



Question Answering System in a Business Intelligence Context

Nicolas Kuchmann-Beauger

► To cite this version:

Nicolas Kuchmann-Beauger. Question Answering System in a Business Intelligence Context. Other. Ecole Centrale Paris, 2013. English. NNT : 2013ECAP0021 . tel-00799246v2

HAL Id: tel-00799246

<https://theses.hal.science/tel-00799246v2>

Submitted on 16 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ÉCOLE CENTRALE PARIS
ÉCOLE DOCTORALE 287
SCIENCES POUR L'INGÉNIEUR

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'École Centrale Paris

Mention : INFORMATIQUE

Présentée et soutenue par

Nicolas KUCHMANN-BEAUGER

Question Answering System in a Business Intelligence Context

Thèse dirigée par Marie-Aude AUFAURE

préparée à SAP BusinessObjects / SAP Research
et au Laboratoire MAS (EA 4037)

soutenue le 15 février 2013

Jury :

<i>Rapporteurs :</i>	Matthieu ROCHE	-	Université Montpellier 2
	Juan Carlos TRUJILLO MONDÉJAR	-	Universidad de Alicante
<i>Directeur :</i>	Marie-Aude AUFAURE	-	École Centrale Paris
<i>Examineurs :</i>	Patrick MARCEL	-	Université de Tours
	Yannick CRAS	-	SAP Research
	Élisabeth MÉTAIS	-	CNAM Paris
<i>Invité :</i>	Philippe MEINIEL	-	SAP France

Acronyms

AI	Artificial Intelligence	11
BI	Business Intelligence.....	vii
CMS	Content Management System.....	8
CRM	Customer Relationship Management	7
DBMS	Dababase Management System	3
ERP	Enterprise Resource Planning.....	7
FOL	First Order Logic.....	45
GUI	General User Interface.....	23
IE	Information Extraction	75
IR	Information Retrieval	vi
MDX	MultiDimensional eXpression.....	16
NER	Named entity recognizer.....	64
NL	Natural Language	v
NLP	Natural Language Processing.....	42
OLAP	Online analytical Processing.....	5
Q&A	Question Answering	2
RDF	Resource Description Framework	16
SparQL	SPARQL Protocol and RDF Query Language	16
SQL	Structured Query Language	16
SW	Semantic Web.....	16
Web	World Wide Web	8

Contents

Acronyms	iii
Contents	v
List of Figures	ix
List of Tables	xi
Listings	xiii
1 Introduction	1
1.1 Structured data	2
1.1.1 Relational model	3
1.1.2 Multidimensional model	4
1.2 BI and the need for relevant answers	7
1.2.1 Introduction to Business Intelligence	7
1.2.2 Mobility in BI and the growing popularity of natural search interfaces	7
1.2.3 BI and corporate settings	8
1.3 Q&A and its benefits for BI	9
1.3.1 WATSON, a success story	11
1.3.2 Challenges in the field of structured data search	11
1.4 Problem formulation	12
1.4.1 Machine translation and pivot languages	12
1.4.2 What is not of the scope of our work	13
1.5 Contribution to the state of the art	13
1.6 Organisation of the thesis	14
2 State of the Art	15
2.1 Main Dimensions	17
2.1.1 Data sources	18
2.1.2 Users' questions	18
2.1.3 Internal representation	20
2.1.4 Database queries	21
2.1.5 Domain-independence	23
2.1.6 Portability	23
2.1.7 Metrics	24
2.2 Anatomy of Natural Language (NL) interfaces systems	25
2.2.1 Lexicon	25

2.2.2	Semantic rules	25
2.2.3	Main problems to solve	26
2.3	Taxonomy of main approaches	26
2.3.1	Domain-dependent semantic parsing	26
2.3.2	Complex question translation	28
2.3.3	Feedback-driven approaches	28
2.3.4	Learning-based approaches	29
2.3.5	Schema-unaware approaches	30
2.4	Domain-dependent semantic parsing	31
2.4.1	BASEBALL [25]	31
2.4.2	LUNAR [72]	33
2.5	Complex question translation	34
2.5.1	CHAT-80 [69]	34
2.5.2	QWERTY [49]	36
2.5.3	IRUS [6]	37
2.5.4	PRECISE [54, 53]	38
2.5.5	PANTO [68]	39
2.6	Feedback-driven approaches	40
2.6.1	TEAM [28]	41
2.6.2	MASQUE/SQL [2]	43
2.6.3	NALIX [42] and DaNaLIX [41]	44
2.6.4	C-PHRASE [47]	45
2.6.5	ORAKEL [12]	46
2.7	Learning-based approaches	47
2.7.1	Miller et al. [46]	47
2.7.2	Zettlemoyer et Collins [76]	48
2.7.3	WOLFIE [66]	49
2.8	Schema-unaware approaches	50
2.8.1	POWERAQUA [44]	50
2.8.2	DEEPQA [16]	52
2.9	Challenges	54
2.10	Conclusion	54
3	Personalized and Contextual Q&A	57
3.1	Q&A Framework	59
3.1.1	Authentication	59
3.1.2	Search & prediction services	61
3.1.3	Answering system	62
3.1.4	Search engines	67
3.2	Extensibility of the framework	68
3.2.1	Implementing new plugins	68
3.2.2	Configuration of linguistic resources	69
3.3	Multithreading and performance considerations	72
3.4	Conclusion	73
3.4.1	Summary & discussion	73
3.4.2	Organization of the next chapters	73
4	Linguistic patterns	75
4.1	Linguistic patterns in Information Retrieval (IR)	76
4.2	Patterns for structured information	77

4.2.1	Running example	77
4.2.2	Users' graphs and annotations	79
4.2.3	Feature	79
4.2.4	Parse graph	82
4.2.5	Pattern	83
4.2.6	Structured queries	87
4.2.7	Query logs	88
4.3	Ranking the results	88
4.3.1	Confidence	88
4.3.2	Selectivity	88
4.3.3	Complexity	88
4.3.4	Metrics from query logs	89
4.3.5	Overall measure	91
4.4	Summary & discussion	91
4.4.1	Learning approaches	92
4.4.2	Authoring tool	95
5	Query Modeling	97
5.1	Fact tables and functional dependencies	97
5.2	Multidimensional queries and preferences	98
5.2.1	Formal representation of a query	99
5.2.2	Database queries	102
5.2.3	Personalization of multidimensional queries	103
5.2.4	Example of how patterns are mapped to structured queries	107
5.3	Similarity measure based on preferences	109
5.4	Prediction model	111
5.4.1	Motivating Scenario	111
5.4.2	Architecture	112
5.4.3	Query Clustering	113
5.4.4	Query Prediction	114
5.5	Summary & discussion	116
6	Experiments & evaluation	119
6.1	Evaluating an IR system	119
6.1.1	Classic evaluation metrics	120
6.2	Evaluation proposal	120
6.2.1	US Census Bureau data	121
6.2.2	Evaluation corpus	122
6.2.3	Performance results	128
6.2.4	Evaluation results	128
6.3	Auto-completion – an experiment on the search platform	131
6.3.1	Usage statistics in Business Intelligence (BI) documents	131
6.3.2	Collaborative co-occurrence measure	132
6.3.3	Query expansion	133
6.3.4	Results of the experimentation	134
7	Conclusion	135
7.1	Summary	135
7.2	Challenges	138
7.2.1	Personalized patterns	138

7.2.2	Linguistic coverage	138
7.2.3	Prediction model	138
A	Pattern	139
B	Query logs	143
C	Mutldimensional queries	145
C.1	Automatic generation of SQL/MDX queries	145
D	Application screenshots	147
D.1	Desktop application	147
D.2	iPhone™/iPad™ application	148
D.3	Search results	148
E	Source code	151
	Bibliography	159

List of Figures

1.1	Evolution of the number of users with company growth	2
1.2	Conceptual model of the <i>EFashion</i> dataset used for experimenting our system	3
1.3	Multidimensional data schema of the dataset <i>eFashion</i>	5
1.4	State-of-the-art BI tools for exploring data	6
1.5	Graphic interface to build multidimensional queries	8
1.6	Charts generated in an Excel instance must be of relatively small size compared to other applications like a dashboard	10
1.7	Chart rendered on a desktop application	11
1.8	Two classic translation appraoches	13
2.1	Example of parse tree for the question “Who produced the most films?” from [67].	21
2.2	Linguistic coverage vs. logical coverage	24
2.3	Big picture of NL interfaces systems	26
3.1	General architecture of the answering system	58
3.2	Snapshot of corporate social network of 18 users sharing two types of relations: ‘reportsTo’ and ‘hasBusinessContact’	60
3.3	HTTP/REST communication between the back-end and front-end systems	61
3.4	Architecture of the answering system	62
3.5	Example of chart rendered without any knowledge on dimensions’ data types	65
3.6	Two visualization types corresponding to the same dataset	67
3.7	Search engines implemented as plug-ins	68
3.8	Average overall processing time depending on the number of threads being executed in parallel	72
4.1	Translating a question in a structured query	78
4.2	Example of a parse graph	80
4.3	Example for parse graph constraints and mapping rules to generate a structrued query	83
4.4	Case-based reasoning approach applied to the problem of pattern learning	92
4.5	Classification problem involved for a large number of patterns	93
4.6	Mutation and reward of individuals	94
4.7	<i>Novelty</i> and <i>efficiency</i> of newly-created individuals	94
4.8	Authoring tool	95

4.9	Graphic construction of the pattern	96
5.1	Decomposition of <i>axis groups</i> in the formal query representation . .	100
5.2	Automatic database query generation	103
5.3	Architecture of the module responsible for predicting and recommending queries	113
5.4	Illustration of the Markov chains	115
6.1	A few tables from the census dataset	121
6.2	ManyEyes collaborative platform	123
6.3	Popular dataset tags on ManyEyes	124
6.4	Processing time before and after the chart generation process as a function of the schema input nodes count	128
6.5	Success of answering goldstandard queries compared to WolframAlpha™	129
6.6	Variant of success of answering goldstandard queries compared to WolframAlpha™	130
6.7	Search-box of the answering system's front-end with an auto-completion implementation based on collaborative metrics	131
D.1	Screenshot of the desktop application	147
D.2	Screenshot of the iPhone™ application	148
D.3	Example of report with two hierarchies (time and geographic hierarchies)	149

List of Tables

1.1	Data growth trends	1
2.1	Question classification for temporal databases from [57]	20
2.2	Semantic meaning representations	21
2.3	Overview of major NL interfaces to structured data. ‘Q’ stands for ‘Query’, ‘D’ for ‘Data’, ‘A’ for ‘Answer’, ‘P’ for ‘Portability’, ‘L’ for ‘Linguistic coverage’ and ‘E’ for ‘Error feedback’	27
2.4	Different statistical models used by learning-based systems	30
2.5	Volume of training data of different systems	30
2.6	Systems belonging to the <i>domain-dependant parsing</i> range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “Shortcomings”.	31
2.7	Systems belonging to the <i>complex question translation</i> range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.	35
2.8	Systems belonging to the <i>feedback-driven</i> range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.	41
2.9	Systems that incorpore machine learning approaches in order to interface structured data. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.	48
2.10	Systems belonging to the <i>schema-unaware</i> range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.	50
3.1	Examples of named entity types used by the basic query plugin . . .	63
3.2	Examples of NL features that are annotated	64
3.3	Facts answering the question “Sales target per department in 2001”. . .	66
4.1	The same idea can be represented in different ways. The second column expresses constraints that are satisfied by all questions from the first column	76
4.2	Features and annotations for the example (4.3 on page 77)	81
4.3	Annotation types in the parse graph	82
4.4	Example of selectivity metrics for the example question (4.3) and the dataset <i>eFashion</i> (see figure 1.3 on page 5)	89
5.1	Facts answering the question “Sales target per department in 2001”. . .	98
5.2	Some chart types and their associated analysis types	106

6.1	Comparison of logical names and terms used in the data model of the Census dataset	121
6.2	Comparison of the sizes of tables from Census dataset	122
6.4	Average precisions of QUASL and WolframAlpha™	130
6.5	Top-5 dimensions that most co-occur with the “Sales revenue” measure	134
D.1	Example of a cross-table with two hierarchies	148

Listings

2.1	Example of SparQL query template operating on top of RDF data	17
3.1	Interface of any query plugin	69
3.2	Interface of any search plugin	69
3.3	Interface of any answer plugin	69
3.4	NL feature definition	70
5.1	SQL query generated from conceptual query (5.4)	101
5.2	SQL query generated from conceptual query (5.4)	101
5.3	Example of MDX query	102
5.4	Structured query generated from conceptual query (5.1)	103
A.1	Pattern matching queries like “Revenue per state in 2001?”	139
B.1	Query log example	143
C.1	SQL query generation (1)	145
C.2	SQL query generation (2)	145
E.1	Search plugin for the pattern approach	151

Acknowledgements

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, I address my utmost gratitude to Prof. Marie-Aude Aufaure who kindly accepted to supervise my thesis, and to guide my research.

I want to thank Chahab Nastar and Alexis Naibo who took me aboard at SAP, and Philippe Meiniel who gave me many accurate advices and expressed his technical knowledge on many subjects.

I would also like to thank my colleagues and staff at SAP Research in Paris and in Dresden on the one hand, and at École Centrale Paris in Châtenay-Malabry on the other hand for the good times we spent together.

Of course, I must be thankful to my dear friends who always supported me in their own way.

Last but not the least, I thank my dear parents and family for supporting me throughout my studies and giving me the strength to not giving up.

Abstract

The amount and complexity of data generated by information systems keep increasing in Warehouses. The domain of Business Intelligence (BI) aims at providing methods and tools to better help users in retrieving those data. Data sources are distributed over distinct locations and are usually accessible through various applications. Looking for new information could be a tedious task, because business users try to reduce their work overload. To tackle this problem, Enterprise Search is a field that has emerged in the last few years, and that takes into consideration the different corporate data sources as well as sources available to the public (e.g. World Wide Web pages). However, corporate retrieval systems nowadays still suffer from information overload. We believe that such systems would benefit from Natural Language (NL) approaches combined with Q&A techniques. Indeed, NL interfaces allow users to search new information in their own terms, and thus obtain precise answers instead of turning to a plethora of documents. In this way, users do not have to employ exact keywords or appropriate syntax, and can have faster access to new information. Major challenges for designing such a system are to interface different applications and their underlying query languages on the one hand, and to support users' vocabulary and to be easily configured for new application domains on the other hand.

This thesis outlines an end-to-end Q&A framework for corporate use-cases that can be configured in different settings. In traditional BI systems, user-preferences are usually not taken into account, nor are their specific contextual situations. State-of-the art systems in this field, SODA¹ and SAFE² do not compute search results on the basis of users' situation. This thesis introduces a more personalized approach, which better speaks to end-users' situations. Our main experimentation, in this case, works as a search interface, which displays search results on a dashboard that usually takes the form of charts, fact tables, and thumbnails of unstructured documents. Depending on users' initial queries, recommendations for alternatives are also displayed, so as to reduce response time of the overall system. This process is often seen as a kind of prediction model.

Our work contributes to the following: first, an architecture, implemented with parallel algorithms, that leverages different data sources, namely structured and unstructured document repositories through an extensible Q&A framework, and this framework can be easily configured for distinct corporate settings; secondly, a constraint-matching-based translation approach, which replaces a pivot language with a conceptual model and leads to more personalized multidimensional queries; thirdly, a set of NL patterns for translating BI questions in structured queries that can be easily configured in specific settings. In addition, we have implemented an iPhone/iPadTM application and an HTML front-end that demonstrate the feasibility of the various approaches developed through a series of evaluation metrics for the core component and scenario of the Q&A framework. To this end, we elaborate on a range of gold-standard queries that can be used as a basis for evaluating retrieval systems in

¹L. Blunschi, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger (2011) Data-thirsty business analysts need SODA: search over data warehouse. CIKM 2011.

²G. Orsi, P. Milano, P. Leonardo, L. Tanca, E. Zimeo, and U. Sannio (2011) Keyword-based , Context-aware Selection of Natural Language Query Patterns. EDBT 2011.

this area, and show that our system behave similarly as the well-known WolframAlpha™ system, depending on the evaluation settings.

Résumé

Le volume et la complexité des données générées par les systèmes d'information croissent de façon singulière dans les entrepôts de données. Le domaine de l'informatique décisionnelle (aussi appelé BI) a pour objectif d'apporter des méthodes et des outils pour assister les utilisateurs dans leur tâche de recherche d'information. En effet, les sources de données ne sont en général pas centralisées, et il est souvent nécessaire d'interagir avec diverses applications. Accéder à l'information est alors une tâche ardue, alors que les employés d'une entreprise cherchent généralement à réduire leur charge de travail. Pour faire face à ce constat, le domaine « Enterprise Search » s'est développé récemment, et prend en compte les différentes sources de données appartenant aussi bien au réseau privé d'entreprise qu'au domaine public (telles que les pages Internet). Pourtant, les utilisateurs de moteurs de recherche actuels souffrent toujours de du volume trop important d'information à disposition. Nous pensons que de tels systèmes pourraient tirer parti des méthodes du traitement naturel des langues associées à celles des systèmes de questions/réponses. En effet, les interfaces en langue naturelle permettent aux utilisateurs de rechercher de l'information en utilisant leurs propres termes, et d'obtenir des réponses concises et non une liste de documents dans laquelle l'éventuelle bonne réponse doit être identifiée. De cette façon, les utilisateurs n'ont pas besoin d'employer une terminologie figée, ni de formuler des requêtes selon une syntaxe très précise, et peuvent de plus accéder plus rapidement à l'information désirée. Un challenge lors de la construction d'un tel système consiste à interagir avec les différentes applications, et donc avec les langages utilisés par ces applications d'une part, et d'être en mesure de s'adapter facilement à de nouveaux domaines d'application d'autre part.

Notre rapport détaille un système de questions/réponses configurable pour des cas d'utilisation d'entreprise, et le décrit dans son intégralité. Dans les systèmes traditionnels de l'informatique décisionnelle, les préférences utilisateurs ne sont généralement pas prises en compte, ni d'ailleurs leurs situations ou leur contexte. Les systèmes état-de-l'art du domaine tels que SODA¹ ou SAFE² ne génèrent pas de résultats calculés à partir de l'analyse de la situation des utilisateurs. Ce rapport introduit une approche plus personnalisée, qui convient mieux aux utilisateurs finaux. Notre expérimentation principale se traduit par une interface de type *search* qui affiche les résultats dans un *dashboard* sous la forme de graphes, de tables de faits ou encore de miniatures de pages Internet. En fonction des requêtes initiales des utilisateurs, des recommandations de requêtes sont aussi affichées en sus, et ce dans le but de réduire le temps de réponse global du système. En ce sens, ces recommandations sont comparables à des prédictions.

Notre travail se traduit par les contributions suivantes : tout d'abord, une architecture implémentée via des algorithmes parallélisés et qui prend en compte la diversité des sources de données, à savoir des données structurées ou non structurées dans le cadre d'un *framework* de questions-réponses qui peut être facilement configuré dans des environnements différents. De plus, une approche de traduction basée sur la résolution

¹L. Blunski, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger (2011) Data-thirsty business analysts need SODA : search over data warehouse. CIKM 2011.

²G. Orsi, P. Milano, P. Leonardo, L. Tanca, E. Zimeo, and U. Sannio (2011) Keyword-based , Context-aware Selection of Natural Language Query Patterns. EDBT 2011.

de contrainte, qui remplace le traditionnel langage-pivot par un modèle conceptuel et qui conduit à des requêtes multidimensionnelles mieux personnalisées. En outre, en ensemble de patrons linguistiques utilisés pour traduire des questions BI en des requêtes pour bases de données, qui peuvent être facilement adaptés dans le cas de configurations différentes. Enfin, nous avons implémenté une application pour iPhone/iPadTM et une interface de type « HTML » qui démontre la faisabilité des différentes approches développées grâce à un ensemble de mesures d'évaluations pour l'élément principal (le composant de traduction) et un scénario d'évaluation pour le framework dans sa globalité. Dans ce but, nous introduisons un ensemble de requêtes pouvant servir à évaluer d'autres système de recherche d'information dans le domaine, et nous montrons que notre système se comporte de façon similaire au système de référence WolframAlphaTM, en fonction des paramètres d'évaluation.

Chapter 1

Introduction

Outline

1.1	Structured data	2
1.1.1	Relational model	3
1.1.2	Multidimensional model	4
1.2	BI and the need for relevant answers	7
1.2.1	Introduction to Business Intelligence	7
1.2.2	Mobility in BI and the growing popularity of natural search interfaces	7
1.2.3	BI and corporate settings	8
1.3	Q&A and its benefits for BI	9
1.3.1	WATSON, a success story	11
1.3.2	Challenges in the field of structured data search	11
1.4	Problem formulation	12
1.4.1	Machine translation and pivot languages	12
1.4.2	What is not of the scope of our work	13
1.5	Contribution to the state of the art	13
1.6	Organisation of the thesis	14

The amount of data that is stored in companies is growing over time. As an example, table 1.1 shows the evolution of the proportion of companies storing more than 1TB and more than 100TB in 2009 and 2010 (from [62]). These

Year	companies storing more than 1TB	companies storing more than 100TB
2009	74%	24%
2010	87%	29%

Table 1.1 – Data growth trends

growing data must be stored in appropriate data structures and algorithms should be implemented in such a way that these data can be queried in an efficient manner. Databases are generally distributed over different data sources and can be corrupted or data can be redundant. Data loading from production

systems is performed by an extraction transformation loading (ETL) tool to be then analyzed by BI tools to generate reports and dashboards.

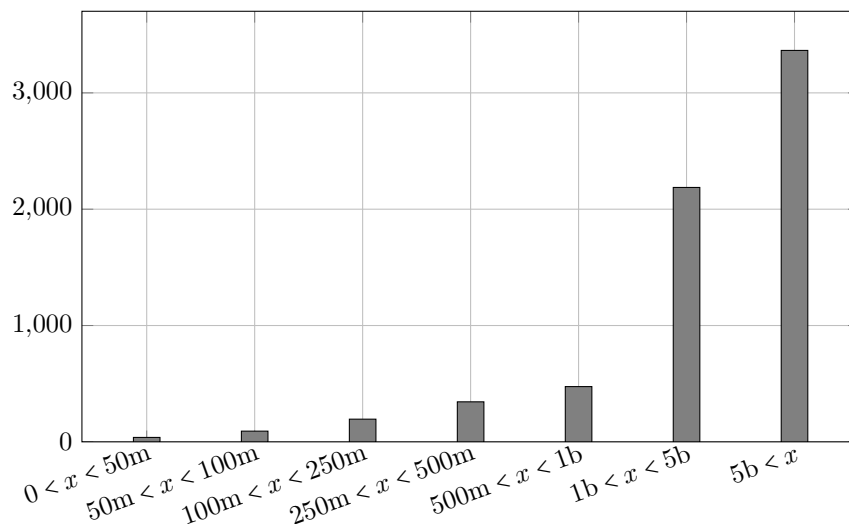


Figure 1.1 – Evolution of the number of users with company growth

As shown on the chart Figure 1.1, the number of users grows with the growth of companies (with respect to revenue). A direct consequence is an increased amount of documents generated by users. As a result, users are faced with the problem of *information overload*. Indeed, many corporate applications are entry points for information. Besides, searching for information in a single application (like the company's intranet) is not an easy task, because of information overload, and because the order according to which result items are ordered is not based on relevancy for the specific user.

