Vol. FS-06-06, pages 70-77, 2006.

[55] S. Yeates. Automatic extraction of acronyms from text. In *Third New Zealand Computer Science Research Students' Conference*, pages 117-124, 1999.

[56] M. Yoshida, K. Fukuda, T. Takagi. PNAD-CSS: a workbench for constructing a protein name abbreviation dictionary. In *Bioinformatics*, 16(2), pages 169-175, 2000.

[57] H. Yu, G. Hripcsak and C. Friedman. Mapping abbreviations to full forms in biomedical articles. In *J. Am. Med. Inform. Assoc.*, 9, pages 262-272, 2002.