In our thesis, we tackle this problem in supporting information access. Indeed, our contributions are:

- a framework that allows natural language search over data warehouses
- a question answering system that offers personalized results

First, we present how data are structured in warehouses. Second, we introduce the BI domain and some of current challenges in this area. Then, we present how BI can benefit from techniques used in Question Answering (Q&A) systems. Finally, we formulate the main problem of interest of our work.

1.1 Structured data

The history of structured data has followed the history of physical storage devices. Indeed, improved storage capabilities has made it possible to store larger databases. Recent years have seen a major turning point in database history. Indeed, state-of-the-art database management systems rely on main memory instead of classic disk storage.

The way data are organized in the database is defined in a *model*: it defines how elements are organized from a semantic point of view. Early data

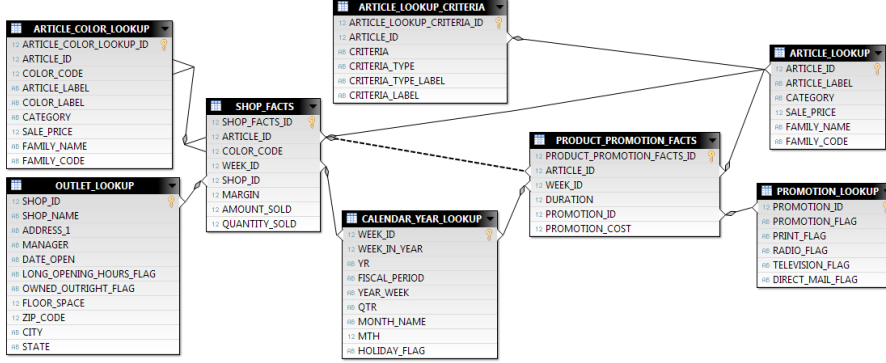


Figure 1.2 – Conceptual model of the *EFashion* dataset used for experimenting our system

structures like *specification lists* used in the BASEBALL NL interface [25] are based on hierarchical relations between database entities. We reproduce below an example of specification list (from [26]):

Month = July
 Place = Boston
 Day = 7
 Game Serial No. = 96
 (Team = Red Sox, Score = 5)
 (Team = Yankees, Score = 3)

This structure combines *hierarchies* (for instance, the game occurs at a specific place on a specific month) and *associations* marked with parentheses.

In modern database systems, there are several abstraction layers for representing data structures:

- *Physical* layer: Database Management System (DBMS)-specific data structure
- *Logical* layer: data organization which eases database administration (usually performed by database administrator). The most popular logical representation of data is based on the relational architecture
- *Conceptual* layer: domain- and application-specific representation of relations, dependencies and constraints of data.

We further present the relational model in section 1.1.1 and introduce the multidimensional model which is widely used to model data warehouses in section 1.1.2.

1.1.1 Relational model

Relational model is a logic model which defines a set of *relationships* (or *tables*). Each relationship is described with *attributes*. Besides, each relationship is identified with a set of its attributes (this set is then called *primary key* of

the relationship). Some attributes that reference primary keys of other relationships are called *foreign keys*.

This model is used by many modern DBMS, and therefore it has been popular for the last few decades. The data are usually first *conceptually* designed by domain experts, in some visual representation.

We reproduce figure 1.2 an example of conceptual model on the basis of the *eFashion* dataset, which is described Appendix ???. In this model, the *relationships* are represented as boxes (for instance `ARTICLE_LOOKUP`) and attributes (e.g. `ARTICLE_ID`) are represented with rows. The constraints of uniqueness (i.e. the combination of primary and foreign keys) are depicted with links between boxes attributes.

1.1.2 Multidimensional model

Multidimensional models are popular, especially for BI purposes, because they “facilitate complex analyses and visualization” [10]. Data warehousing is a collection of decision support technologies and aims at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions [10]. Indeed, the relational model dating from the 70’s is not optimized for the processing of huge amount of data. For instance, knowledge workers will process this amount of data and aggregate them at different levels of granularity along different *analysis dimensions* to understand some *facts* that seem unusual. Aggregating data is a very expensive task and therefore the need for a new model has emerged over years.

This model has influenced front-end tools, database design and query engines for OLAP [10]. A classic definition of multidimensional models has been proposed by Golfarelli et Rizzi [23]. An intuitive representation is a multidimensional cube, where each axis corresponds to a dimension of the model and a cell of the cube corresponds to a fact instance aggregated along the different dimensions. The objects of analysis are numeric measures (e.g. sales, budget, revenue, etc.). Each numeric measure depends on a set of dimensions which provide the context for the measure [10]. Each dimension is described by a set of attributes. Attributes can be related to each other through a hierarchy of relationships [10]. Figure 1.3 is an example of a multidimensional data model. In this figure, two fact tables (‘sales’ and ‘reservations’) contain several measures (e.g. ‘days’, ‘revenue’ and ‘reservation days’). Rounded nodes correspond to dimensions organized in hierarchies. The depth of the node (i.e. the distance from the fact table) to the dimension corresponds to the level of the dimension in the hierarchy. Finally, attributes at a specific hierarchy level are underlined (e.g. ‘age’, ‘phone number’ and ‘invoice year’).

As introduced above, multidimensional architecture is usually preferred to relational one, when there is a huge amount of data to aggregate. Therefore, some DBMS implementations optimize some of these operations, and the multidimensional architecture is preferred to the relational one. In some implementations, the multidimensional architecture can be used directly with relational databases, because the targetted applications prefer the multidimensional representation. Hybrid architectures have also appeared, where the logical layer is composed of both relational tables (for instance for representing large quantities of detailed data) and multidimensional implementations for less detailed / more aggregated data.

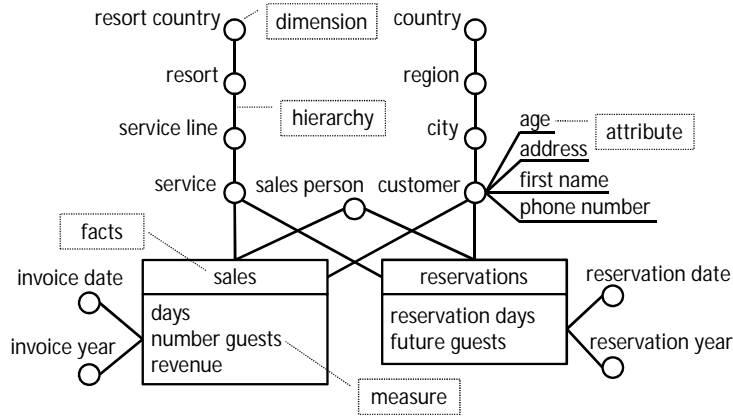


Figure 1.3 – Multidimensional data schema of the dataset *eFashion*

We present in the following the star data schema, which is probably the most popular one. Then we explain why multidimensional models are used in BI. In addition, we introduce the notion of functional dependencies which brings constraints in multidimensional data models. Finally, we briefly introduce database query languages.

Star data schema

A popular implementation representing the multidimensional data model is the *star schema* [10]. It consists of a single fact table and surrounding tables for each dimension. A refinement of this model is the *snowflake schema*, where the (dimensional) hierarchy is represented by normalizing the dimension tables.

Star-schema is the best-known schema for modeling data warehouses. Typically the star-schema is composed of several tables in the middle called *fact tables*. These fact tables correspond to the measures. All tables around these fact tables correspond to *analysis axes* or *dimensions*. The star-schema is actually a special case of the snowflake-schema, which is not further detailed here¹.

The model displayed figure 1.2 is organized according to the star-schema: two fact tables (**SHOP_FACTS** and **PRODUCT_PROMOTION_FACTS**) are surrounded by several tables corresponding to different analysis axes.

Online analytical Processing (OLAP) operations

Standard operations on the cube are called OLAP operations: rollup (increase the level of aggregation), drill-down (decrease the level of aggregation) along one or more dimension hierarchies, slice and dice (selection and projection) and pivot (reorient the multidimensional view of data) [10]. Front-end tools let users execute these operations. Figure 1.4 is a screenshot of a BI tool that

¹The interested reader can refer to http://en.wikipedia.org/wiki/Snowflake_schema for more details on the snowflake data schema.

is used to create dashboards. When checking/unchecking boxes on the left-upper panel or when double-clicking on parts of the chart, these operations are triggered and new charts are being rendered accordingly.

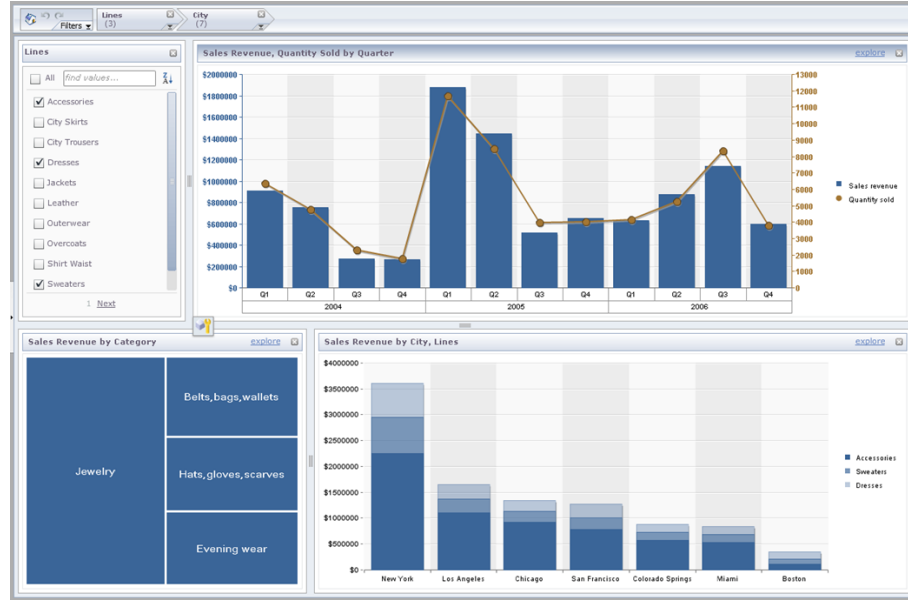


Figure 1.4 – State-of-the-art BI tools for exploring data

Role of hierarchies

Two objects (measures or dimensions) are functionally *dependant* if one determines the other (see also section 5.1 page 97). For instance, knowing an instance of the dimension ‘City’ determines the related instance of the dimension ‘State’. Another example that involves a measure and a dimension is that the ‘Sales revenue’ is determined by a ‘Customer’ (e.g. aggregated from unit sales in a fact table). Functional dependencies are transitive: if A determines B which determines C , then A determines C . In the most simple scenario, all measures are determined by all dimensions. This is the case when using a basic dataset, for instance reduced to one fact table with dimensions in a star schema. Functional dependencies are important to compose meaningful queries. For instance, they can be used to ensure that suggested queries do not contain incompatible objects which would prevent their execution. However, business domain models do not necessarily capture and expose this information. Hierarchies of dimensions are more common though, usually exploited in reporting and analysis tools to enable the drill-down operation. For instance, if a $\text{Year} \rightarrow \text{Quarter}$ hierarchy is defined, the result of a user drilling down on “Year 2012” is a more fine-grained query with the “Quarter” dimension, filtered on “Year”’s value 2012. If hierarchies of dimensions can be used to determine minimal dependency chains, techniques are required to help with automatic detection of functional dependencies. In particular, the approach

presented by [55] is to create domain ontologies from conceptual schemas and use inference capabilities.

1.2 BI and the need for relevant answers

The BI field aims at increasing the efficiency of decision-making activities in providing key figures of very large data that are generated and processed for instance by Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems. To ease this task of seeking for relevant data, tools have been designed (see screenshot figure 1.4). Such tools can be used by non database experts. However, the construction of a valid query that meets user's need is still a pain. Indeed, such tools are not *natural* in the sense that users need to know how data are conceptually organized in the data model, in order to formulate meaningful queries.

In the following, we introduce the BI domain and the increasing importance of mobility in business workers' tasks. Then, we detail specific aspects of corporate settings.

1.2.1 Introduction to Business Intelligence

BI is defined as the ability for an organization to take all capabilities and convert them into knowledge, ultimately, getting the right piece of information to the right person at the right time via the right channel. During the last two decades, numerous tools have been designed to make available a huge amount of corporate data for non expert users (see Figure 1.4). These tools vulgarize notions belonging to the multidimensional world like *data schema*, *dimensions*, *measures*, *members*, *hierarchy levels* etc. These concepts are usually not named in those tools, but their existence is assumed. In the above-referenced figure, none of these concepts appears. The upper-ribbon is composed of *filters* (i.e. selected members); the left panel lets users select dimensions. Measures have been selected in a previous step. User experience studies have been carried on [5] to determine where to position each component, such that its semantics is better understood by users (and seems natural to them). Figure 1.5 is an example of a graphic interface that lets users select dimensions (e.g. 'Year'), measures (e.g. 'Sales revenue'), predefined filters (e.g. 'This year') and execute the query in a subsequent step.

1.2.2 Mobility in BI and the growing popularity of natural search interfaces

In recent years, mobile devices have significantly invaded business users' everyday life. Indeed, capabilities of such devices have increased and wireless internet connections have developed as well. As a result, mobile devices are getting more and more popular and are extensively used in corporate settings, especially by employees who travel a lot.

The major advance in this field is probably the speech-to-text feature. Indeed, voice recognition is a technique that has improved significantly (at the condition that the user is a native speaker). Users can thus speak out loud instead of interacting with computers through traditional devices (mouse and

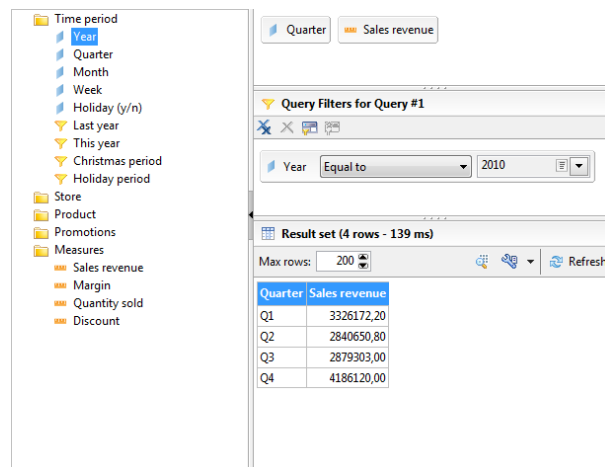


Figure 1.5 – Graphic interface to build multidimensional queries

keyboard). This eases access to corporate data sources, but a range of requirements specific to business use-cases still needs to be checked. We briefly describe those requirements in the following subsection.

1.2.3 BI and corporate settings

In corporate settings, users are well identified on the network and use different applications with possibly different authentication methods. The different systems can be for instance CRM systems or any Content Management System (CMS).

These various systems interact in some cases to offer aggregated information to end-users. However, each system usually has its own security management rules for users and/or groups of users. Therefore, allowing these applications to interact is a tricky issue.

Security requirements

As opposed to traditional answering systems (e.g. retrieving data from the World Wide Web (Web)), corporate systems must comply to security constraints. Some of these constraints are for instance:

- some users *cannot* access some resources, but they are *allowed* to get access to them
- some users *can* access some resources, but they are *not allowed* to get access to them
- some users *can* grant access to some resources they host to *some* users or groups of users
- the access to some resources to users or groups of users is granted at a specific time, for a limited (or unlimited) duration, but the terms can be changed at any time by an authorized user (or by a user belonging to an authorized group of users)

A direct implication of these constraints is that a resource that was granted to some user at a specific time, which authorization has been revoked later on **must not** be accessible to the user (e.g. as a search result) afterward. These aspects will be further detailed and applied to our case in chapter 3 which describes the implementation of the proposal.

Context-aware applications

Researchers agree that aggregating information from several data sources – not only corporate ones in the best case – promises more accurate results in the context of IR (see [33]). To allow interaction between these data sources, one needs to enter into an agreement on what can be shared across applications (this involves security considerations as sketched out above) and on how the environment is being modeled. Business workers share a consensual view on main concepts used in their daily work, which eases this process.

Context is a very active research domain, and thus it has been defined many times. The definition that is usually kept because of its generality is the one from Dey [14]:

“A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.”

Taking into account context in corporate settings makes sense, first because users are well identified on the network (for instance through single-sign-on technology). This “context-awareness” can be used in an efficient way in order to provide personalized items of interest (e.g. search results, answers, etc.). To illustrate this, let us consider a user who builds a query on a spreadsheet application and expects a chart as a response (see figure 1.6). On this figure, the generated chart must be of convenient size (i.e. not too large, such that it fits well in the spreadsheet). On the other hand, charts displayed on dashboards (see figure 1.4) or on desktop-specific applications (see figure 1.7) should be of bigger size.

1.3 Q&A and its benefits for BI

According to Hirschman [34], the goal of Q&A is to allow users to ask questions in NL using their own terminology and receive back a concise answer. Q&A is often seen as a subfield within IR (see for instance [27]). The purpose of IR [45] is to retrieve meaningful, or at least relevant information from a large repository of documents basing on a formulation of an information need. This information need is usually expressed by the means of keywords. The most popular examples of IR systems are search engines on the Web. While IR systems retrieve documents relevant for a specific query, Q&A systems aim at answering a question as concisely as possible [34]. The main benefit of Q&A systems is that users do not have to get through the list of documents to find appropriate answers; the most likely answer is returned by Q&A systems. The most advertised advantage of Q&A systems is that users waste less time in searching for answers.

We have seen in section 1.2 that existing tools interfacing data warehouses support users when designing their queries. However, there is still a gap to

bridge which can be compared to the Q&A gap between query space and document space. Concerning BI, users are supposed to be experts of the domain (i.e. they know the key indicators, or dimensions, used to describe their data), but they do not know the *exact* terminology that has been used to conceptually model the data schema. Users may use synonyms, or variant terms, or phrases formulated differently. The document space can be compared to the database itself, where a (structured) document would be a structured query (which is the requested information in our case). The growth and the “triumph” [31] of the Web in the 90’s have focused researchers’ attention on IR: indeed, the Web can be considered as an *infinite* corpus. One of its major drawbacks is that in the field of Q&A systems, it’s not possible to tell whether the answer of a question is known or not, while it usually is in corpus-based systems. Data warehouses in productive environments are modeled with hundreds of measures and possibly as many dimensions. As a result, the number of potential queries (i.e. the combinations of these objects) is huge, and similar in that sense to the problem of answering questions from large corpora of text.

Recently, researchers have emphasized the benefits and possible interactions of Q&A systems for BI systems. Ferrández and Peral [15] have proposed a system where the data schema of the data warehouse and the ontology schema of a Q&A system can be merged. In that way, the system is able to generate queries to the data warehouse and to the Web. More generally, the topic of combining data coming from various sources in corporate settings is of utmost interest and has been named “enterprise search” by Hawking [31] ten years ago. The different data sources can be trusted data sources (e.g. data warehouses), corporate information systems (e.g. intranet Web pages) or external sources

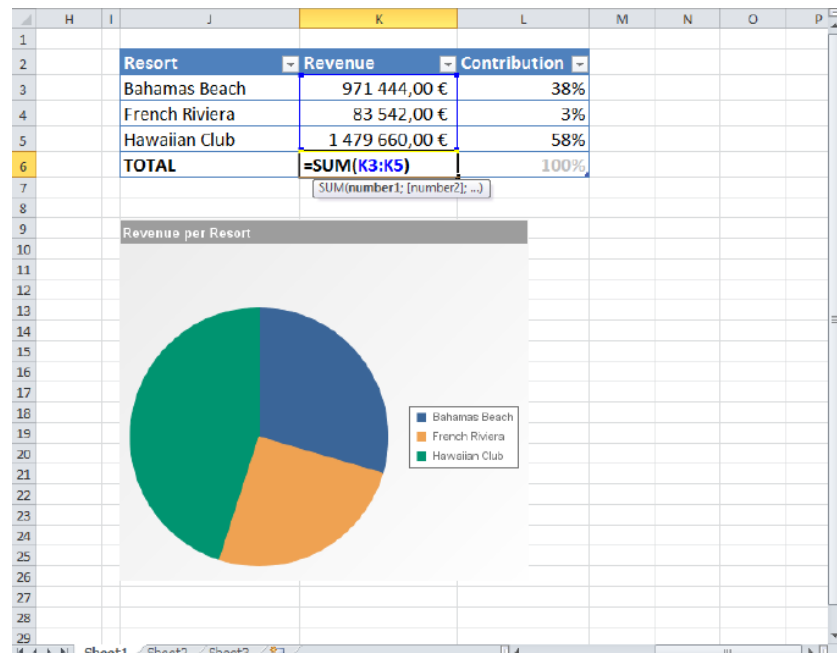


Figure 1.6 – Charts generated in an Excel instance must be of relatively small size compared to other applications like a dashboard

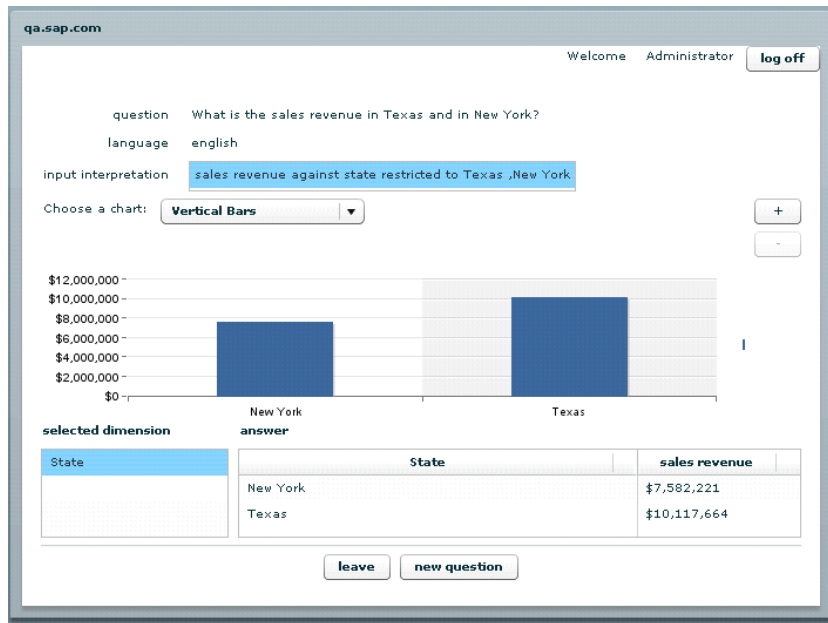


Figure 1.7 – Chart rendered on a desktop application

(e.g. Web pages from competitors)

1.3.1 WATSON, a success story

WATSON [17] has probably been the system that has made the strongest impression in Artificial Intelligence (AI) in 2010. Indeed, it is the first artificial program that has won the american *Jeopardy!* TV show. In this game, *clues* are presented to players (instead of questions), and players must provide answers in the form of *questions* (i.e. the question corresponding to the clues). For instance, instead of answering “Ulysses S. Grant” or “The Tempest”, answers must be “Who is Ulysses S. Grant?” or “What is the Tempest?” [17]. The underlying architecture (called DEEPQA [16]) is generic enough, such that it can be applied not only to *Jeopardy!*, but also to text retrieval (for instance compete to the TREC campaigns), enterprise search, etc. The system is composed of several statistic models that generate hypothesis and evidences. The synthesis of all potential hypotheses and evidences produces answers that are in turn merged and ranked. A part of the process consists in acquiring knowledge from various sources, which is partly a manual task.

1.3.2 Challenges in the field of structured data search

Research in the field of search over structured data consists mainly of keyword search over structured data. As introduced above, natural interactions with structured data becomes now popular in the community. Personalization meets a similar goal to recommender systems, where a personalized query can be seen as as a suggested query from recommender systems point of view. Besides, extensive work focused on *prediction* systems. Indeed, analysis of sessions

of multidimensional queries shows that queries are often refined in an interactive manner. To ease query formulation taking into account this observation, predictive models have been proposed, that predict queries that are more likely to be next users' queries (and pro-actively execute them).

A great challenge in this field is thus to propose comprehensive frameworks that tackle the problems of *personalization*, *query prediction* and *recommendation* in the context of multidimensional analysis (and not only one of them). As an example, personalization in the work of Golfarelli et al. [24] is tackled with users' *preferences* (at least for qualitative preferences).

1.4 Problem formulation

We consider a user's query q called *question* in the following (to make the distinction with structured queries simply called *queries*). The problem of translating a user's query expressed in natural language in a structured (database) query can be formalized as finding mappings t from a question q and a family of results $(r_i)_{i \in I}$:

$$t: \begin{cases} \mathcal{Q} \rightarrow R^I \\ q \mapsto (r_i)_{i \in I} \end{cases} \quad (1.1)$$

where $I = [1, n]$ is the interval of ranks associated to each result (i.e. n is the number of results that the question leads to). The index i of items r_i of the family is interpreted in that case as the *rank* of the result r_i (i.e. $\text{rank}(r_i) = i$). The rank i assigned to a result r is computed by a *scoring* function that we note $\text{score}(r) \in [0, 1]$:

$$\text{rank}: \begin{cases} R \rightarrow I \\ r \mapsto \phi(\text{score}(r)) \end{cases} \quad (1.2)$$

where ϕ is a bijection from $[0, 1] \subset \mathbb{R}$ to $I \subset \mathbb{N}^*$. In the context of multidimensional queries, we consider that a *result* is a multidimensional query Q and a set of metadata (which can be the chart title or application-specific requirements in terms of visualization like the color panel, the expected size of the frame containing the chart, etc.). We note: $r = (Q, M) \in R$; in the following Q is the notation for a structured query and M the one for metadata.

1.4.1 Machine translation and pivot languages

The problem introduced above (formula 1.2) is similar to the problem of translating natural language sentence from a (natural) language to another. The difference is that we target here a structured (artificial) language (i.e. a database language) instead of a natural language. This problem is one of the most difficult one in AI and one of the most popular (at least until the 90's); it has thus a long history in computer science. A popular representation of two classic approaches is displayed figure 1.8. Figure 1.8a is the class of approaches that use a "pivot-language" to translate a source language to a target language. The difficulty of this approach, and its main limitation, is that the different sub-task when going from the first language to the pivot language generate noise, and more noise is introduced when translating this pivot language to the second language. The second class is represented figure 1.8b: no pivot language is

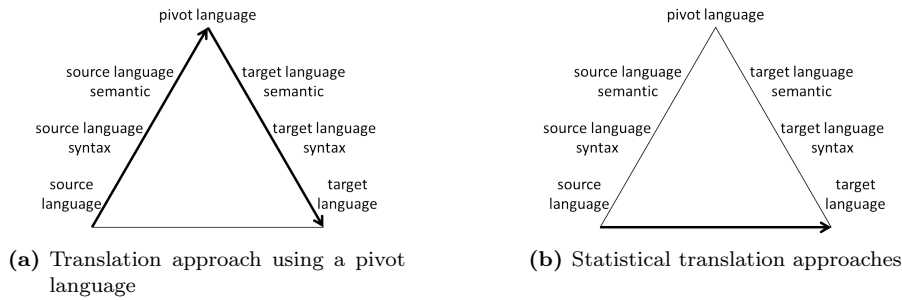


Figure 1.8 – Two classic translation approaches

required, but the translation usually relies on statistical models, that require costly resources.

1.4.2 What is not of the scope of our work

In our work, following interesting topics have not been investigated:

- searching over the Web and aggregating results from the Web and results from trusted sources
- generating results that cannot be rendered as charts

Indeed, we have focused on translating questions in structured queries for data warehouses (see section 1.4.1). The framework described in chapter ?? is extensible and also supports other kinds of data sources. The application-driven motivation of our work explains why we consider a subset of possible query results.

1.5 Contribution to the state of the art

Our contributions to the state of the art are as follows:

1. a comprehensive framework for Q&A dedicated to business users which leverages contextual information to offer more personalized results
2. a NL query interface associated to a speech-to-text tool
3. a translation approach
 - that has been proven to be valid in at least 3 european languages
 - which graph-matching bases on constraints satisfaction rules
4. a plugin-based architecture which ensures a high degree of portability
5. an overall approach which makes the system quite independant with respect to the domain according to our evaluation results

1.6 Organisation of the thesis

This thesis is organized as follows:

1. chapter 2 page 15 surveys the related work of Q&A systems for structured data for questions formulated in NL
2. chapter 3 page 57 presents the global architecture of the system and provides implementation details of the end-to-end approach
3. chapter 4 page 75 provides details on the core technique used to translate NL questions in database queries
4. chapter 5 page 97 defines the adopted conceptual query modeling and provides research directions on personalization based on this model
5. chapter 6 page 119 reviews our experimentation results and compares the performances of our system to some standards of the field
6. chapter 7 page 135 concludes the thesis and suggests follow-up research based on the proposal

As additional material, the interested reader will find further information in the following appendix sections:

1. appendix A page 139 provides a (long?) example of pattern which is part of the system
2. appendix B page 143 is an example of query logs introduced in chapter 5
3. appendix C page 145 discusses the conceptual query modeling and its translation in actual structured queries
4. appendix D page 147 provides screenshots of the front-ends
5. appendix E page 151 reproduces the source code of some classes and illustrates how parallelism has been achieved in the implementation

Chapter 2

State of the Art

Outline

2.1	Main Dimensions	17
2.1.1	Data sources	18
2.1.2	Users' questions	18
2.1.3	Internal representation	20
2.1.4	Database queries	21
2.1.5	Domain-independence	23
2.1.6	Portability	23
2.1.7	Metrics	24
2.2	Anatomy of NL interfaces systems	25
2.2.1	Lexicon	25
2.2.2	Semantic rules	25
2.2.3	Main problems to solve	26
2.3	Taxonomy of main approaches	26
2.3.1	Domain-dependent semantic parsing	26
2.3.2	Complex question translation	28
2.3.3	Feedback-driven approaches	28
2.3.4	Learning-based approaches	29
2.3.5	Schema-unaware approaches	30
2.4	Domain-dependent semantic parsing	31
2.4.1	BASEBALL [25]	31
2.4.2	LUNAR [72]	33
2.5	Complex question translation	34
2.5.1	CHAT-80 [69]	34
2.5.2	QWERTY [49]	36
2.5.3	IRUS [6]	37
2.5.4	PRECISE [54, 53]	38
2.5.5	PANTO [68]	39
2.6	Feedback-driven approaches	40
2.6.1	TEAM [28]	41
2.6.2	MASQUE/SQL [2]	43
2.6.3	NALIX [42] and DaNaLIX [41]	44
2.6.4	C-PHRASE [47]	45

2.6.5	ORAKEL [12]	46
2.7	Learning-based approaches	47
2.7.1	Miller et al. [46]	47
2.7.2	Zettlemoyer et Collins [76]	48
2.7.3	WOLFIE [66]	49
2.8	Schema-unaware approaches	50
2.8.1	POWERAQUA [44]	50
2.8.2	DEEPQA [16]	52
2.9	Challenges	54
2.10	Conclusion	54

NL is the most natural and easy way to express information need [33], because it is not restricted to expert users who know how to formulate formal queries (e.g. Structured Query Language (SQL) or MultiDimensional eXpression (MDX) queries).

NL interfaces have been investigated by researchers in both Information Retrieval (IR) and Database communities. In the IR community however, the keyword search has become popular because of the success of commercial search engines which have made extensive use of algorithms for matching keywords to documents. Interfaces to unstructured documents have become popular as the Web has made available huge amount of textual data, and recently as the famous *Jeopardy!* quiz was won by IBM's WATSON [17] system. This trend seems to evolve slightly. For instance, WolframAlpha™ is a popular search service¹ based on structured data, that accepts keyword queries as well as some questions in NL. However, the proportion of NL questions that are understood by WolframAlpha™ is still low, and therefore this system is still subject to improvements.

Interfaces to structured data have focused much researchers' attention for decades, but the field seems to have known a renewed interest for a few years, probably thanks to the development of the Semantic Web (SW). Today's semantic technologies cannot be easily manipulated by standard users, and therefore there is a need for interfaces. Indeed, users prefer NL interfaces than the logic which bases the SW. Unger et al. [67] have illustrated this problem in the context of the SW, where data are usually represented as Resource Description Framework (RDF) *triples*. Consider for instance the following question [67]:

“Who wrote The Neverending Story?” (2.1)

In the best case, triple data answering this question would be of the form:

< [person,organization], wrote, ‘Neverending Story’ > (2.2)

where [person,organization] would be a placeholder for a subject representing a person or an organization in the data. Depending on the adopted approach, this *triple* would be either retrieved based on distance metrics with respect to the question (2.1), or the result of the execution of a formal query (in this case a SPARQL Protocol and RDF Query Language (SPARQL) query) which would look like:

¹<http://www.wolframalpha.com/>

```

1 SELECT ?x WHERE { ?x ?p ?y .
2             ?y \ac{RDF}:type ?c .
3 }
4 HAVING expr_1
5 ORDER BY expr_2
6 LIMIT expr_3
7 OFFSET expr_4

```

Listing 2.1 – Example of SparQL query template operating on top of RDF data

where `expr_1`, `expr_2`, `expr_3` and `expr_4` are placeholders for various SPARQL expressions. In this example, not-expert users would not write *triples* (especially because some NL questions cannot be translated in this representation, e.g. when there are aggregations or filter constructs which are not faithfully captured [67] by SPARQL). Nor would they write a formal SPARQL query like the one shown in 2.1, because the syntax is not straightforward to standard users.

The area of NL interfaces to structured data is even broader than IR, in particular there is a range of systems that translate NL queries in *goals*, e.g. in the context of (household) appliances (see for instance the EXACT system [75]). These kind of systems are out of the scope of our thesis, and therefore they will not be described in this chapter.

In our work, we outline the major dimensions of existing systems and present the new trends and challenges of state-of-the-art systems. The history of NL interfaces can be developed as follows:

1. early years of domain-specific systems
2. complex question answering in a specific domain
3. rise of domain-awareness in NL interfaces (through learning techniques)
4. data-driven systems (or schema-unaware approaches)

We present in the following the main dimensions used throughout the paper.

2.1 Main Dimensions

The input to every NL interfaces is the (1) *data*, and (2) *user questions* representing information needs, which are translated to an (3) *internal representation* of the needs employed by the system and/or directly mapped to (4) *database queries* that finally, are executed by the underlying query engine to produce the final answers. The main problem tackled by a NL interfaces approach is to transform the question to the internal representation and then to the database query – i.e. (2) \mapsto (3) \mapsto (4) – or to directly mapped the question to the query – i.e. (2) \mapsto (4). This problem is hard when considering the large and rapidly evolving mass of structured data that have become available. The major challenges modern NL interfaces are facing in this regard is to operate across domains (5) *domain-independence* as well as to adopt to new domains (6) *portability*. For evaluating the quality of the NL interfaces output, different (7) *metrics* based on the user questions the system can understand as well as the database queries it can produce, have been proposed.

2.1.1 Data sources

Data are organized in *structures* of various kind. In the case of unstructured documents (e.g. raw textual documents), the structure is defined by the corpus that contains the documents and the metadata possibly attached to the documents (e.g. the title or the authors' names). In semi-structured documents (e.g. Web pages) the structure explicitly surrounds the content of the documents (e.g. categories in Wikipedia). In structured documents (databases, knowledge bases, etc.) the nature of the structure depends on the logic adopted when these documents have been created. In any case, data structures can be reduced to a set of *entities* and *relations* between those entities of possibly various kind.

An early data structure is called *hierarchical list* and used in the BASEBALL system [26]. We reproduce an example of such a list below:

```

Month = July
Place = Boston
Day = 7
Game serial No. = 96
  (Team = Red Sox, Score = 5)
  (Team = Yankees, Score = 3)

```

This list could also be represented as a set of *facts*. 'Month', 'Place', 'Team' and 'Score' are *entities*, 'Game serial No.' is an attribute, and there are two kinds of relations: hierarchical relation (depicted with white spaces in the list representation) and a standard relation (depicted with parentheses).

We classify the different data structures as follows:

- early data structures (Prolog databases; hierarchical lists)
- relational databases
- XML databases (which is an example of hierarchical database)
- ontologies; *linked data*

The way to access (i.e. query) and modify data depends on the data structure. For example, relational databases are logically represented with the relational model (see section 1.1.1 on page 3) and queries are usually expressed in a specific language, like SQL (or MDX). Hierarchical lists are hierarchical structures (like XML documents) that are composed of embedded key-value pairs with optional attributes. XML documents are more generic, since keys in those documents are nodes (i.e. can be trees themselves). Ontologies are not more generic than XML databases from the structure point of view, but ease the expression of semantic relationships between entities (namely *concepts* and *individuals*).

2.1.2 Users' questions

Traditional DBMSs provide an interface where users can query the data in various ways, usually in a formal query language such as SQL. We present in the following two kinds of input to database interfaces, namely *keyword queries* and *NL queries*.

Keyword query

A *keyword query* consists in a set of words traditionally used in the context of document search. Traditionally, documents are represented in a vector space, where the vectors are composed of the terms of the documents. Search engines have made this paradigm popular. Hearst [33] reports that the number of words in queries over Web search engines trends to increase (the experiment compared queries performed over a month in 2009 and in 2010): queries from 5 to 8 words increase up to 10% while queries from 1 to 4 words decrease up to 2%. It seems indeed that users are more and more aware of new capabilities of state-of-the-art search engines, and are less reluctant to express their information need in a more “natural” way. Therefore we focus further on natural inputs to interfaces in the following section.

Natural language query

In this state of the art, we focus on natural language interfaces, and not on keyword search over structured data, even if the latter has become quite popular lately. A significant recent system with this respect is SODA [8].

Early systems do not understand well-formed sentences (usually in English), but some basic syntactic constructions; such language was called “English-like” by some authors [72]. More recent systems try to go further and to capture as much as possible regularities as well as some irregularities of NL.

Most of current systems handle only a subset of NL questions. This subset of NL is called *controlled NL*. Some systems even go further, and are able to analyze why a question can not be answered. There are basically two cases:

- the question is beyond the system’s linguistic coverage (i.e. the system does not understand the question)
- the underlying knowledge base (e.g. database) does not contain any fact answering the question (i.e. the semantics of the question is fully or partly captured but there is no answer to such a question)

NL questions are classified based on their *type*:

- *factoid questions* (i.e. questions that can be answered by an *objective* fact, usually questions starting with *wh*-words except ‘why’)
- *complex questions* which answers are usually *subjective* to the answerer, and for which there might be several correct answers, possibly contradicting each other

Complex questions can be ‘why’-questions, ‘how’-questions or *definition*-questions. Factoid questions have been defined by Soricut & Brill [61] as questions “for which a complete answer can be given in 50 bytes or less, which is roughly a few words” (see also Kwok et al. “Scaling question answering to the Web, WWW). Several finer classification have been proposed in specific domains or applications. For instance, in the case of temporal questions, Saquete et al. [57] have proposed the classification reproduced table 2.1 on the following page.

Question type	Example
Single event temporal questions without temporal expressions	“When did Jordan close the port of Aqaba to Kuwait?”
Single event temporal questions with temporal expressions	“Who won the 1988 New Hampshire republican primary?”
Multiple events temporal questions with temporal expressions	“What did G. Bush do after the U.N. Security Council ordered a global embargo on trade with Irak in August 90?”
Multiple events temporal questions without temporal expressions	“What happened to world oil prices after the Iraqi annexation of Kuwait?”

Table 2.1 – Question classification for temporal databases from [57]

Error feedback

NL interfaces output database queries. Those queries must be then be executed by underlying DBMS. The execution of queries can lead to different failing states:

- the query execution fails (the generated database query is not valid)
- the query execution lead to an empty result set

In both cases, the system may inform the user and/or suggest or rephrase the query. This is a task performed by systems belonging to the range of feedback-based approaches (see section 2.6 on page 40).

2.1.3 Internal representation

The syntactic representation of a question (also called *parse tree* because the tree representation is usually adopted) is an intermediate representation, before the internal representation of the question (i.e. the semantic representation) is being created. In many systems however, the syntactic representation also contain pieces of semantic information. For instance, nodes of the syntactic tree contain information about how to generate fragments of the target database query. The nodes of the tree representation contain information about words and relations between words of the question. Typical semantic information contained in a syntactic parse tree are the database elements that those words refer to. Those semantic information (also referred to as *meaning* in early systems) are kept in a lexicon. The syntactic parse tree usually does not try to resolve ambiguities, and keep all possible interpretations. Resolution of ambiguities is done afterward, when the semantic representation is being built out of the syntactic representation. Figure 2.1 on the facing page is an example of parse tree of the question “Who produced the most films?” processed in the system described in [67]. In the figure, nodes hold syntactic information (e.g. ‘WP’ stands for *wh*-pronoun). The result of the parse tree depends on the chosen parser.

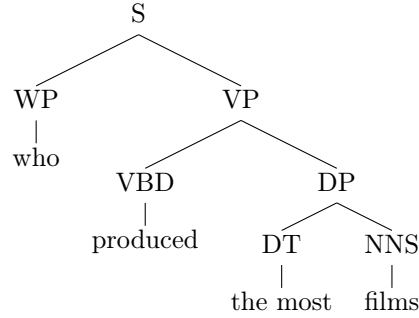


Figure 2.1 – Example of parse tree for the question “Who produced the most films?” from [67].

System	Internal representation	DB query
BASEBALL [25]	specification list	✓
LUNAR [72]	meaning representation language	X
CHAT-80 [69]	logic expression	X
TEAM [28]	logic expression	X
QWERTY [49]	logic expression	X
IRUS [6]	meaning representation language	X
PRECISE [54]	graph representation	X
MASQUE/SQL [2]	meaning representation language	X
NALIX [42, 41]	XQuery	✓
DANALIX		
C-PHRASE [47]	λ -calculus	X
PANTO [68]	graph representation	X
ORAKEL [12]	λ -calculus	X
Miller et al. [46]	semantic frames	X
Zettlemoyer et Collins [76]	λ -calculus	✓
WOLFIE [66]	Prolog	✓
POWERAQUA [44]	graph representation	X
DEEPPQA [17]	semantic triples	✓

Table 2.2 – Semantic meaning representations

2.1.4 Database queries

The parse tree (which may or not contain semantic information in the nodes of the parse tree) must then be *interpreted* in some internal semantic representation. Table 2.2 displays semantic meaning representations used in various systems. As shown in the table, the internal semantic representation can be or not the target query representation. The semantic representation is intended to capture as much as possible the user’s intent, and is sufficient to generate the final database query. While the syntactic representation may contain lots of ambiguities, the semantic representation does not. The adopted semantic representation depends on the data structure, and thus the target DBMS. However, some systems contain components that translate the semantic representation in the different database query languages, for instance when there are several

underlying systems with different query languages. This kind of architecture participate to make the system more *domain-independant* (see section 2.1.5 on the facing page for more details).

Query languages

Relational databases are generally associated with the SQL query language (and temporal databases are associated to an extension of SQL called temporal SQL). Multidimensional databases (e.g. data warehouses) are associated with MDX, which is a SQL-like query languages dedicated for handling measures, dimensions and hierarchies (key concepts of multidimensional models). Other data structures are associated with their own query language. For instance, hierarchical lists (or specification lists) is not associated with a language, but to a similar structure, which is a template where empty slots correspond to the expected items of the resultset. XML databases are associated to XPath, and semantic databases (e.g. RDF) are usually queried with languages derived from SPARQL which looks similar to SQL.

Recently, a new range of query languages have appeared and are associated to a new class of DBMSs, namely NoSQL (for ‘not only SQL’). These DBMSs are optimized for special data models and large data sets beyond the scope of this survey.

In the following we review the main primitives that appear in query languages: selection, constraints and query modifiers.

Selection a selection consists in choosing the expected database attribute to appear in the resultset. In SQL, the selection is expressed as `SELECT t.x`, where t is a table and x a field or attribute belonging to t . In SPARQL it corresponds to *variables* that are defined in the query, and that must satisfy the constraints defined in the `WHERE` section of the SPARQL query (see section 2.1.4). In MDX, the selection corresponds to the ordered set of dimensions or measures that should appear in the resultset, with the information about the level of the expected attributes in the hierarchy, and the filters. For instance, `SELECT Country.[All members]` corresponds to the selection of the dimension ‘Country’ with no filter (i.e. selection of all members of the dimension ‘Country’). Selection for a hierarchical list corresponds to a slot in the structure template. A selection in a XPath query can be expressed: `nodename` (selects all nodes with the name *nodename*), `/nodename` (selects the root node *nodename*), `//nodename` (selects all nodes *nodename* no matter where they occur), `@attr` (selects attribute *attr*), etc.². In SPARQL, a selection can be expressed also differently. Indeed, the output of a SPARQL query can be a data structure of the same kind than the data themselves (i.e. a graph). This is performed with the `CONSTRUCT` keyword.

Constraints Constraints in database queries aim at reducing the size of the resultset. In the case of SQL, SPARQL and MDX, such constraints are introduced with the `WHERE` keyword. In SQL, these constraints can define how different tables can be *joined* together such that attributes belonging to different tables can appear in a single view (i.e. in the resultset). Other kinds of

²for more details, see http://www.w3schools.com/xpath/xpath_syntax.asp

constraints are about the values of attributes. In SPARQL, the constraints are expressed with *triples* (which can be made of some *blank node* or undefined node). Thus, a SPARQL constraint is about relations between entities (see *joins* for relational databases), or about entities themselves (which type must be the entity? If this is a *literal*, in what range of value must it belong?, etc.).

Query modifiers Conceptually, modifiers are primitives that modify the resultset afterwards. Such primitives are:

- order the resultset along a given attribute (e.g. `ORDER BY` in SQL)
- select at most n *tuples* of the resultset (e.g. `LIMIT x` in SQL)
- select only different tuples (e.g. the keyword `DISTINCT` in SQL)
- etc.

In SPARQL, it is possible to test if a set of constraints can be satisfied from the data, and to combine these tests with optional constraints (`OPTIONAL` keyword) or mandatory constraints (`FILTER` keyword).

2.1.5 Domain-independence

Domain-independence is the ability of a system, to operate accross different domains simultaneously. In practice, this means that a domain-independant system would be on top of different data sources that belong to different domains, and that this system is able to meet users' requests for any of these domains. The main difficulty of building domain-independant systems is the ability to translate NL constructions differently in the different application domains. For instance, the qualitative expression "middle-aged" can be interpreted completely differently accross the different domains (see how it is interpreted in our Q&A system in section 4.2.1 on page 77).

2.1.6 Portability

A system is said *portable* if it can be easily configured for different domains (and not the first domain for which it was first designed). Domain-independence (section 2.1.5) is a step further, because different domains are considered simultaneously; while in portable systems the challenge consists in easing the effort of the system administrator when configuring the system. Portable systems are also referred to as *configurable* interfaces in the following. Configurable interfaces let users improve the system's capabilities, both the linguistic coverage S and the system coverage K (see Fig. 2.2 on the following page).

Minock et al. [47] have identified three kinds of configuration applicable to NL interfaces:

- let users name database elements, so that phrases used in the question can be easily mathed with database elements
- offer a General User Interface (GUI) that automatically generates semantic rules or a grammar for translating NL questions in database queries

- use machine learning techniques to induce semantic rules or a grammar from annotated corpora

These ways of configuring interfaces are not of equal cost: for instance, the third one (machine learning techniques) can be highly costly if it requires a huge volume of annotated data. The cheapest configuration is based on user interaction, where no initial configuration is needed, but domain-specific knowledge is learned based on user interaction. An example of such a system is NALIX [42].

2.1.7 Metrics

Several evaluation metrics have been introduced in various systems. We review them briefly below. We present the interesting figure 2.2 copied from Han et al. work [29]. It shows the tradeoff between linguistic coverage (S in the figure) and the expressions that can be answered from a knowledge base (K in the figure). The goal of any interface would be that the linguistic coverage comprises all expressions that can be answered from the knowledge base.



Figure 2.2 – Linguistic coverage vs. logical coverage within interfaces, copied from [29]. S represents the range of expressions that are understood by the system and K the range of expressions that can be answered from the knowledge base.

Fluency_{Woods}

Woods [72] defines *fluency* as “the degree to which virtually any way of expressing a given request is acceptable”. Fluency measures roughly how easy to use is a system. Intuitively, a good fluency means a “natural” interface in the sense of natural interfaces [33]. This requires advanced natural language techniques, and would probably lead to systems that can interpret more expressions than those that can be actually answered by the knowledge base ($S \setminus K$ in figure 2.2 would not be negligible).

Completeness_{Woods}

Completeness was first defined in Woods’ work related to the Lunar [72] system. It measures if there is a way of expressing any query which is logically possible from the database. In the end, it measures if the interface can answer all possible questions. In figure 2.2, completeness would be represented by S comprising K ($K \subset S$).

Soundness_{Popescu}

An interface is said *sound* if “any SQL output is a valid interpretation of the input English sentence” [75]. In figure 2.2, this evaluates the expressions of $V = S \cap K$.

Completeness_{Popescu}

An interface is said *complete* if it “returns all valid interpretations of input sentences” [75]. Yates et al. have reused this metrics in their EXACT system (not surveyed here). They do not claim that users should restrict their questions to query only tractable questions; but they suggest that identifying classes of questions that are semantically tractable and measuring the prevalence of these questions is the direction of current research.

User’s intent

Yates [75] makes an assumption, that if the interface is sound, complete and if a single SQL statement can be produced from a NL question, then the interface has unambiguously determined user’s intent.

Predictability

Within user interfaces, predictability has been pointed out as the essential feature of user interfaces in the 90’ by Norman [50] and Schneiderman et Maes [58]. Predictability and the feel of control seems however to be in contradiction with personalization, which is the one pillar of recent IR systems.

2.2 Anatomy of NL interfaces systems

The main problem to solve is to map user’s intent expressed in a natural way (say unstructured way) to a database query, which is a structured expression where there is no room for ambiguity unlike natural language. The problem that NL interfaces systems try to solve is equivalent to finding a mapping f between natural language questions q and families of structured queries $(q'_i)_i$ (where the index corresponds to the *rank* of the structured query):

$$f : \begin{cases} Q \rightarrow (Q')^I \\ q \mapsto (q'_i)_i \end{cases}$$

where $i \in I = [0, n]$ is the index of q'_i .

2.2.1 Lexicon

The lexicon is a data structure that is used to reduce the ambiguity in words when analyzing users’ questions. NL interfaces are usually composed of two lexicons: a domain-dependant lexicon and a domain-independant lexicon. The domain-dependant lexicon defines words in terms of semantic rules (see section 2.2.2). The domain-independant lexicon defines how to interpret words independantly from a specific domain. For instance, *wh*-question words define constraints on the structure of the expected database query (e.g. in SQL).

2.2.2 Semantic rules

Semantic rules define the semantics of question words. The input is a node, or a part of the parse tree being constructed, and the output contains information on how to construct part of the final database query. The final database query

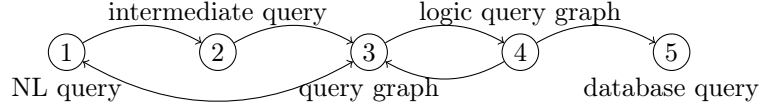


Figure 2.3 – Big picture of NL interfaces systems

will be generated from the parse tree, which contains semantic information in the nodes, in addition to lexical and syntactic information. The idea behind this process is a linguistic theory [missing ref], where the global meaning of a sentence is defined by the individual meanings of words / expressions in the sentence, and by the syntactic relationships between the words / phrases.

2.2.3 Main problems to solve

The main problems to solve are threefold. First, NL systems should have the broadest linguistic coverage (such that in the best case, users can employ their own terminology). Secondly, there should be some mechanisms that allow users to query other domains with a low porting cost. Thirdly, there should be some components responsible for checking that generating database queries are valid (in terms of syntax).

2.3 Taxonomy of main approaches

Figure 2.3 depicts the big picture of translating NL queries to database queries. In the figure, nodes represent various steps involved in the translation (from NL questions to structured queries) and edges the iterations performed by actual systems surveyed in this work. We consider two main ranges of approaches, namely *classic translation approach* and *iterative approach* introduced below, which are then further refined.

- Classic translation approach: The classic translation approach consists in going from ① to ⑤ through ②, ③ and ④ (step ② being optional).
To reach step ③, semantic rules are needed to find out what database elements should be associated to question phrases.
- Iterative approach: The iterative approach does not go directly from ① to ④, but goes back to ③ (and/or to ④) several times before reaching step ⑤. Those iterations correspond to question reformulations, that involve user-feedback to interpret semantically the question in terms of a database query.

The way translation from NL questions to formal query has evolved over years. Table 2.3 on the next page presents an overview of major systems that we take into consideration in this survey.

2.3.1 Domain-dependent semantic parsing

Early systems belong to this class of approaches. The amount of knowledge necessary to translate NL questions to database queries is encoded in a lexicon.

Taxonomy	Approach	System	Dimensions					
			Q	D	A	P	L	E
Classic translation approach	Domain-dependent semantic parsing	Baseball [25]						
		LUNAR [72]						
	Complex question translation	Chat-80 [69]				+	+	
		IRUS [6]				+		
		QWERTY [49]		temporal		-		
		Precise [53, 54]						+
Iterative approach	Feedback-driven approaches	PANTO [68]					+	+
		MASQUE/SQL [2]		Prolog/SQL		+		
		TEAM [28]				+		+
		NALIX & DANALIX [42]				+		
		C-PHRASE [47]						
		Orakel [12]				+		
	Learning-based approaches	Zettlemoyer et al. [76]						
		Miller et al. [46]						
		WOLFIE [66]						
	Schema-unaware approaches	PowerAqua [44]						
		DEEPA [?]						

Table 2.3 – Overview of major NL interfaces to structured data. ‘Q’ stands for ‘Query’, ‘D’ for ‘Data’, ‘A’ for ‘Answer’, ‘P’ for ‘Portability’, ‘L’ for ‘Linguistic coverage’ and ‘E’ for ‘Error feedback’

The systems allow users to employ a subset of English (i.e. a controlled language) to query databases. The lexicon, however, is a huge linguistic resource which defines how each word and/or phrases must be translated into database elements, and what semantic rules to trigger to get, in the end, the desired database query.

Lexicon

The lexicon is a resource that defines a set of words that belong to a domain. Those words are associated to a meaning, which is a semantic rule that controls how this word must be interpreted in the current data domain and for the data structure. In addition, the lexicon also contains a list of specific rules that modify the global meaning of the query being generated, based on some words that are already defined in the lexicon. The lexicon usually combines both syntactic with semantic pieces of information. For instance, the same word would have a different interpretation whether it is a noun or a verb in the sentence where it occurs.

Limitation

The domain dependence is however the great limitation of those systems; this is due to the cost of the lexicon. Indeed, porting such systems to other domain would mean provide a new lexicon and corresponding semantic rules, which is highly costly.

2.3.2 Complex question translation

The next generation of NL interfaces aims at increasing their linguistic coverage. This is performed with the distinction of both domain-dependent knowledge and domain-independent knowledge. The domain-dependent knowledge base consists in semantic rules triggered by words, phrases or syntactic information encoded in a parse tree. Those rules produce fragments of the target query language (or of the intermediate query language) to be then combined and modified to generate the final query language. The domain-independent knowledge base is composed of lexical information in a dictionary (which might be completed with domain-dependent knowledge, like the most likely senses of words used in the application domain).

2.3.3 Feedback-driven approaches

This range of systems translate NL questions to database queries basing on users' feedback. We distinguish between the following kinds of feedback:

Authoring tool configuration

Authoring tools are GUI that permits users to edit domain-specific knowledge (e.g. the lexicon that is used to translate NL questions to internal queries). Editing such domain-specific knowledge within these tools can be straightforward (like synonyms to be used in user's question to describe the same database elements) or more complex (like semantic rules to translate user's terms to logical elements). Describing this knowledge is not an easy task for standard users,

for instance semantic rules that define how to map words and phrases to logical elements, and rules that define how information from the parse tree must be combined to produce the final logical query. For that reason, recent advanced systems try to infer such rules basing on dialog-like interaction with the user.

Interactivity

Authoring tools introduced in the previous section are intended to be used as a preliminary task when porting the interface to other domains. Other systems do not explicitly ask users to answer questions to get the domain-specific knowledge, but infer this knowledge on the basis of interaction. This range of systems suggests NL questions to users that could be reformulations of the current user's question. To the best of our knowledge, there are two different kinds of such interaction, when the system cannot interpret a question:

- the system tries to change words and/or phrases in user's question, so that the system is able to interpret the question
- the system also comprises a repository of successfully answered questions, and suggests one of those questions replacing some slots with terms used in the user's question, and ensuring that the generated question can be interpreted by the system

Recent work [38] investigate how to present similar questions in NL as interpretations of the current question. This is a way of making the user think she controls what is happening, which is one of the expected features in modern interfaces. This paraphrasing feature is also present in NALIX and DaNaLIX [42, 41] and C-Phrase [47] works.

2.3.4 Learning-based approaches

The *learning-based approach* consists in learning a grammar or a set of rules that map NL sentences to logical forms. Learning-based approaches are popular among the NLP community. Indeed, learning techniques reduces the cost of linguistic resources, which is being learned over time. In the case of NL interfaces, this allows to port the system to other domains with a limited cost. Most systems need a corpus of labelled examples, e.g. a set of sentences mapped to their corresponding logical form.

However, such corpora are rarely available, and are costly to produce. Some systems adopt strategies to address this (see WOLFIE section 2.7.3 on page 49 for instance).

Statistical models

Table 2.4 on the next page summarizes the different statistical models used by learning-based systems. Miller et al. [46] use a probabilistic recursive transition network and consider the probability $P(T|W)$ of a parse tree T given a word string W :

$$P(T|W) = \frac{P(T) \times P(W|T)}{P(W)}$$

System	Model
Miller et al. [46]	recursive transition network
Zettlemoyer et Collins [76]	log-linear model

Table 2.4 – Different statistical models used by learning-based systems

System	Training data volume
Miller et al.	4000 sentences
Zettlemoyer et Collins [76]	600/500 training examples ³

Table 2.5 – Volume of training data of different systems

This model combines both *state transition probabilities* and *word transition probabilities*. The state transition probability concerns the labelling of a combination of semantic and syntactic information recursively (the label of a node is computed given the labels of the previous node in the syntactic order, and the parent node). The word transition probability is the probability of a word, given the previous syntactic word and a semantic information (attached to the parent node in the parse tree).

Zettlemoyer et Collins [76] use a log-linear model to learn a combinatory categorial grammar. This grammar, which also defines the domain-specific lexicon of the parser, contains semantic information (*i.e.* to which λ -calculus formula a given word and/or phrase should be associated).

Shortcomings of these approaches

These components rely on statistical models, that often require a large amount of annotated data. For instance, a statistical parser require a significant number of questions annotated with corresponding parse tree. Table 2.5 shows the volume of training data corresponding to two systems (Miller et al. and Zettlemoyer et Collins). Besides, training data should also be composed of negative examples, whose availability is a strong requirement as pointed out by Giordani et Moschitti [22].

2.3.5 Schema-unaware approaches

A range of recent systems aggregate potential answers from different sources. This range of systems have emerged in parallel with the success of the SW technologies. The particularity of these approaches is that they cannot rely on the schema of underlying knowledge bases because the different sources are potentially modelled in very different ways. Thus, the set of systems represented by these approaches try to bridge the gap between user terminologies and the different terminologies used in the different knowledge bases that the interfaces talk to. These approaches can be seen as an extension of several approaches presented before, namely ‘complex question translation’ (because the question is further analyzed to map it with the terminology used in the different knowledge bases) and ‘learning-based approaches’ (because some of the systems represented in these approaches contain learning components).

System	D	SK	S
BASEBALL	specification lists	lexicon semantic rules	cost of the lexicon
LUNAR	proprietary schema	lexicon semantic rules	cost of the lexicon simple data structures exact term matching

Table 2.6 – Systems belonging to the *domain-dependant parsing* range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “Shortcomings”.

Terminology mapping

In previous approaches, end-users were not aware of how data were modeled in the knowledge base. This requires advanced natural language processing to map user’s terminology with database terminology. In schema-unaware approaches, the systems do not talk to a single database, but to potentially an unlimited number of sources where the structured data are to be found. Each of those sources has its own logical schema, naming conventions, etc. Then, the system must know how to communicate with these knowledge bases, and what strategies to adopt to reduce the computation cost of generating distributed queries (see below). In some cases, it’s also critical to aggregate results from different sources.

Shortcoming of these approaches

The major shortcoming is related to scalability and efficiency. In particular, in cases where the knowledge bases are searched over Internet (for instance Linked Data⁴), the computational complexity of mapping and expanding user query terms to the terminology of respective knowledge bases is a limiting factor. Internet latency in this case is also to be considered since the databases are not hosted.

2.4 Domain-dependent semantic parsing

The major systems are BASEBALL [25] and LUNAR [72]. Both systems are surveyed below. Table 2.6 overviews these systems.

2.4.1 BASEBALL [25]

BASEBALL aims at answering questions about baseball results. The main concepts in the data are games, teams, scores, day, month, place (of a game). The domain is quite close, which means that there is few ambiguity in terms of word meaning.

Data structure

The data structure is a list structure called *specification list*. This is a hierarchical structure, where each level correspond to an attribute associated to a

⁴See <http://linkeddata.org/>.

value, or a nested specification list. An attribute can also be modified. For instance, the city *Boston* is represented by `City = Boston`; an unknown number of games is represented by `Gamenumber of = ?`.

Semantic knowledge

The semantic knowledge necessary to map questions to the data structure is defined in the lexicon on the one hand, and in a set of semantic rules (called subroutines) on the other hand.

Lexicon The lexicon maps words or idioms to their meaning in the same data structure presented section 2.4.1 on the preceding page as well as their POS (part-of-speech). Wh-words are also referenced in the lexicon.

Subroutines Subroutines are a set of semantic rules that modify the query representation or make choices in cases of ambiguity. For instance, a word that can have two POS (noun and verb) is disambiguated with the help of some heuristic, like the fact that any sentence can only have one main verb. A meaning modification routine consists for instance in adding a modifier in an attribute. For example, the word ‘team’ has the meaning `Team = (blank)`. The word ‘winning’ before the word ‘team’ will lead to the modification `Teamwinning = (blank)`.

Question translation step by step

Dictionary lookup The question is first tokenized in words, and empty words are left aside. The remaining words as well as adjoining words are looked up in the lexicon for the POS and the meaning. The output is a list of attribute/value pairs with extra information like the POS of each word and if the word is a wh-word.

Syntactic bracketting POS of each word is used to syntactically analyze the question in term of phrases. Phrases are surrounded by brackets and the main verb is left aside. The parsing proceeds from right to left and bases on heuristics. For instance, prepositions are associated to the rearest right noun phrase to generate a prepositional phrase. Each phrase is then tagged with its functional role in the sentence (subject and object).

Subroutines activation Some words trigger additional rules that modify the data structure of the query and/or disambiguate some words. For instance, the word ‘What’ followed by the word ‘team’ (whose meaning is `Team = (blank)`) modify the latter meaning to `Team = ?`.

Shortcomings

The main shortcoming of the system is the cost that is required to produce the semantic knowledge base (the lexicon and the set of semantic rules). Authors suggest an improvement for handling unknown words, where the meaning could be expressed based on existing words in the lexicon. Porting the system to

another domain requires a significant effort, since one needs to rewrite entirely the knowledge base.

2.4.2 LUNAR [72]

LUNAR was published more than ten years after BASEBALL. LUNAR (unlike BASEBALL) has been experimented with scientific data and targets expert users.

Data structure

Authors do not give much details about the data (provided by the NASA⁵). It looks almost like relational tables with a dedicated formal query language. The data are about chemical analyses of lunar rocks from the Apollo 11 expedition⁶. The application domain is again very closed but more complex than that of BASEBALL; some expertise is required to validate the answered provided by LUNAR.

Internal query representation

Woods [72] has defined a meaning representation language which is used to represent internally users' intent. This language is a combination of propositions (whose evaluation leads to a truth value) and commands (or actions to be performed by the DBMS). Propositions are composed of database objects (classes or table names and instances or variables). Propositions are combined together with logical predicates like **OR**, **AND**, etc. Commands are **TEST** (to test the truth of a proposition), **PRINTOUT** to print out the evaluation of a proposition and commands for loops (**FOR**) to be used with a quantifier.

Question translation step by step

Syntactic parsing the input question is first syntactically parsed using a general-purpose grammar (domain-independent grammar). The grammar is based on Augmented Transition Network linguistic formalism [71], which is almost equivalent to the context-free grammar formalism. The syntactic parsing also needs a lexicon which contains terms belonging to the domain. It contains for instance technical names of samples recollected during the expedition. 'S10046' is thus recognized as a proper noun by the parser [74].

Semantic mapping a set of rules transform the syntactic parse tree into a meaning representation (see section 2.4.2). The rules are triggered both by the syntactic structure of the parse tree (the label of nodes like **NP** for noun phrase or **VP** for verb phrase) but also on the words present in the question (like 'S10046' which is a sample name in the lexicon, or 'contain' which has a semantics also defined in the lexicon). The rule results in a database query pattern with slots to be filled with items from the lexicon. As the system also supports quantification, authors presents some heuristics on how to resolve the attachment in the generated propositions.

⁵See <http://www.nasa.gov>.

⁶See http://www.nasa.gov/mission_pages/apollo/missions/apollo11.html.

Query execution The internal query representation is composed of commands that the database retrieval component understands and executes to retrieve data and display/print them.

Shortcomings

The shortcomings of the LUNAR system are threefold. First, the NL processor (called “English processor” by its authors) is closely tailored to “the way geologists habitually refer to” database elements [73]. Therefore, the system is highly domain-dependant, and the lexicon cannot be re-used in different domains. Secondly, authors state that it is tailored to very simple data structure, and that the system – or part of it – must be re-implemented in order to consider new data sources. Last but not least, the database entities matching approach is an exact term matching technique, which is not applicable in most domains (e.g. the *logic name* given by the database administrator to tables is usually different from the *conceptual name* considered by experts of the domain). As an example, compare logical names in table 6.2 on page 122 and conceptual names on figure 1.2 on page 3.

2.5 Complex question translation

This class of approaches aims at increasing the linguistic coverage of NL interfaces. In addition to the systems of this class need domain knowledge, but the involved processing are independant from the underlying DBMS.

As for domain-dependant systems, the domain-dependent knowledge base corresponds to a lexicon which defines the meaning of words and expressions. This meaning is expressed with a set of semantic rules that map words and expressions plus syntactic information to fragments of database queries.

Besides, the domain-independant knowledge base is composed of the following components:

1. a syntactic parser which operates iteratively with the semantic rules
2. a set of NLP tasks that aim at resolving linguistic ambiguities (such as anaphora resolution and ellipsis resolution [6])

The syntactic parsing might be performed iteratively and in parallel with the semantic component process, as it is the case in IRUS [6].

Table 2.7 on the facing page overviews various systems that we compare in this section.

2.5.1 CHAT-80 [69]

This system is the ancestor of many future interfaces [3] like MASQUE [2]. The database is composed of facts about world geography (facts about oceans, seas, rivers, cities and relations. The database itself is implemented as ordinary Prolog. Questions are expressed in a subset of English. The subset of English questions is a formal but user-friendly language [69]. The main difference with its predecessor (LUNAR) is that much effort has been spent on increasing the linguistic coverage. In particular, the system translates English determiners (‘a’, ‘the’, ‘some’, ‘all’, ‘every’) and negation and focus on linguistic phenomena,

System	D	SK	S
CHAT-80	Prolog	vocab. (100 domains)	NL ambiguities
		domain-indep. knowledge base	presuppositions
QWERTY	temporal DB	semantic rules	grammar tailored to DB schema
		semantics of temporal PPs	
IRUS	hierarchical DB	domain-dep. dictionary	domain knowledge to generate MRL
		interpretation rules	
		linguistic resources	
PRECISE	relational DB	lexicon	prevalence of tractable questions
		semantic rules	SQL-specific lack of user-feedback
PANTO	triple store	dom. indep. lexicon	parser limitation
		authoring tool	SPARQL expressiveness

Table 2.7 – Systems belonging to the *complex question translation* range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.

like noun attachment and transformational aspects (that cannot be covered by context-free grammars).

Portability

The authors claim that the system is adaptable to other application [69]. In particular, it is composed of a small vocabulary of about 100 domains (excluding proper nouns), and a small domain-independent knowledge base of about 50 words.

Lexicons

The system is composed of a small vocabulary of English words that are related to the database domain, plus a dictionary of about 50 domain-independent words. These lexicons consist in rules in the Extraposition grammars formalism [52]; those rules are processed by Prolog and output Prolog clauses. In addition to these two lexicons used to parse the question, a dictionary is made up of semantic rules in the form of templates that define how a word associated to a predicate must be also associated with its arguments.

Question translation steps

Parsing The parser analyzes the syntactic categories of words and determiners (domain-independent lexicon) plus nouns and verbs which are database-related elements (domain-dependant lexicon). Proper nouns are represented by logical constants while most verbs, nouns and adjectives are represented as predicates with one or more constants.

Interpretation The output of the parser is interpreted by filling the predicates identified in the previous step. This is performed using a set of templates that are part of the system’s initial configuration.

Scoping This step consists in defining the scope of determiners and some operators (for instance the operator that counts items).

Planning The output of the previous step is a logical expression. However, to avoid combinatory explosion when executing the Prolog query, some strategies to optimize the query have been implemented: reordering the predications in the Prolog query; putting braces around independent subproblems to avoid too many backtracking procedures

Limitations

Constraints in NL are not convered (for instance “Which ocean. . .” presuppose there is only one right answer).

Query execution Even relatively complex queries are answered in less than one second [69]. The Prolog expression is executed to retrieve the answer. Authors note however, that the answering process (i.e. query execution) is the limiting factor (while modern systems are limited by the question analysis task).

2.5.2 QWERTY [49]

The specificity of this system is that it is intended to interface temporal databases. This system has thus an increased linguistic coverage and a better temporal expressivity. Input questions are expressed in controlled NL. The grammar used in the system takes into account some aspects of temporality of NL such as tenses and temporal PPs that modify sentences. The system produces queries in SQL/Temporal query language, which is dedicated to temporal databases.

Portability

The grammar used for parsing questions and translating it into the formal language is specifically designed for use with a particular database schema. Thus, this system cannot be considered as a portable system.

Question translation steps

Semantic parsing The NL question is parsed using the Type Grammar framework. While parsing, the question is being trasformed into a logical representation called L_{Allen} . This formal language is based on interval operators. The translation bases on a linguistic theory, that semantics of sentences is modified by temporal preposition phrases (PPs). In this work, PPs are considered as variants of standard generalized quantifiers, where the quantification is over time. The temporality in NL questions can be explicit (like ‘When’, ‘Which year’) or implicit (“Did Mary work in marketing?”). The quantification

also allows iterations of PPs (“every year until 1992”). In addition to temporal quantifiers, the system recognized quantification over individuals (“some employees”), coordination and negation. The semantic mapping to logical expression is performed basing on a bottom-up approach, simultaneously with the parsing.

Query translation The logic expression is translated in SQL/Temporal, the database query language dedicated to temporal databases. Some logical query produce infinite SQL/Temporal queries. Some heuristics have been implemented to prevent such behaviour.

Query execution The SQL/Temporal query is finally evaluated by the database engine to produce the answers.

Shortcomings

The translation of NL questions in Temporal/SQL is performed using a grammar which is tailored to the schema of the database. Therefore this system cannot be used for other databases.

2.5.3 IRUS [6]

The IRUS system processes the question independantly from the underlying domain and DBMS, which is a big change with respect to previous systems. The meaning formalism used to represent internally the query is the same as in the LUNAR system (MRL, namely meaning representation language) but its expression is domain and DBMS-independent. Besides, IRUS analyses linguistically the question and integrates state-of-the-art NL processing components, such as anaphora and ellipsis resolution.

Internal query representation

The internal meaning representation language is a descendant of that of Lunar. The language has the following general form [6]:

(FOR <quant> X / <class> : (p\ X) ; (q\ X))

where **quant** is a quantifier such as EVERY, SOME, THREE, HALF, etc., X is the variable of quantification, <class> is the class of quantification of X, (P\ X) is a predicate that restricts the domain of quantification and (q\ X) is an expression being quantified, or an action such as PRINT\ Y.

Question translation steps

Syntactic parsing The syntactic parsing of NL questions is performed using the ATN grammar formalism. The authors claim that the syntactic parser can benefit from semantic mapping as well in the syntactic parsing, both evolving in a *cascaded system*. The output of the syntactic parsing is a parse tree, where nodes correspond to syntactic information about question words and phrases

Semantic mapping The semantic mapping is done in interaction with the syntactic parser. It requires a domain lexicon, which defines the semantics of the words and expressions used in users' queries. The main subtasks involve disambiguation (pronouns and other anaphoric expression resolution, ellipsis resolution, references resolution through discourse information).

Query execution The meaning representation language can be used to interface any database system, at the condition that there is a component responsible for the translation from the internal language to the target query language.

Shortcomings

The system is said *transportable* – portable to new application domains and can interface any DBMS – but a new domain-specific knowledge base must be provided. Authors propose as next step an authoring tool where this knowledge can be written by expert users.

2.5.4 PRECISE [54, 53]

PRECISE maps questions expressed in natural language to SQL queries. The interesting approach in this system, is the introduction of *semantically tractable questions*. This class of questions is guaranteed to be mapped to the correct SQL query. This reduces then the classic gap between the query space and the data space.

Internal query representation

The internal query representation is different from the one of systems presented so far. The system builds an *attribute-value graph* that maps words of the question to database elements, and a *relation graph* that maps *relation tokens* (i.e. some words of the question) to the names of relations belonging to the database. Both graphs are eventually used to generate the SQL query.

Lexicon

The lexicon defines the mapping between tokens and database elements. It is composed of 1) the tokens; 2) the database elements; 3) the binary relations that bind both tokens and database elements.

Question translation steps

Syntactic parsing A lexicon is composed of elements that have been automatically extracted, and is used to perform the matching with question tokens. NL processing task are: tokenizing the question into words, categorizing those tokens into *syntactic markers* (i.e. empty words) and *tokens* which are words that can be potentially associated with database elements. In addition to the lexicon, the system is composed of a parser which is implemented as a plug-in. The parser can thus be changed for experimentation purposes. Authors have experimented the system with a syntactic dependency parser which outputs a graph (namely an *attribute/value graph*) composed of paths linking database

elements together. These paths will then be composed together based on aggregation and combination of foreign keys in the case of relational databases. The lexicon is also composed of a set of restrictions corresponding to prepositions and verbs. These restrictions define the join paths connecting relation and attributes. The set of those restrictions defines the semantic part of the lexicon. A component is also responsible for classifying questions into *tractable* and *not-tractable* questions. This is done linking words of the question with a set of *compatible* elements of the database. PRECISE also implements strategies for correcting syntactic parsing errors (namely *semantic over-rides*) based on the semantics defined in the lexicon.

Semantic mapping the interpretation in terms of structured query consists in choosing a path between database elements. This choice is a constraint satisfying problem defined in the parse tree. Paths generate SQL fragments that are then aggregated.

Feedback component

In the case where there is no possible interpretation (even trying to correct possible syntactic errors) the system asks the user to rephrase the question.

Shortcomings

The main contribution of the system is the ability of predicting whether a question can be answered (*tractable questions*) or not. Experiments have been lead in a few domains (geography, restaurants and jobs) but there is no proof that tractable questions are prevalent *in general*. Besides, the semantic knowledge component contains semantic rules that define how to translate two graphs (namely *attribute-value graph* and *relation graph*) to the target query in SQL. There is no claim regarding the cost of changing the target query language (here SQL). Moreover, authors conclude on the user-feedback component, and suggest that users should get better information on why answering a question had failed.

2.5.5 PANTO [68]

PANTO generates SPARQL queries, that can be executed to get answers to the information need expressed through a question in NL. The data are organized in a knowledge base, more specifically an ontology (RDF or OWL⁷ formalism). The most interesting aspect in PANTO is that its most important component (the linguistic component, i.e. the parser) is implemented as a plug-in component, that can be easily replaced by an other one. This permits thus to benefit from the improvements in terms of linguistic coverage, when integrating state-of-the-art parsers.

Portability

“PANTO is designed to be ontology-portable”. To ensure portability, PANTO comprises a domain-independant component. This component is a lexicon

⁷See <http://www.w3.org/TR/owl-features/>.

composed of WordNet⁸ entries. For portability purposes, users can collaborate and improve the domain-dependent component (the domain ontology) defining their own synonyms to be added in the lexicon which bases the question parsing step.

Internal query representation

The internal query representation is a graph representation called *query triples*. It is a representation of the parse tree, that is then mapped to database queries (i.e. RDF triples) thanks to the lexicon.

Question translation steps

Syntactic parsing words from the NL query are first mapped to entities (concepts, instances, relations) of the ontology. This corresponds to the “entity recognition” task, and several tools are used, such as WordNet and string metrics algorithms. Then, a syntactic parser is used to recognize nominal phrases in the question. Those phrases are represented by pairs in the parse tree. In addition, some NLP tasks are integrated in this step, such as the recognition of negation.

Semantic mapping Pairs of nominal phrases from the parse tree are associated to triples in the sense of the ontology, and composed of entities of the domain ontology. This association is performed using the domain knowledge (the database). Besides, two additional components are involved in this step: the question target identifier which identifies the target in the parse tree, and a component responsible for the recognition of solution modifier in the sense of SPARQL (e.g. commands `FILTER` or `UNION`). Those components basically base on rules triggered by the recognition of some words (e.g. *wh*-words for the former component). The triples mentioned before along with the target and modifier information constitute the internal representation of the query.

Query execution The internal representation of the query mentioned above is interpreted into SPARQL statements, that can be then executed to retrieve requested facts. The target item is used in this step to decide what to put after the `SELECT` command in the generated SPARQL statement. Query post-processing procedures are triggered, for example basing on the negation recognized in the parsing step. The negation is usually translated in the `FILTER` SPARQL clause to specify the set of triples that must not appear in the result of the execution of the SPARQL statement.

2.6 Feedback-driven approaches

The systems presented in the previous section still require intensive configuration efforts, and thus cannot be considered as *portable*. As a result, several systems have arisen where semantic grammars are created automatically on the basis of user interaction. We distinguish between *configuration* (authoring

⁸See <http://wordnet.princeton.edu>.

System	D	SK	S
TEAM	<i>any</i>	domain-dep. lexicon	NLP limitations
		semantic rules	no proper eval.
		authoring tool	missing discourse knowledge
MASQUE/-	Prolog	lexicon (logic pred.)	exec. time
SQL	relational DB	authoring tool	use-feedback
NALIX	XML DB	lexicon	NLP limitations
DANALIX	XML DB	lexicon	no evaluation
		dom.-indep. knowledge	XQuery only
		user interaction	
C-PHASE	relational DB	lexicon	basic evaluation
		semantic rules	
		sentence patterns	
ORAKEL	triple store	dom.-indep. lexicon	ling. assumptions
		dom.-dep. lexicon	labels in DB are concepts/instance names
		semantic rules	
		authoring tool	

Table 2.8 – Systems belonging to the *feedback-driven* range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.

tools) and *knowledge acquisition* through user interaction when the system is being used.

Table 2.8 sums up the different systems that belong to this class of approaches.

2.6.1 TEAM [28]

The data are structured in a database about geographic facts like the largest cities in the world, the population of each country etc. The TEAM system is intended to be used with two kinds of users: standard users and database experts, who engages dialogue to provide needed information to port the system to other application domains. This system belongs thus to *configurable* systems, where the required knowledge to port the system is provided by the expert user, interacting with a dedicated authoring tool.

Internal query representation

The meaning of users’ queries are internally represented in formal logic.

Lexicon

A lexicon is used to map words and expressions of NL to their meaning in terms of database elements. Close classes of words are supposed to be domain-independent and to have a fix meaning. Open classes of words, however, have a much more important frequency in users’ queries, and their meaning is supposed to be domain-independent. The *meaning* is composed of both syntactic and semantic information. Entries for nouns are the instances to which their

refer, or a class (or concept) in a type hierarchy; entries for adjectives and verbs correspond to the possible predicate and how to find arguments of the predicate in the NL question.

Database schema

In addition to the lexicon, a resource about how logical forms can be translated in terms of database elements must be available. This resource expresses for instance the link between predicates (that appear in the logical forms) and database relations and attributes or the definition of the class hierarchy in terms of database relations or fields.

Portability

Portability is performed in using the system in a different mode (*knowledge acquisition*). In this mode, the database/domain expert informs the system on how data are organized in the database, what are the database elements and what words and expressions from NL are used for those elements. The acquisition consists in a tool, where the expert must answer questions; the answers of those questions will impact the resources like the lexicon and the database schema, that are both used to interface the database.

Question translation steps

The system is divided into two sub-systems: DIALOGIC that maps NL questions to formal expressions and a schema translator that translates formal logical queries in database queries.

Syntactic parsing The parsing of the NL question is performed using an augmented-phrase structure grammar. The parser produces possibly several parse trees for one question; then one of the parse trees is selected basing on syntactic heuristics.

Semantic mapping Several processings are responsible for resolving some domain-specific ambiguities like noun-noun combinations and vague predicates like ‘have’ or ‘of’ [28]. Finally, a quantifier determination process is triggered. In the end, a logical form of the question is identified.

Query generation The logical form is then translated in the database query. This translation bases on the conceptual schema as well as the database schema (which defines the structure of the database) to perform the translation.

Shortcomings

TEAM relies on some Natural Language Processing (NLP) modules, and therefore the overall performance of the system depends on the performances of these tasks. Moreover, the system is able to retrieve facts from the database, but not to aggregate these facts.

2.6.2 MASQUE/SQL [2]

MASQUE is a NL interface to Prolog databases, and MASQUE/SQL is an extension of it that supports SQL query language. The system is entirely written in Prolog. The system is meant to be domain-portable. Users can indeed add new entities in the lexicon through a domain editor.

Lexicon

The lexicon defines the semantic of words that are expected to appear in users' questions. The meaning of words are described in logic predicate. Possible argument types of predicates are organized using the hierarchical *is-a* relation.

Internal query representation

The internal query representation is a Prolog-like language called Logical Query Language. Question words are translated to predicates or predicate arguments. The types of those arguments are described in the *is-a* relation hierarchy

Portability

The system can be used with different domain databases, but this requires to edit the specific knowledge in a dedicated editor. This knowledge consists in entities linked with the hierarchical *is-a* relation. User also has to explicit links between words and corresponding logic predicates; the entities in the taxonomy are used to restrict the possible arguments of the predicates. Each predicate is also linked to the corresponding SQL statement.

Question translation steps

Syntactic parsing A dictionary consists of all English words that the system understands and is used by an extraposition grammar to parse the question.

Semantic mapping The dictionary composed of lexical units also associates words to their meaning in the form of logic predicate. The internal formal representation is a Prolog-like meaning representation language (LQL).

Query execution The internal representation is translated to SQL using an algorithm, and then SQL is executed in the underlying DBMS. The translation consists of rules triggered by the structure of the LQL expression; each unit LQL expression is associated with a SQL fragment; all fragments are then combined to produce the final SQL expression. The system has also been experimented with Prolog as query language in association with a Prolog database.

Shortcomings

Authors report two major limitations of the system. First, the execution time of the system for any query is not less than 6 seconds. Secondly, users are not properly informed of the reason why their queries have failed (i.e. which step of the processing has failed).

2.6.3 NALIX [42] and DaNaLIX [41]

NALIX is an interactive interface to XML databases for questions expressed in natural language.

Question history

The system is composed of a query history that keeps all successfully answered queries. Queries from this history are intended to be used as templates for formulating new queries. This feature also permits users better understand the linguistic coverage of the system.

Internal query representation

The internal query representation is the target database query, i.e. XQuery⁹.

Portability

There is no internal representation of the query (the NL question is directly translated to an XQuery expression). Thus, the translation is dependent on the DBMS and the target query language (i.e. XQuery). In addition to the portability feature of NALIX, DANALIX takes advantage of domain-dependent knowledge which can be automatically acquired on the basis of user interaction. When ported to a new domain, the system starts with a generic framework; then domain-dependent knowledge is being learned from user interaction.

Question translation steps

Syntactic parsing The parsing consists in identifying words and phrases using the MINIPAR dependency parser (dependency among words and not hierarchical constituents). An other component is responsible for checking whether words and phrases identified in the previous step can be mapped to directives (e.g. **return** or **group by** clauses) in the target query language (XQuery). Each word and phrase that can match a directive is further typed, depending on the kind of directive. The output of the parser is a tree with those roles as labels of phrases of the initial query. The vocabulary mismatch is overcome using WordNet. In DANALIX, an additional step consists in transforming the parse tree using domain knowledge basing on relevant rules.

Semantic mapping Each item from the parse tree is translated into a fragment of the target query language (XQuery). This is done using a series of processings. Basically, a set of rules define how to combine items of the parse tree to get XQuery clauses. There are also further treatments like *nesting* and *grouping* developed in [43]. In cases when the system does not understand how to map a given phrase to a XQuery constituent, the system interacts with users and suggests potential reformulations. NALIX seems to be the first NL interface that introduces user interaction to select the correct parse interpretation.

⁹See <http://www.w3.org/TR/xquery/>.

Query execution The question is directly translated into a XQuery expression, which is the data query language. The XQuery expression is then executed to retrieve answers. Answers are basically XML answers. Different visualization of the answers are possible (text view for simple answers, hierarchical list view or raw XML). Besides answers to queries, the system implements an advanced error manager that also supports users in rephrasing queries that were not correctly parsed nor accurately translated.

2.6.4 C-PHRASE [47]

C-PHRASE is a system that translates questions expressed in NL in queries for relational databases. The system outputs expressions in tuple calculus – closed to First Order Logic (FOL) – that can be easily translated in a database query language like SQL.

Lexicon

Semantic information are encoded in a lexicon. It maps tokens with syntactic information (such as the head and the modifier in dependency analysis) and semantic information (translation in λ -calculus). It is represented as a set of rules that form the grammar of the parser. An example of such a rule is:

$$\text{HEAD} \rightarrow \langle \text{"cities"}, \lambda x. \text{City}(x) \rangle$$

In addition, the system comprises a set of sentence patterns, in the form of context-free rules. Such a rule is for instance:

$$\text{QUERY} \rightarrow \langle \text{"list the".NP}, \text{answers}(x | \text{NP}(x)) \rangle$$

Authors say that they “rarely” see users who do not use already defined sentence patterns. This set of rules are automatically created by an authoring tool. The tool explicitly asks for meaningful names of different database elements (i.e. relations, attributes and join paths). It also permits to define new concepts in NL.

Internal query representation

The internal query representation is λ -calculus.

Question translation steps

Semantic parsing The parsing bases on a context-free grammar, augmented with λ -calculus expressions. The framework (λ -SCFG) works with two parsing trees: one for parsing syntactically the NL sentence; the other one for expressing the semantics of the first one in λ -calculus. Input questions are first tokenized and normalized. Then, the sequence of tokens is analyzed to identify database elements, numeric values, and proceed to some spelling corrections. The structure is then transformed in the form of tuple calculus queries. This is done based on a set of rules that map lexical and syntactic parses to tuples. Each rule has a plausibility; in the end, the product of plausibilities of all rules used in a parse is used to rank potential semantic interpretations. In case of ambiguity, the user is asked to rephrase the question, or to select the best rephrasing proposition.

Query generation The tuple query is converted in SQL.

Shortcomings

Authors of the C-PHRASE system suggest that the system should be further evaluated. Besides, they point out that bootstrapping the authoring tool is a tedious task.

2.6.5 ORAKEL [12]

ORAKEL [12]’s main feature is the portability. This portability is based on the use of subcategorization frames, which are linguistic structures composed of predicate and arguments. The feedback consists in an authoring tool. This tool (which is called FRAMEMAPPER) is used to generate the domain-specific knowledge and is intended to be used by expert of the domain.

Role of lexicon

The system is composed of three kinds of lexicons:

- domain-independant lexicon defining determiners, *wh*-pronouns and spatio-temporal prepositions
- domain-specific lexicon which defines the meaning of verbs, nouns and adjectives
- ontological lexicon which is automatically created from the data, and maps ontology instances and concepts to proper nouns and standard nouns

Categories of the domain-independant lexicon are generic categories such as those of a generic ontology like DOLCE¹⁰.

Internal query representation

The internal query representation is λ -calculus. The question is represented in a language which is an extension of FOL (in addition to FOL: quantifiers and operators).

Customization process and portability

Users can customize the system, i.e. create a domain-specific lexicon which maps subcategorization frames (arity n) to an ontology relation (arity n). This mapping is called a *definition*. The interesting thing is that binary relations where the range is an integer – for instance *height(mountain, int)* – can also be mapped to adjectives with various degrees (base, comparative, superlative forms) and possibly the positive or negative scale (as in TEAM). This mapping is performed by users themselves through a front-end tool.

¹⁰See <http://www.loa.istc.cnr.it/DOLCE.html>.

Question translation steps

Semantic mapping Both processes question parsing and semantic mapping are done in a single process. The parsing is based on the LTAG¹¹ linguistic representation. The parsing operates in a bottom-up fashion: each question word is associated with an elementary tree. Then, all elementary trees are combined together to get the syntactic parse tree of the entire sentence.

Query execution The FOL internal query representation is then translated in the database query language like SPARQL (for ontologies) or the language used for ontologies expressed in F-Logic. This translation is performed using a Prolog program.

Shortcomings

Supports only factoid questions (*wh*-questions plus questions starting with expressions like “How many” for counting) but not complex questions like those starting with ‘why’ or ‘how’. Besides, there is a strong assumption that categories of the domain-independent lexicon should be aligned with those of the database (the domain ontology). The second one, is that the generation of the ontological lexicon assumes that the labels used in the database correspond to instance and concept names. The front-end system used to improve the domain-specific knowledge allows to increase the linguistic coverage and is a nice tool for porting the interface; however it might not be user-friendly since the underlying concepts (like the semantic frames) are complex.

2.7 Learning-based approaches

Learning-based approaches are approaches where machine learning algorithms are the core of the translation mechanism. These algorithms aim at learning domain-specific knowledge (e.g. a lexicon). This knowledge is used to parse the question and get clues on how to translate it to a database query.

Table 2.9 on the following page compares the different systems that belong to this class. The first system (M. ET AL. in short) is more theoretical, and therefore has not been evaluated with any specific database implementation. In the following, we further describe each of these systems.

2.7.1 Miller et al. [46]

The main characteristics of this system is that it is fully statistical. It is composed of three components for parsing, semantic interpretation and discourse resolution and are associated with corresponding statistic models. Each component produces a set of ranked items, and the chosen interpretation of the question is the best one of the final component.

Question translation steps

The different steps correspond to the different components of the system.

¹¹Lexicalized Tree Adjoining Grammar, see <http://www.cis.upenn.edu/~xtag/tech-report/>.

System	D	SK	S
M ET AL.	<i>any</i>	stat. models	no evaluation
Z. ET C.	λ -claculus DB	dom.-dep. lexicon semantic rules	only λ -calculus
WOLFIE	Prolog DB	machine-learning (lexicon)	no eval. for long phrases

Table 2.9 – Systems that incorpore machine learning approaches in order to interface structured data. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.

Syntactic parsing The string of words W is searched for the n -best candidates of parse trees T basing on the measure $P(T)P(W|T)$. The parse tree contains syntactic as well as semantic information. The statistical model is based on the recursive transition network model¹². This model is more detailed in section 2.3.4 on page 29. The model is trained with a set of questions annotated with the corresponding parse trees.

Semantic mapping The semantic mapping step is composed of two sub-steps: first, a model associates a *pre-discourse meaning* to both a parse tree and the corresponding string of words. Second, a *post-discourse meaning* is retrieved from the pre-discourse meaning, the string of characters, the parse tree and the history. Both pre-discourse meaning and post-discourse meaning are represented using semantic frames. The construction of the frames is integrated in the parse tree. Then, the statistical model is used to disambiguate the frames, for instance when there is no information about the frame type or when there is no information about a slot to fill. The second phase (post-discourse meaning) corresponds to ellipsis resolution. It takes the previous meaning (final meaning of previous questions) and the pre-discourse meaning (of the current question), and finds the best meaning. Previous meaning and pre-discourse meaning are represented as vectors, whose elements correspond to slots in the frame meaning representation. The statistical model applies different kinds of operations on those vectors (INITIAL, TACIT, REITERATE, CHANGE and IRRELEVANT). Those operations combine elements of both vectors to compute the vector which corresponds to the meaning of the current question.

Query generation The paper does not explain how is performed the query generation.

2.7.2 Zettlemoyer et Collins [76]

The system aims at translating NL sentences to λ -calculus expressions. The system is based on a learning algorithm that needs a corpus of sentences labelled with λ -calculus expressions. The system induces then a grammar. The statistic model is a log-linear model. The paper focus only on the generation of λ -calculus expression from questions expressed in NL.

¹²See <http://www.informatics.sussex.ac.uk/research/groups/nlp/gazdar/nlp-in-pop11/ch03/chapter-03-sh-1.1.html>.

Portability

The proposal has been tested for two application domains: a database of US geography and a database of job listings.

Internal query representation

The query is internally represented in λ -calculus.

Question translation

The syntactic parsing is performed using a combinatory categorial grammar. The nodes of the resulting parse tree are composed of both syntactic and semantic information. The grammar of the parse operates with a domain-specific lexicon, which maps question words and expressions to a syntactic type as well as a semantic type. Several functional rules define how syntactic types can be associated

Learning component

The learning algorithm takes as input a training set composed of pairs (S_i, L_i) where S_i is a string of words and L_i a logical form. The algorithm also takes as input a lexicon. The learning algorithm will determine how a string of words will be parsed to produce a parse tree, and what words from the lexicon are required to produce such trees. The algorithm involves then learning the lexicon and learning the distribution over parse trees for a given string of words.

Shortcomings

The experiments are based on databases rewritten in λ -calculus. It would be interesting to implement an additional module that transforms λ -calculus expressions to any database query language and measure the overall performance.

2.7.3 WOLFIE [66]

WOLFIE is a system that learns a lexicon that is used to translate NL questions in database queries.

Lexicon

The lexicon consists in a mapping from sentences to semantic representation (logical database queries). The mapping consists in two steps: first, phrases that compose the sentence are associated to database elements (or *symbols* in the representation). Second, it says how those intermediate representation must be combined to get the final database query.

Learning

Due to the nature of the lexicon, the number of possible interpretations of a NL sentence is huge (the problem is computationally intractable). To cope with that, authors propose an active learning algorithm which selects only most relevant examples to be then annotated. The learning algorithm bases on a greedy

System	D	SK	S
POPWERAQUA	triple stores	dom.-indep. lexicon	data indexing assumption
DEEPQA	semantic data	dom.-indep. lexicon dom.-dep. lexicon	primarily designed for
	sources	syntactic frames semantic frames	unstructured documents

Table 2.10 – Systems belonging to the *schema-unaware* range of approaches. ‘D’ stands for “data structure”, ‘SK’ for “semantic knowledge” and ‘S’ for “shortcomings”.

approach. The generation of candidate meanings for each phrase consists in finding maximally common meaning for each phrase. But, the algorithm also aims at finding a general meaning for the whole sentence and not only meanings for the phrases that are part of the sentence. The evaluation corpus is composed of 250 questions on US geography paired with Prolog queries.

Shortcomings

The system has not been evaluated with long phrases; the paper describes how it works for phrases of maximum two words. Besides, “the algorithm is not guaranteed to learn a correct lexicon in even a noise-free corpus”.

2.8 Schema-unaware approaches

This section describes modern systems that are portable, because they operate accross various domains simultaneously (they interface databases of various domains). Table 2.10 sums up the different systems that are surveyed in this section.

2.8.1 POWERAQUA [44]

POWERAQUA takes as input several semantic sources (heterogeneous ontologies) and a question (expressed in NL). This system is an improvement of the system Aqualog [19]. The latter system was able to answer questions related to one domain ontology only (which is a problem in the context of the SW composed of many heterogeneous ontologies). The geatest contribution of POWERAQUA is the method for mapping user terminology with ontology terminology (and the system is unaware of the data structure of the semantic sources).

Domain portability

The portability cost is negligible. Indeed, the system has a learning component responsible for the acquisition of domain-specific lexicon which maps users’ relations (expressed in NL) to knowledge represented in the domain ontology.

Internal query representation

The internal query representation is called *query triples*. It is triples like RDF triples used in the data (ontology), but these triples are based on terms of the query.

Lexicon

POWERAQUA as well as AQUALOG use domain-independent lexicon WordNet.

Question translation steps

Syntactic parsing Users' questions are first parsed syntactically, and questions are categorized (for instance on the basis of the *wh*-question word). Both the parse tree and the category of the question are used to generate the internal query representation, query triples. The sense of words used in the question are disambiguated using word sense disambiguation algorithms. Those query triples are composed of question words, and may be then modified so that they are *compatible* with the database. The linguistic component of POWERAQUA also consists in algorithms that make decisions, for instance for correctly interpreting the conjunctions and disjunctions terms ('and'/'or'). The question may also contain constraints, that are sub-questions to be answered prior the actual question. Furthermore, the system provides a way to treat two instances (from two semantic sources) as equivalent, and thus ease the answering process (in particular in cases where more than one semantic source are required to answer the question). The mapping from users' terms to database elements is done using a metrics similar to the edition distance.

Semantic mapping the semantic mapping consists in processing query triples to retrieve the ontology triples that are associated to the query triples. First, the algorithm tries to filter the sources (the ontologies) in order to only consider those that contain all or most of the *query triples*. Query triples must then be filtered to take into account the potential high number of triples generated in the previous step, and the fact that a question can involve several different semantic sources. The mappings established in the previous step (query terms and ontology terms) are ranked basing on a sense disambiguation algorithm that uses WordNet and the *is-a* taxonomy. In this step, the equivalence between two terms is computed based on the label of the term (edition distance) but also on the position in the taxonomy (ancestors and descendants).

Query generation Potential ontology terms (database elements) must be processed to generate the final ontology query. Terms identified in different relevant ontologies must be used to generate triples (first sub-step); these triples must then be linked, these triples belonging or not to the same ontology (second sub-step). In this step, relations will be created from terms identified in the previous step. A relation (in the query triple) does not always correspond to a relation (expressed in ontology triple): sometimes, a new triples must be generated.

Limitation

Authors make the assumption that the SW provides an indexing mechanism.

2.8.2 DEEPQA [16]

The DEEPQA project is part of the well-known WATSON system. WATSON has been the major event in the Q&A community in 2010. Indeed, it is the first artificial system which has won the american *Jeopardy!* quizz. WATSON queries a wide range of different sources and aggregates the different results to produce answers.

Question

WATSON has been designed for *Jeopardy!* where questions are not standard questions, but *clues* that should help understanding what the question is about. For instance, in a classic Q&A process a question would be “What drug has been shown to relieve the symptoms of ADD with relatively few side effects?”. In *Jeopardy!* the corresponding clue would be “This drug has been shown to relieve the symptoms of ADD with relatively few side effects”. The expected response would then be “What is the Ritalin?” [17].

Answers

Another particularity of *Jeopardy!* is that an answer are not simply a phrase corresponding to what the clues are about, but the answer must be the expected question corresponding to the clues. For instance, a valid response can be “Who is Ulysses S. Grant?” but not “Ulysses S. Grant” [17].

Data

The data (sources) used by WATSON are mostly unstructured documents as in most Q&A systems. DEEPQA however also leverages databases and ontologies such as DBPEDIA¹³ and the Yago¹⁴ ontology; this is why we consider this system in this survey.

Question translation

Question translation for databases involves different steps that are detailed below:

Question analysis The parsing of the question involves many tasks: shallow parses; deep parses; logical forms; semantic role labelling; coreference resolution; relations (attachment relation and semantic relations); named entities; etc. Semantic relation detection is one of those tasks. For instance for the question “They’re the two states you could be reentering if you’re crossing Florida’s northern border”, the relation would be `borders(Florida, ?x, north)`.

¹³See <http://dbpedia.org/About>.

¹⁴See <http://www.mpi-inf.mpg.de/yago-naga/yago/>.

Hypothesis generation The module responsible for this step takes as input the analysis parses and generates candidate answers for every system sources. Those candidate answers are considered as hypothesis to be then proved basing on some degree of confidence. To produce candidate answers, a large variety of search techniques are used. Most of them concern textual search, but some consist in searching knowledge bases, more specifically triple stores. The different search techniques lead to the generation of multiple search queries for a single question; then the result list is modified to take into account constraints identified in the question. The search is based on named entities identified in the clue. If a semantic relation has been identified in the question analysis step, a more specific SPARQL query can be performed on the triple store. In this step, recall is preferred as precision, leading to the generation of hundreds of candidate answers to be then ranked according to the confidence of each candidate.

Soft filtering Candidate answers are not directly scored and ranked, since those algorithms require lots of resources. Instead, the hundreds of candidate answers corresponding to a single question are first pruned, to produce a subset of initial candidate answers. The involved algorithms are lightweight in the sense that they do not require intensive resources (soft filtering). The candidate that do not pass the soft filtering threshold are routed directly to the final merging stage. The model used to filter the candidate in this step as well as the threshold are determined on the basis of machine learning over training data.

Hypothesis and evidence scoring Candidate answers that successfully pass the previous step are scored again in this step. This involves a wide range of scoring analysis to support evidences for candidate answers. First, evidences for the candidate answers are being retrieved. In the context of triple stores, those evidences are triples related to entities and semantic relations identified in the question. Those evidences are then scored, to measure the degree of certainty of these evidences. For instance the score is determined on the basis of subsumption, geospatial and temporal reasoning. The DEEPQA framework supports a wide range of scorers (components) that provide scores for evidences with respect to a candidate answer. The evidences from different types of sources can also be combined (for instance from unstructured content and from triple stores) using a wide range of metrics. In the end, scores of evidences are combined into an *evidence profile*. This profile “groups features into aggregate evidence dimensions that provide an intuitive view of the feature view”.

Final merging and ranking This step aims at ranking and merging “the hundreds of hypotheses based on potentially hundreds of thousands of scores to identify the single best-supported hypothesis given the evidence and to estimate its confidence” (or likelihood).

Answer merging To cope with candidate answers that might be equivalent (e.g. leading to the same answer) but with different surface forms, authors propose to first find similar candidate answers and to merge them in a single candidate answer.

Final ranking After merging candidate answers, they must be ranked based on merged scores. This is done using a machine-learning approach. This approach requires a training dataset of questions with known answers with appropriate scores. In this step ranking scores and confidence estimation (estimation of the likelihood of a given candidate answer) must be computed in two separated processes in intelligent systems. For both processes, the set of scores can be grouped according to the domain and intermediate models specifically used for the task. The output of these intermediate models is a set of intermediate scores. Then, a *metalearner* is trained over this ensemble of scores. This approach allows for iteratively improving the system with more complex models, and adding new scorers. The *metalearner* uses multiple trained models to handle different question classes (for instance, scores for factoid questions might not be appropriate for puzzle questions).

Drawbacks

WATSON has been primarily designed for unstructured content: “Watson’s current ability to effectively use curated databases to simply ‘look up’ the answers is limited to fewer than 2 percent of the clues”.

2.9 Challenges

According to Hearst [33], challenges of modern NL interfaces to structured data are twofold:

- speech input (dialog-like interaction: Siri). This means that the future systems must be able to understand ill-formed sentences and misspelled words.
- social search (collaboration, asking people, crowdsourcing). This could also comprise systems that broadcast the question to other systems that are expert in some fields, and then aggregate respective answers.

The next generation of interfaces will not focus on the user (personalized systems) but on non-textual information through non-textual input [33]. A promising research direction is thus to analyze non-textual documents (pictures, videos, voice records, etc.) and try to find *clues* based on textual (or non-textual) formulations of any user’s information need.

Going back to BI, nowadays’ challenges are to make business tools more user-friendly in terms of user experience. In the last few years, the major challenge was to prevent users from writing structured queries (e.g. in SQL or MDX). Now, they do not want to spend too much time: 1- learning how to use their business tools; 2- how to adapt the existing tools to new infrastructures (e.g. DBMS) or domain. The latter challenge meets the ones of the so-called *gamification* domain which is a research area that has become popular in BI.

2.10 Conclusion

We have reviewed most significant systems that belong to NL interfaces. These systems are however only a tiny subset of all systems that interface structured

data, or of Q&A systems. For instance, a popular domain of research nowadays is the search over structured data on the basis of keywords (instead of natural language inputs). Besides, we tried to focus on recent systems, since Androutsopoulos et al. [3] surveyed systems in the mid 90's.

We proposed a rough taxonomy of approaches, namely *classic translation approaches* and *iterative translation approaches* that are then refined in the following classification of approaches: approaches basing on domain-dependant semantic parsing; approaches performing complex question translation; feedback-driven approaches; learning-based approaches and schema-unaware approaches. This classification is motivated by the various methods employed by the systems of these different classes. Early years of NL interfaces focused on specifics of some DBMS in order to generate valid database queries. Later on, while standards in database query languages (e.g. SQL) have appeared, the focus was more on increasing the linguistic coverage of the respective systems. Then, in parallel with the success of semantic technologies, the need has appeared to interface various databases of possibly different domains: the respective systems are not aware anymore on how data have been modeled, and what are the terms in use in the particular domain. This knowledge has then to be learnt automatically.

A learning from this survey, is that the performance of most systems (or maybe of all of them) rely on the performances of underlying tasks, namely natural language tasks (e.g. entity recognition, identification of frames, etc.) which still focus much researchers' attention.

Challenges for future natural language interfaces are even more *natural* interfaces, such as speech-to-text features (users won't have to type their queries anymore in the near future) as well as the role of communities (in particular *crowdsourcing* which is already very popular among the Q&A community).

In the next three chapters (chapter 3 on page 57, chapter 4 on page 75 and chapter 5 on page 97) we will present our contribution to the state of the art as a proposal for a system interfacing data warehouses in the context of BI.

Chapter 3

Personalized and Contextual Q&A

Outline

3.1	Q&A Framework	59
3.1.1	Authentication	59
3.1.2	Search & prediction services	61
3.1.3	Answering system	62
3.1.4	Search engines	67
3.2	Extensibility of the framework	68
3.2.1	Implementing new plugins	68
3.2.2	Configuration of linguistic resources	69
3.3	Multithreading and performance considerations	72
3.4	Conclusion	73
3.4.1	Summary & discussion	73
3.4.2	Organization of the next chapters	73

Previous chapter concludes on challenging systems, that they should not only offer *personalized* results, but also *non-textual* pieces of information on the basis of not-well formed queries, or even on the basis of non-textual formulation of an information need. To this end, we present a personalized Q&A system that leverages queries expressed in NL or keywords, and offers results in the form of charts or tables of values. This system has been entirely implemented, and details are provided in this chapter.

The notion of *context* of a given entity is defined by a set of entities that interact with this entity. Thus, the context of a user is defined by its environment – the application she is using, the documents she creates, the search that she performs, etc. It must also take into account her *social network*, or other users who have a common interest or goal which can be measured by some distance metrics. These notions are further detailed in [65] through an application to auto-completion of BI queries. The personalization process that is presented in this work takes into account not only characteristics about the user – her profile – but also so-called *preferences*, which are used basically to filter and disambiguate the vast amount of potential resources of interest for the current user.

This chapter is organized as follows: first, we present the Q&A framework itself, its architecture and how personalized results are rendered. Secondly, we detail the extensibility of the framework from the point of view of a system administrator who would like to extend its current capabilities. Thirdly, we elaborate on performance considerations and more specifically on the multithreaded implementation of the framework described in this chapter. Last but not least, we detail how we have integrated the notion of context introduced above in our system, and provide as an example further insight on *usage statistics*.

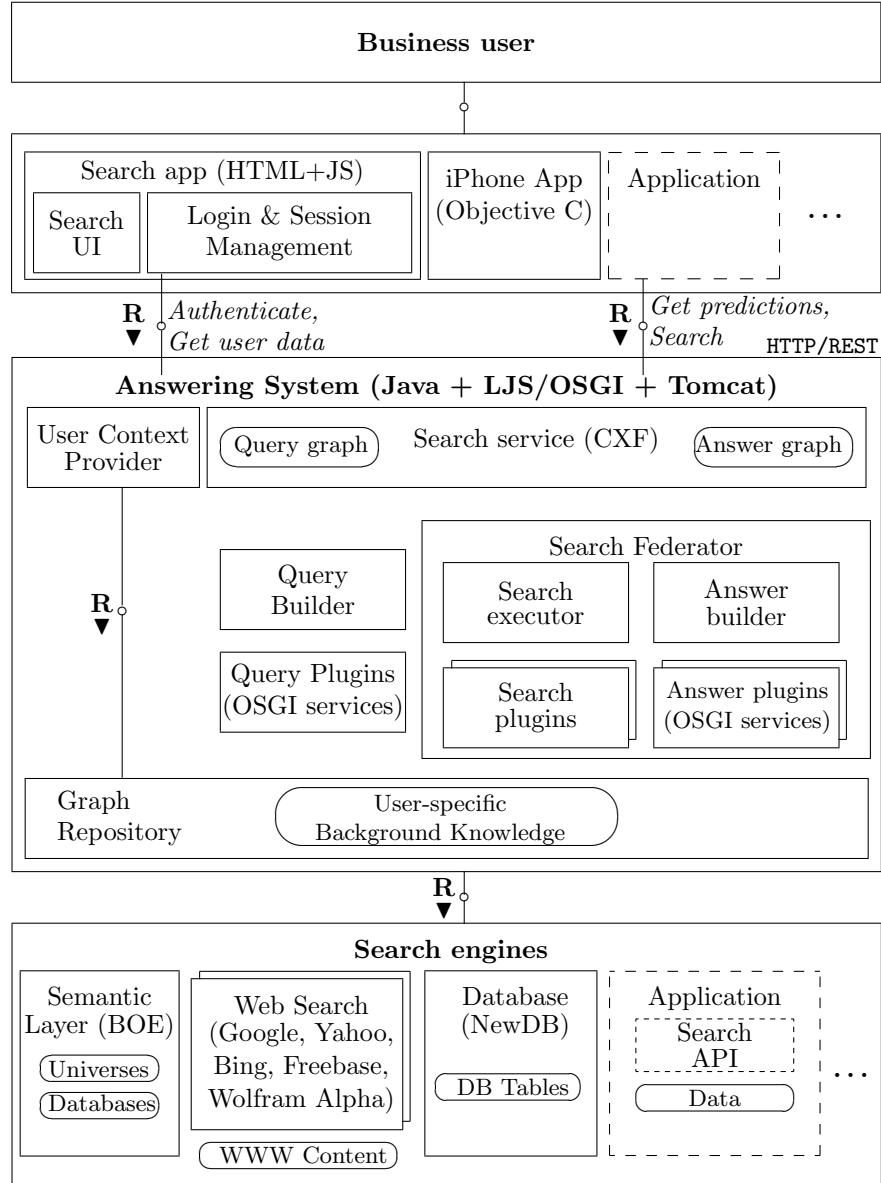


Figure 3.1 – General architecture of the answering system

3.1 Q&A Framework

As pointed out in the state of the art in chapter 2 on page 15, a limitation of many historic Q&A systems is that they are tailored to specific domains and sometimes datasets. Our proposal is a framework with standard linguistic *features*¹ that can be efficiently extended to specific domain applications.

The general architecture of the system is represented on figure 3.1 on the facing page. The upper ribbon corresponds to the different available front-ends. Currently, two front-ends have been implemented:

- a desktop application which is available as an HTML 5 application
- and an iPhone™/iPad™ native application

The interested reader will find screenshots of the application Appendix D on page 147. The second horizontal ribbon (entitled ‘Answering System’ on figure 3.1 on the facing page) corresponds to the main processing of the query (sent by front-ends applications) to underlying search engines. The lower ribbon corresponds to the different search engines involved for retrieving answers from different data sources integrated to the system.

3.1.1 Authentication

We have identified three security requirements in corporate environments that we consider in our proposal:

- users authenticate to the platform, possibly using a single-sign-on technology
- each user accesses a different subgraph of the data model; and usually cannot access the entire schema
- the token given to the user for authentication purposes is associated to a timeout; in which case user must authenticate again in case of inactivity

Authenticating on the platform consists in filling the pair ‘user name/password’ that must be specified on the front-end. Users who want to get personalized results must log on the system. Not-authenticated users can however benefit from generic results, when the underlying plug-ins do not explicitly require users to be authenticated. Thus, some search results are common to all users while some are specific to authenticated users (according to their profile and/or situation as detailed later).

Social network of users

Users have properties that are of interest for offering accurate personalized results. Such properties can be for instance their job title, their work location (country and city), their branch etc. For example, users of a search client on a mobile device (e.g. iPhone™) provide queries of minimum length, but expect it to be augmented with contextual information, like their geo-location from which the city they live in can be inferred. These information can be provided by corporate directory services, for instance via the LDAP² exchange format.

¹We will define these features in chapter 4 on page 75.

²Lightweight Directory Access Protocol

Users are commonly represented in a graph called *social network*, where nodes represent users and edges the relationships that they share. These relationships can be either hierarchical (for instance the ‘reports-to’ relationship) or not (like ‘has-business-contact’). Figure 3.2 provides a snapshot of a social

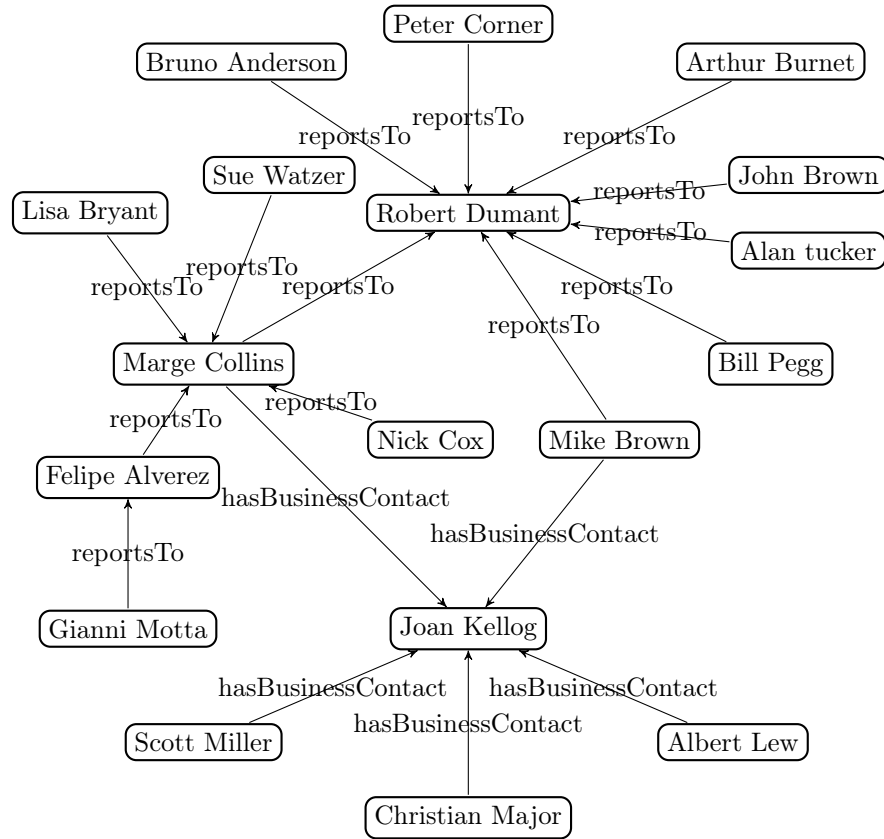


Figure 3.2 – Snapshot of corporate social network of 18 users sharing two types of relations: ‘reportsTo’ and ‘hasBusinessContact’

network of some corporate users. This kind of representation allows to compute easily distance metrics between users. Examples of these kinds of metrics will be given in section 6.3 on page 131 in the context of auto-completion.

Integration of corporate single-sign-on technologies

In corporate settings, single sign-on is a technology that monitors access control to independant software systems (i.e. applications) that users are allowed to access. The two major properties of this technology are:

- users have to log in once, and are not asked again for their credentials when accessing other applications that belong to the corporate network
- when users log off from any application, they are logged off from all applications that use single sign-on

The front-end applications that have been developed in the context of this work use single sign-on. This ensures that any corporate user has access to the Q&A system, and is able to experiment it and eventually provide useful feedback.

3.1.2 Search & prediction services

The *search app* component (see figure 3.3) is responsible for sending users' requests to the back-end system, and feedback to the user (e.g. error codes and messages). As displayed on the figure, the front-end tools send information

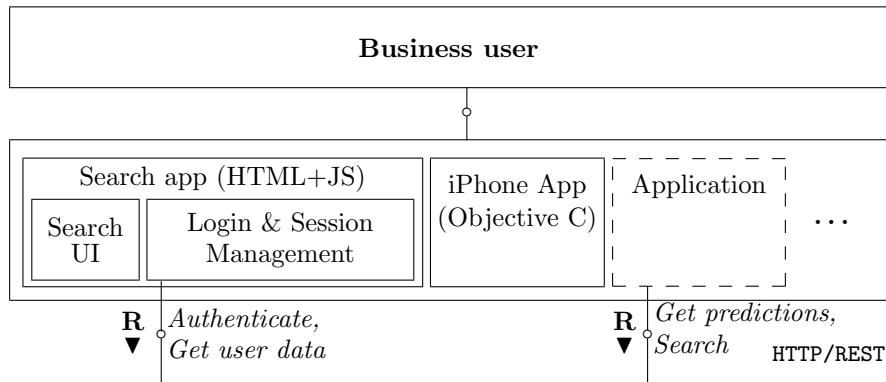


Figure 3.3 – HTTP/REST communication between the back-end and front-end systems

about context (e.g. which kind of device is being used, the geo-location of the user, etc.). The front-end application displays some of the error messages that are caught in the back-end or front-end system. The *prediction service* introduced here aims at providing auto-completion of users' queries (e.g. propose 'revenue' when user has typed 're'). The prediction mechanism that we have implemented in this context is detailed section 6.3 on page 131 and in [65].

Users' questions

Users' questions are expressed in NL. We decided not to restrict the way users formulate queries (i.e. the syntax of *valid* queries), because we assume that the linguistic coverage of the system will improve over time, based on query logs that will be collected by system administrators and possibly thanks to machine learning techniques (on-going work).

Speech-to-text

The mobile front-end applications (iPhone™ and iPad™) are composed of a speech-to-text component. Currently, the system supports three european languages (English, German and French) and we do expect even more in a near future. These applications have been implemented in the context of the project 'Search360' at SAP Research.

3.1.3 Answering system

This section describes the main component of the Q&A framework. It aims at analyzing and processing users' input in order to retrieve pieces of information of interest. Figure 3.4 depicts the architecture of the core answering system. It is composed of two services: *user context service* and *search service*. The

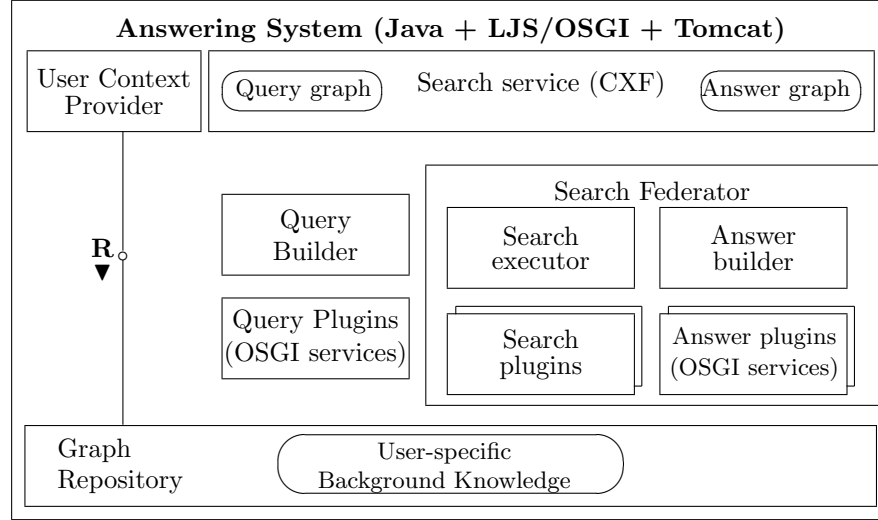


Figure 3.4 – Architecture of the answering system

former service provides information about the user (i.e. information about her profile, properties about which front-end application she is currently using, which device has been used, etc.). These information can then be directly accessed by search plugins, to offer personalized results. The latter service is triggered by search requests initiated by front-end applications.

Question analysis, as well as results retrieved by search engines (see section 3.1.4 on page 67) are stored in two data structures: *query graph* and *answer graph*. Both query tree and answer tree are graph structures. The query graph, in particular, can be compared to a *parse tree* in Q&A systems.

In the next section, we further introduce the parse tree, and present the different *plugins* involved in providing annotations to this graph structure.

Parse tree & plugins

In Q&A systems, the parse tree is a graph structure that stores linguistic information about user's question (e.g. part-of-speech tags, words stems, ...). Here, we use the term 'query graph' since it's less generic and targets the generation of structured queries (e.g. MDX or SQL). The query graph is a common tree data structure for storing *annotations*, i.e. information resulting from the analysis of users' questions. The annotations are created by different annotators called *query plugins*. We present below different query plugins that have been implemented.

To support more sophisticated queries (e.g. range queries or queries with orderings) and go beyond keyword-like questions, we have to allow adminisitra-

tors to define custom vocabulary (such as ‘middle-aged’) and more complex linguistic patterns, where ‘top 5’ is a simple example. Such artifacts may export variables, such as the beginning and end of the age range in the case of ‘middle aged’ or the number of objects to be shown to the user in the case of ‘top 5’. We will discuss further our solution for defining linguistic patterns in chapter 4 on page 75.

Information Extraction plugins These plugins are named ‘Query plugins’ on figure 3.4 on the facing page. These plugins are the first ones in the pipeline that analyze the user’s question by applying information extraction components. These components contribute to a common data structure, the so-called query tree. By default, the system is equipped with three types of information extraction plugins that can be instantiated for different data sources or configurations:

- plugins for matching artifacts of the data source’s metadata within the query
- plugins for recognizing data values (directly executed inside the underlying database)
- plugins for applying natural language patterns (e.g. for range queries)

These plugins jointly capture lower-level semantic information to interpret a user’s question that can be interpreted in the context of the data warehouse metadata in subsequent processing steps.

These basic plugins are part of the framework (by default). They are intended to be general-purpose query annotators for standard *entities* (in the sense of named entities). These entities can be of different type (see basic examples of such entities in table 3.1). We use a standard information extrac-

Entity type	Example
numeric	123
date	June, 1st 2012
country	Germany

Table 3.1 – Examples of named entity types used by the basic query plugin

tion system (SAP BusinessObjects Text Analysis™, a successor of the system presented in [35]) with a custom scoring function. As scoring function for evaluating individual matches we adapted the scoring that was presented in [9]. In a nutshell, it combines TF-IDF³-like metrics with Levenshtein⁴ and punishes in addition matches where the length of a term in the metadata is much longer than the string occurring in the users’ question. A threshold on the score limits the number of matches that are considered for further processing.

We discuss in section 3.2 on page 68 how the standard query plugin can be efficiently configured for application-specific settings.

³Term Frequency-Inverse Document Frequency, see <http://fr.wikipedia.org/wiki/TF-IDF>.

⁴See http://en.wikipedia.org/wiki/Levenshtein_distance.

1. Data schema query plugin The data schema query plugin annotates words and phrases that correspond to entities belonging to the data schema of a data warehouse. These entities are indexed in a dictionary (or lexicon) in an efficient way, and are compiled by the Named entity recognizer (NER) such that they can be efficiently recognized at query time. This process (indexing and named entity compilation) is performed when the system starts-up and when a change in the data schema is observed. Besides, it is possible to specify variants for entities being recognized and to define confidence metrics for each entity from the query input. These confidence values are computed by default with the distance mentioned above, but it is still possible to define custom confidence values before compiling the dictionary. An example that demonstrates the importance of such metrics and that shows how we use these metrics is presented in section 4.3.1 on page 88.

2. Natural language query plugin This component annotates expressions (which are language-specific) in users' questions. The created annotations are supposed to be independant from the application domain, but can be configured for a specific application. In particular, it is well-known that various expressions refer to the same meaning (and also that the same expression can be interpreted differently based on the context where it appears). For example, the expression "best country" is interpreted (in BI applications) as the selection of the first member of the dimension "Country", where members are ordered basing on a measure (which can be explicit in the query). Table 3.1 on the preceding page lists some examples of expressions grouped by a type called *feature*.

Feature	Example
top- k	"best country"
	"2 least cities selling x "
range	"between x and y "
	"more than x "
	"before January"

Table 3.2 – Examples of NL features that are annotated

The actual implementation of this plugin is not described here, but further described in section 4.2.3 on page 79.

3. Background knowledge query plugin The system is composed of a background knowledge base intended to be domain-independant, which can be configured and specialized for the specific applications. As an example, knowing the data type of dimensions is much important when rendering charts, and this kind of knowledge can be easily stored in the knowledge base. Figure 3.5 on the facing page is an example of chart⁵ that has been rendered without any knowledge about data type of the main dimension (i.e. 'Invoice date'). Indeed, the x -axis is a time dimension (dimension 'Invoice date'). In that case, the convention is to order the members along this dimension (while in general, for non time or numeric related dimensions the order is based on the values of the plotted measure, like in this example).

⁵ Charts rendered in the system are further described in section 3.1.3 on page 66.

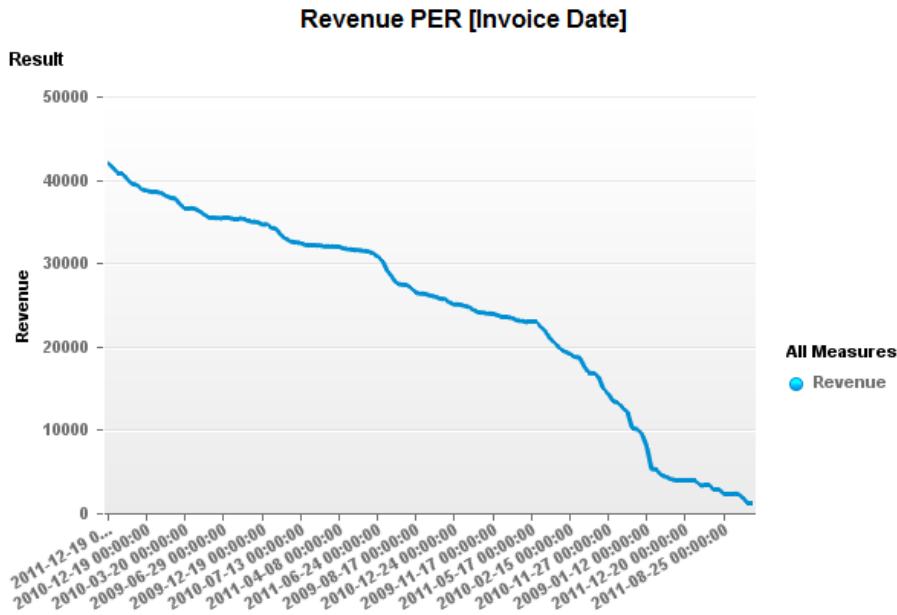


Figure 3.5 – Example of chart rendered without any knowledge on dimensions' data types

Interpreting the semantics of dimensions is however not obvious. In general, in the case of dates, there are many different time formatting templates (e.g. YYYY-MM-dd or YYYY, dd MM). Depending on which template has been adopted, the order of members on the axis would be different. Similarly, there is no single way to refer to members. For instance, the values of the dimension 'Quarter' might be in the form 'Qx' (e.g. 'Q1', 'Q2', ...) or of the form 'x' (e.g. '1', '2', ...). The BI community⁶ has established some guidelines or good practices for rendering charts. In our experiments (see chapter 6 on page 119), we have observed following guidelines in order of importance:

1. the chart type should represent the desired analysis type (e.g. a pie chart is preferred for analyzing contributions while bar charts are better for comparing them). The notion of *chart preference* will be defined section 5.2.3 on page 107.
2. time-related dimensions should be on the *x*-axis or in the legend area (but never on the *y*-axis). Besides, series should be ordered based on time order (and not based on the values of the plotted measure).
3. when a legend is required (e.g. when there are more dimensions than the chosen chart type can render), the number of members of the series appearing in the legend should be as small as possible

The similar problem arises for geographic data. Indeed, in that case it is better to represent measures' values on a geo-map (representing the values of

⁶UK national statistics department has published some of these guidelines at <http://www.neighbourhood.statistics.gov.uk>.

the geographic dimension) instead of ordering geographic members basing on their lexico-graphic order.

Nevertheless, a *standard form* of the dataset must be used in order to compare results. In our work, this standard form is defined as follows:

- measures are ordered given the lexico-graphic order
- dimensions are ordered given the same order
- dimensions' values are ordered given the lexico-graphic order of their members, regardless of the data type of the corresponding dimension (as the chart displayed figure 3.5 on the preceding page)

Answer tree

The answer tree is a graph data structure that contains different *answers* of different types corresponding to the input question. The answer types that we consider are enumerated below.

Charts A *chart* is a chosen visualization for a given dataset. It aims at better describing the data, emphasizing graphically areas of interest in the data. The same dataset can be rendered in different ways (e.g. as a bar chart, a pie chart, etc.).

Figure 3.6 on the next page represents two ways of representing graphically the same dataset. The first visualization (a) is a *bar chart* while the second one (b) is a *pie chart*.

Tables *Tables* are a flat representation of the dataset. In BI, it often takes the form of a subset or of an aggregation of (a) fact table(s). For instance, the

Fact	Title	Sales
f_1	Pacific Sales Manager	383200\$
f_2	European Sales Manager	1124400\$
f_3	North American Sales Manager	1660050\$

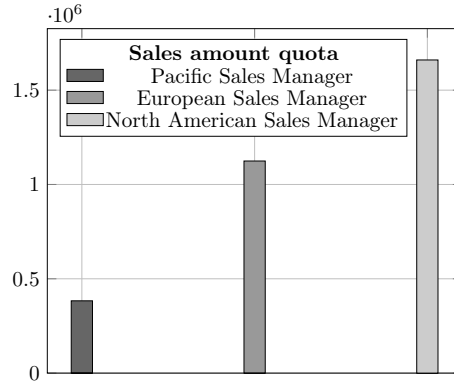
Table 3.3 – Facts answering the question “Sales target per department in 2001”

facts represented in table 3.3 correspond to the dataset which visualization is depicted on figure 3.6 on the next page. In this example, data come from a relational database, and the real attribute and table names have been renamed to the dimension and measure names used in the domain application.

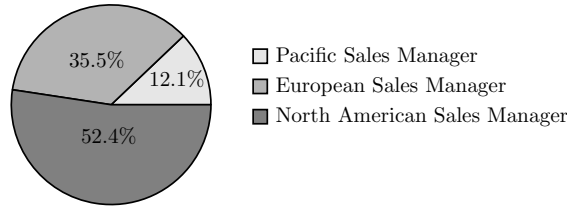
A table can also be a *cross-table*, in which case the first column corresponds to a dimension (like the dimension “Title” in Table 3.3). We have reproduced an example of such a cross-table table D.1 on page 148 .

Web results *Web results* are pieces of Web pages that should be relevant with respect to users' queries. Web results can be:

- unstructured documents (raw text)
- semi-structured content (e.g. Wikipedia pages)



(a) Bar chart visualization



(b) Pie chart visualization

Figure 3.6 – Two visualization types corresponding to the same dataset

- structured content (e.g. Freebase, WolramAlpha, ...)

Searching for documents on the Web and offering relevant documents or pieces of documents is a challenging task, merely because of the large amount of available resources. Intuitively, users' queries should be modified based on trusted information (coming from the different knowledge bases bound to the system). This seems to promise more accurate and personalized results than querying the different Web services with users' inputs. The corresponding component is part of the system and has been implemented, but is not of the scope of this thesis. Therefore we do not focus on it, and its implementation is not further described there.

3.1.4 Search engines

Search plugins operate from *annotations* held by the query tree and generated by the different information extraction plugins. Search plugins intend to execute queries on back-end systems, which could be indeed traditional search engines in the context of the federated search. They use recognized semantics from the query tree and formulate arbitrary types of queries. In the case of a traditional search engine, a plugin might take the users' question, rewrite the question using the recognized semantics to achieve higher recall or precision. However, for Q&A we define an abstract implementation of a plugin that is

equipped with algorithms to transform the semantics captured in the query tree to a structured query.

The output of a search plugin is a stream of objects representing a well-structured result together with its metadata (e.g. consisting of the datasource that was used to retrieve the object or a score computed inside the plugin).

Search engines are components that actually retrieve documents or pieces of documents from a corporate repository or a public data source (i.e. the Web). Figure 3.7 is an illustration of different search engines used by default. Components with dotted lines are examples of new component that can be implemented in addition but that are not necessary at the time.

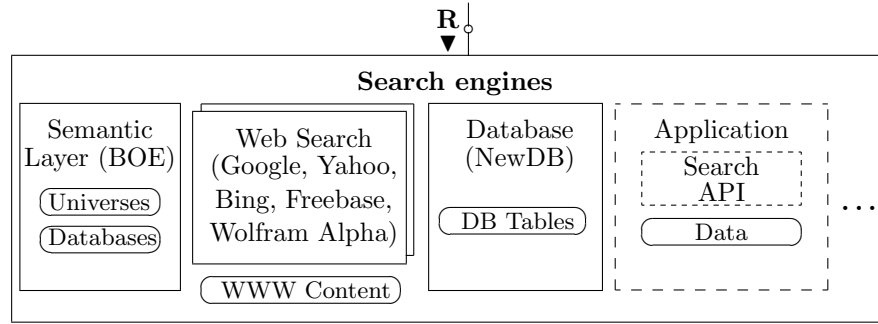


Figure 3.7 – Search engines implemented as plug-ins

In our work, we focus on the second component entitled ‘Content server’ which is a corporate document repository. It contains data schemas of data warehouses (called *universes* in the implementation of the BI system that we use) from which we generate database queries. The mapping from the query tree (see section 3.1.3 on page 62) to answers (stored in a structure called answer tree) is performed through *patterns* which will be defined in chapter 4 on page 75. These patterns capture the semantics of users’ questions, and generate database queries accordingly.

3.2 Extensibility of the framework

The system is configured and set up for standard settings, but can be configured for improving its overall performance in different environments. We present first how to implement new plugins in the framework in section 3.2.1. Then, we describe how to configure the linguistic resources that are part of the system in section 3.2.2 on the facing page.

3.2.1 Implementing new plugins

The plugins that have been introduced so far (see section 3.1 on page 59) can be specialized for application-specific settings. Besides, additional plugins can be easily implemented. In this section, we provide information on how such extensions can be made.

Query plugins

Query plugins annotate the query tree. Any such plugin must implement the interface reproduced on Listing 3.1

```

1  IParsingQueryPlugin {
2      public void annotate(
3          QueryTree queryTree, SessionContext sessionContext);
4  }
```

Listing 3.1 – Interface of any query plugin

As shown above, the method `annotate` takes as argument `sessionContext` that corresponds to available contextual information about the user.

Search plugins

As introduced above, search plugins take as input not only users' tokenized question, but the graph structure where the question has already been analyzed (see listing 3.2).

```

1  ISearchEnginePlugin {
2      public ResultIterator search(
3          QueryTree queryTree,
4          SessionContext sessionContext);
5  }
```

Listing 3.2 – Interface of any search plugin

As reproduced on the listing, the only required method is the `search` method which takes as argument the query tree and the session context of the user currently logged on (`sessionContext`).

Answer plugin

Answer plugins are responsible for rendering *answers* and *results*. As reproduced in listing 3.3, these kinds of plugins consist in augmenting an existing answer with an ordered set of visualizations corresponding to a result.

```

1  IAnswerPlugin {
2      public void augmentAnswer(Answer answer,
3          SessionContext sessionContext,
4          QueryTree queryTree);
5  }
```

Listing 3.3 – Interface of any answer plugin

3.2.2 Configuration of linguistic resources

Query plugins (see section 3.2.1) use lexicons and other linguistic resources to annotate users' questions. The system is composed of following linguistic resources:

- lexicons of the data schema
- domain-independant knowledge base
- parsing rules for natural language *features*

- contextual user model

We describe below how to configure the different resources below.

Configuration of the lexicons

Lexicons of the data warehouse schema are generated automatically. They are generated and indexed again when there are changes in the data models (this is checked when the system starts up). The location of the different data models is specified in a configuration file.

Configuration of domain-independant knowledge bases

The domain-independant knowledge base is composed of time and geographic knowledge. It is currently used to interpret semantically dimensions and dimensions' values of geographic and time-related type as pointed out section 3.1.3 on page 62. The domain-independant knowledge base is a triple repository (RDF) which statements must comply an RDF schema, which can be personalized (with new classes, attributes and predicates).

Natural language features

NL *features* (see section 4.2.3 on page 79) are defined as custom regular expressions which export property pairs. New parsing rules can be easily added to the system, and the syntax depends on the tools used for entity recognition. Custom features (or patterns, see section 3.1.3 on page 62 for a short description) are simply declared in the *feature definition* file.

```

1 feature:topk_feature rdf:type feature:GGUL_Feature ;
2   rdfs:label "top_k" ;
3   query:generatesAnnotationsOfType query:TokKAnnotation ;
4   feature:cgulExpression
5     "#group_topk{<top>|[0D_number]<[0-9]+>[/0D]"
6     ^^xsd:string .
```

Listing 3.4 – NL feature definition

The mechanism for NL pattern is very powerful. It is used to implement custom functionalities (e.g. range queries, top-*k* queries or custom vocabulary such as shown for “middle-aged” (see figure 4.1 on page 78) that goes beyond keyword-matching. As explained in the previous section, natural language patterns are configured using RDF (see listing 3.4 for an example). Mandatory properties are:

- **feature:topk_feature**: the kind of feature being defined. When defining new NL features, the object should always be **feature:GGUL_Feature**
- **rdfs:label**: the name given to the feature. There is no requirement of uniqueness
- **feature:cgulExpression**: the expression to be matched by the rule. This expression must be of type **xsd:string**, and the syntax depends on the NER software being used (see below)

The three main parts of a natural language patterns are:

1. **Extraction Rules:** The basis for natural language patterns are extraction rules. In our case we use the CGUL rule language⁷, which can be executed using SAP BusinessObjects Text AnalysisTM. It bases similarly as CPSL [4] or JAPE [13] on the idea of *cascading finite-state grammars* meaning that extraction rules can be built in a cascading way. Thus any other rule engine can be used for this purpose. We make heavy use of built-in primitives for part-of-speech tagging, regular expressions and the option to define and export variables (e.g. the ‘5’ in ‘top 5’). Note, that a rule might simply consist of a token or a phrase list, e.g. containing ‘middle-aged’.
2. **Transformation Scripts:** Once a rule fired, exported variables may require some post-processing, e.g. to transform ‘15,000’ or ‘15k’ into ‘15000’, an expression that can be used within a structured query. In many cases there is also the need to compute additional variables. The most simple case for such functionality is to output beginning and ending of the age range defined by a term such as ‘middle-aged’. To do additional computations and transformations, we allow to embed scripts inside a natural language pattern, which can consume output variables of the extraction rule and can define new variables as needed.
3. **Referenced Resources:** A rule is often specific for a resource in some metadata graph. For instance in figure 4.2 on page 81 the pattern for **AgeTerms** applies only to the dimension ‘Age’, the ‘Context’ pattern only to nodes within the user profile and other patterns apply only to certain data types (e.g. patterns for ranges to numerical dimension values) – which are also represented as nodes. In order to restrict the domain of patterns, we allow to specify referenced resources.

Configuration of contextual user model

There is not a single way of modeling users’ context in corporate environments: many information about the authenticated user on the network are usually stored in a LDAP⁸ system, but each application usually adds profile attributes that are not shared accross various applications.

In the answering system, we provide a basic profile model, where users are automatically logged on on the basis of the network authentication (see section 3.1.1 on page 59). Our model of users is currently composed of profile information, such as surname, last name, job title, location, etc. This can be configured in order to add new attributes and relations. We base on the corporate social network system that has been described in [63], which provides a so-called *social provider* which is loaded when the system starts up. In this system, the social network is represented as a graph (see figure 3.2 on page 60). Each user of the system is then associated to a *node* in the graph (**userNode**),

⁷http://help.sap.com/businessobject/product_guides/boexir4/en/sbo401_ds_tdp_ext_cust_en.pdf

⁸Lightweight Directory Access Protocol, see http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

and further configuration can be provided monitoring what attributes shall be loaded in the graph.

3.3 Multithreading and performance considerations

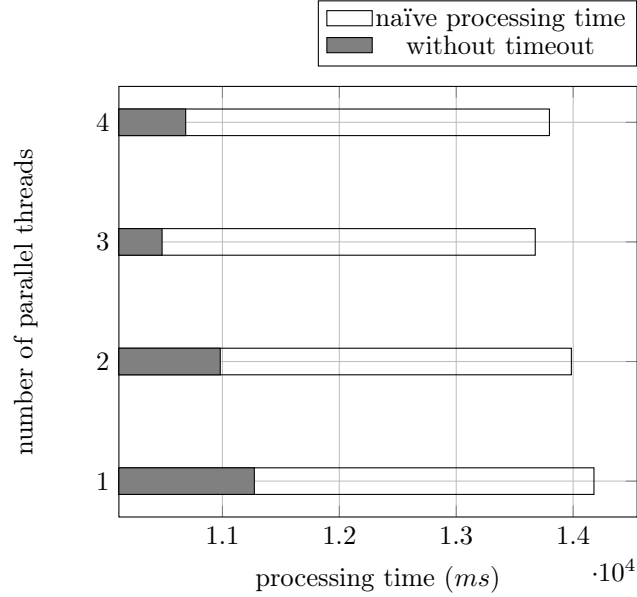


Figure 3.8 – Average overall processing time depending on the number of threads being executed in parallel

The first implementation of the system described so far suffered from the lack of parallel algorithms, and therefore its performance was not satisfactory. We have reported on figure 3.8 the mean execution time (for the same set of queries) for different number of parallel threads. Performance tests have been lead on a 64-bit operating system with 8Gb RAM memory and Intel Core2 Duo processor. The second series (see legend “without timeout”) corresponds to the results of a subset of queries only (the ones that do not generate a timeout signal). Indeed, the timeout is sent by the back-end system in order to prevent the system from being stucked for queries which execution takes too long. Parallel programming does not improve processing time for those queries. As expected, it seems that we gain in performance when increasing the number of cores. The overall processing time remains important, mainly because of the chart generation process which optimization was not the purpose of our work.

We have introduced parallel algorithms in the following components of the answering framework:

- pattern execution
- (object) query generation
- (database) query generation
- query execution

The interested reader will refer to appendix E on page 151 for more information on the actual implementation of the component responsible for pattern execution.

3.4 Conclusion

3.4.1 Summary & discussion

We have presented the architecture of the answering framework. It behaves as a corporate system, where users are well identified on the network, and have profiles of two kinds (shared properties that can be reached from any corporate application, and private properties specific to each application). End-users access the system through the HTTP protocol. So far, three front-end applications have been implemented: a desktop application (in HTML 5), an iPhone and iPad applications (these front-ends will be introduced in chapter 6 on page 119). Information about users (“User Context”), data models of the data warehouses and various domain- and language-specific configurations are aggregated in a graph structure. Then query plugins operate on this graph, and the output (answer graphs) are then processed by search engines to render results (e.g. to execute SQL or MDX queries and to render corresponding charts). In addition, the framework is extensible. Configuration can be easily performed to take into account new domains (the system interfaces several data warehouses) and/or new languages. Configuring additional query plugins is the right way to go in the scope of context-aware applications. Indeed other corporate applications provide meaningful information in some cases, and the system would thus provide more personalized results. Search plugins would be used for instance to support additional structured query languages (in addition to SQL or MDX), like SPARQL. In chapter 6 on page 119 dedicated to experiments and evaluation, we will show how easy is the configuration of the system for additional domains. Moreover, we have implemented the system using parallel algorithms, which has improved a lot the performances of the system.

Compared to state-of-the-art systems in the BI domain (e.g. SODA [8]), we provide more personalized results because the framework takes also into consideration contextual information (from the “user context” component). In addition, the processing pipeline of our system is similar to some extent to some recent systems, like [67]. However, our system is not yet perfect, and we suggest several directions to improve it. First, the execution time of the system can be reduced, even if we have investigated parallel algorithms to make it optimal. Currently, new questions (i.e. questions that have never been asked before) are being answered in a few seconds. We believe that this is due to the call to some native libraries (the NER), plus the fact that the constraint mapping engine (Jena executing SPARQL over RDF) is not optimized. A solution would be to re-implement the constraint mapping algorithms in such a way that it would be executed in-memory.

3.4.2 Organization of the next chapters

In chapter 4 on page 75 we detail the technique used to translate NL queries in database queries, i.e. *patterns*. Then, we present the conceptual query model

that is used in our system to represent multidimensional queries in chapter 5 on page 97. The overall answering system is evaluated in chapter 6 on page 119.

Chapter 4

Linguistic patterns

Outline

4.1	Linguistic patterns in IR	76
4.2	Patterns for structured information	77
4.2.1	Running example	77
4.2.2	Users' graphs and annotations	79
4.2.3	Feature	79
4.2.4	Parse graph	82
4.2.5	Pattern	83
4.2.6	Structured queries	87
4.2.7	Query logs	88
4.3	Ranking the results	88
4.3.1	Confidence	88
4.3.2	Selectivity	88
4.3.3	Complexity	88
4.3.4	Metrics from query logs	89
4.3.5	Overall measure	91
4.4	Summary & discussion	91
4.4.1	Learning approaches	92
4.4.2	Authoring tool	95

Linguistic patterns are widely used in Information Extraction (IE) and Information Retrieval (IR). In IE, patterns are used in unstructured documents (text corpora, Web documents) to extract information basing on the structure (syntactic information) and on the terms (lexical information) identified in the text. More generally, the idea is that the same piece of information can be expressed in various ways, and that a pattern (i.e. a set of constraints) captures the common characteristics of those different expressions of the same idea. This is depicted in table 4.1 on the next page. The strong assumption there is that given a sentence, it is possible to find a set of constraints that can define the meaning of the sentence, or the information that a user is looking for. It is obviously not true in general, because there is not a single interpretation of NL sentence inputs, but constraints in the patterns help in limiting the ambiguity of the textual input.

Question	Entities
Sales revenue per year	(Sales revenue) [Year]
Revenue over years	(Sales revenue) [Year]
Sales results per FY	(Sales revenue) [Year]

Table 4.1 – The same idea can be represented in different ways. The second column expresses constraints that are satisfied by all questions from the first column

4.1 Linguistic patterns in IR

A pattern (or part of it) is usually defined and associated with the notion of syntactic axis. Indeed, *pattern* in this field can be referred to as *lexico-syntactic patterns* (terms and syntax) or *morpho-syntactic patterns* (terms and their categories plus syntax). Despite the fact that they are extensively used, there are very few comments on the definition of such patterns. They have been defined in the linguistic theory [32] as “a schematic representation like a mathematical formula using terms or symbols to indicate categories that can be filled by specific morphemes”. Patterns used by Sneiders [59] are regular strings of characters where sets of successive tokens are replaced by entity slots (to be filled by corresponding terms in the actual textual document). This means that a pattern in this sense is an extension of regular expressions (where *patterns* from the regular expression are *slots* that may be of various type in linguistic patterns). An innovation in [60] is the definition of a pattern being composed of two subpatterns: one *required pattern* (regular patterns) and one *forbidden pattern*, that corresponds to pattern that must not match the message. Finkelstein-Landau et Morin [18] formally define morpho-syntactic patterns related to their IE task: they aim at extracting semantic relationships from textual documents. A pattern A can be decomposed as:

$$A = A_1 \dots A_i \dots A_j \dots A_n \quad (4.1)$$

In this formula, A_k $k \in [1, n]$ denotes an *item* of the pattern A which is a part of a text (without any *a priori* constraint on the sentence boundaries). An *item* is defined as an ordered set of *tokens* composing words¹. In this approach the syntactic isomorphy hypothesis is adopted. Let $B = B_1 \dots B_i \dots B_j \dots B_n$ be a pattern. This hypothesis states the following assertion [40]:

$$\left. \begin{array}{l} \exists(i, j) \quad \text{win}(A_1, \dots, A_{i-1}) = \text{win}(B_1, \dots, B_{j-1}) \\ \text{win}(A_{i+1}, \dots, A_n) = \text{win}(B_{j+1}, \dots, B_n) \end{array} \right\} \implies A_i \sim B_j \quad (4.2)$$

which means that if two patterns A and B are *equivalent* (they denote the same string of characters), and if it is possible to split both patterns in identical *windows* composed of the same tokens (when applied to string of characters), then the remaining items of both patterns (i.e. A_i and B_j) share the same syntactic function.

¹Delimiting tokens is not an easy task in any language, since a word might be composed of several tokens, and in some languages words boundaries are not obvious.

The introduction of *windows* in the definition of patterns imposes their components to be ordered given the syntactic order (for instance left to right in English). The kind of patterns presented here has become very popular, in particular among the SW community, for instance through Ontology Design Patterns², which is a platform meant to define classes of patterns widely used in IE.

In the following, we adopt an other formalism, where the syntactic isomorphism is not kept, i.e. the expected characteristics (or *features*) from the patterns do not have to be ordered along the syntactic axis in the sentence (or the question) matched by the pattern.

4.2 Patterns for structured information

Patterns for structured content consist of structures that define a mapping between text and a data structure (like a database). In our proposal, we relax the syntactic constraint³ and wanted to have any kind of features to appear in the pattern. This mapping is performed in several tasks:

- declaring the constraints that must be satisfied for the pattern to match the question
- defining which part of the question must be exported according to the constraint defined in the previous step
- defining what to recompute from the exported information in the question
- defining the actual mapping from the exported information to the data structure

We will detail these tasks in the following.

4.2.1 Running example

In the following, we will consider as an example the following user's query:

$$\text{"Top 5 middle-aged customers in my city"} \quad (4.3)$$

which is also depicted figure 4.1 on the following page.

Question

Let $Q = \{q_1, \dots, q_n\}$ be a user's question and q_i $i \in [0, n]$ tokens of the question. For example, the question (4.3) can be represented as follows:

$$Q = \{\text{'Top'}, \text{'5'}, \text{'middle-aged'}, \text{'customers'}, \text{'in'}, \text{'my'}, \text{'city'}\} \quad (4.4)$$

Note that the word 'middle-aged' can also be decomposed in two words (i.e. $\{\text{'middle'}, \text{'age'}\} \subset Q'$). In the following, we use the term *query* to denote a database query and the term *question* to denote a user's query.

²See http://ontologydesignpatterns.org/wiki/Main_Page.

³The usefulness of this approach is also motivated by our iPhone™ application, where the generated question is more a keyword query rather than a well-formed NL question.



Figure 4.1 – Translating a user's question into a structured query. NL patterns, constraints given by the data and metadata of the data warehouse (see figure 1.4 on page 6) have been applied to infer query semantics. This was mapped to a logical, multi-dimensional query, which in turn was translated to SQL. Note that the 'revenue' represents a proposed measure, depicted as '?1' in figure 4.1a. The computation of the measure 'revenue' and the join paths are configured in the metadata of the warehouse.

4.2.2 Users' graphs and annotations

The different information held by questions' annotations as well as additional information about users (e.g. contextual information) are stored in a *graph repository*. The framework incorporates a situation manager described in [63]. This framework is used to capture and monitor different kinds of events of context-aware corporate applications. A use-case of the framework is the modeling of the situation of users. This modeling was initially used for offering personalized recommendations. An other use-case that we propose is to provide personalized search results. The situation framework brings a so-called graph repository. Every user can access a set of personalized graphs stored in this repository. The graphs are created by *providers* that decide who is allowed to access what information. We have implemented one provider for each origin of annotations. For instance, all annotations corresponding to data warehouse entities come from a graph (in the graph repository) that has been generated by a specific provider, the one dedicated to data models. At run-time, all graphs belonging to the user (i.e. containing information about the user only, and information that user is allowed to see) are used together with the parse graph (see above), and used as a model (in the sense of RDF) when executing patterns. In nutshell, the graph of the parse graph contains annotation about the user's query, and the other graphs (i.e. the graphs coming from the user's graph repository) define all entities that have been referenced in the parse graph.

So far, the graph repository is composed of the following graphs:

- the graph composed of textual annotations from the current question (i.e. the parse graph, see section 4.2.4 on page 82)
- the graph composed of annotations about the physical device being used and other contextual information like the geo-location of the user

Besides, some user-independant graphs are also part of the repository:

- the graph composed of domain-specific knowledge
- the graph composed of language-dependant linguistic knowledge

The latter graph has been detailed in section 3.2.2 on page 69.

An *annotation* is defined as a rooted graph which root is the set of tokens (possibly empty) from the user's question involved in the annotation. It refers to a node of type **Annotation** belonging to set of personalized graphs. We note $a \in A$ such an annotation and A the space of annotations. Note that an annotation is not always bound to actual tokens of Q (for instance, the annotation related to the physical device is not bound to any token from Q).

4.2.3 Feature

A feature $f_i : Q \rightarrow A \in \mathcal{F}$ is a mapping from the query to the set of annotations for the feature i . The set of all annotations given a query Q for the feature i is given by $f_i(Q)$. In the case where no annotation $a \in f_i(Q)$ is bound to the query Q , a is independant from Q (but is still user-dependant). The number of annotations in the query Q for the feature i is given by $|f_i(Q)|$. For example,

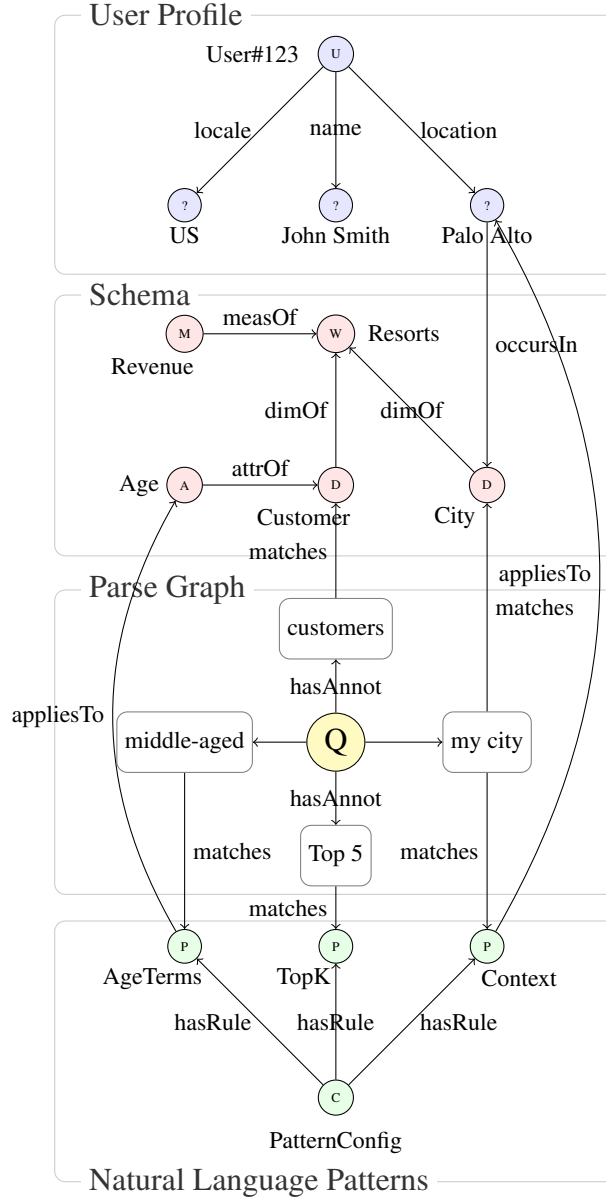


Figure 4.2 – Example of the parse graph generated for the question “Top 5 middle-aged customers in my city”. The upper-box (‘User Profile’) corresponds to the feature f_8 ; the box ‘Schema’ to the feature f_6 . The lower-box corresponds to a class of features to which feature f_5 belongs.

let the first feature be: $f_1 : t \mapsto t$, i.e. the *id* mapping and the second feature be: $f_2 : t \mapsto t'$ where t' is the base form of t , i.e. the mapping from terms to their base forms. We have thus:

$$f_1(Q) = id(Q) = Q \quad (4.5)$$

and:

$$f_2(Q) = \{t' \mid t \in Q\} \quad (4.6)$$

A feature can also map to more complex structures (a tree structure). Let f_3 be the *range* feature and f_4 be the *top-k* feature. An illustration of the latter feature was given figure 4.1 on page 78. The *range* feature (i.e. feature f_3) is a domain-specific rule (which belongs to the range of *custom rules*, see figure 4.1a on page 78) which exports the following information:

- [dimension]: the related dimension
- [begin]: the beginning value of the exported segment of text
- [end]: the ending value of the exported segment of text

The *top-k* feature (i.e. feature f_4) exports:

- [order]: what order will be used (i.e. ascending or descending)
- [nb]: the maximum number of results to be retrieved (i.e. the query modifier operator **LIMIT** in MDX)
- [dimension]: the related dimension
- [measure]: the related measure

Note that in the case of the *top-k* feature, the measure cannot be matched (1? in figure 4.1 on page 78), in which case a *valid* measure will be selected as explained later on. The exported items will then be rewritten or recomputed. For instance in the case of feature f_4 , ‘five’ from ‘Top five customers’ can be rewritten ‘5’ (so that it can be processed by the query generator), and ‘customers’ should be normalized as ‘customer’.

Let \mathcal{F} be the set of features considered in a given application. The above example (4.3) generates $|\mathcal{F}| = 4$ feature types and the features summarized in table 4.2.

Feature	Example	Ann. count
f_1 (<i>id</i>)	‘Top’; ‘5’; ‘middle-aged’;...	$ f_1(Q) = Q = 7$
f_2 (base forms)	‘top’; ‘five’; ‘middle’; ‘age’;...	$ f_2(Q) = 8$
f_3 (range rule)	\emptyset	$ f_3(Q) = 0$
f_4 (top- <i>k</i>)	“[top] [5] [customers] [?1]”	$ f_4(Q) = 1$
f_5 (custom rule)	“middle-aged”	$ f_5(Q) = 1$
f_6 (data model entities)	(Sales revenue) [Customer]; [City]...	$ f_6(Q) = x$
f_7 (geographic entities)	\emptyset	$ f_7(Q) = 0$
f_8 (user profile)	locale=“US” location=“Palo Alto”	$ f_8(Q) = y$

Table 4.2 – Features and annotations for the example (4.3 on page 77)

Note that all annotations do not have to refer to actual tokens in the user’s question. For instance f_7 in table 4.2 is independant from Q . When an annotation actually refers to tokens from Q , it keeps information about which tokens are involved (via properties like *offset* and *length*).

4.2.4 Parse graph

The *parse graph* of the query Q can be written:

$$p(Q) = \{f_1(Q), \dots, f_k(Q)\} \quad (4.7)$$

where f_i is the feature i . The parse graph of Q is the set of annotations for all features. The total number of annotations in the parse graph of Q is given by:

$$|p(Q)| = \sum_{i=1}^k |f_i(Q)| \quad (4.8)$$

The parse graph is implemented as an RDF graph. Each annotation $a \in f_i(Q)$ for the query Q are nodes of type **Annotation** in the RDF graph. The annotations are defined by a set of attributes and predicates (having the annotation as subject):

- **urn:grepo/query-tree#hasAnnotationType** defines the *type* of the annotation. Those types are sub-types of the **Annotation** type. They are summarized in table 4.3.
- **urn:grepo/query-tree#referencesResource** defines a reference to an external resource. For instance, database entities are defined in the data model of the data warehouse.
- **urn:grepo/query-tree#confidence** defines the *confidence* of the annotation, i.e. a score that measures how relevant is the entity with respect to user's query. The computation of this score is performed by the named entity recognition module.
- **http://www.w3.org/2000/01/rdf-schema#label** defines the *label* of the annotation, i.e. the *normalized* text that carries this annotation. The normalization process is further described in section 3 on page 57.
- **urn:grepo/query-tree#originUri** defines the vendor-specific ID of database entities.

Type	Uri
Dimension	DimensionAnnotationType
Measure	MeasureAnnotationType
Member	DimensionValueAnnotationType
NLP feature	NlpFeatureAnnotationType

Table 4.3 – Annotation types in the parse graph

Other predicates and attributes are defined in the case of NLP features, for instance for describing the kind of NLP feature, what item can be exported, etc.

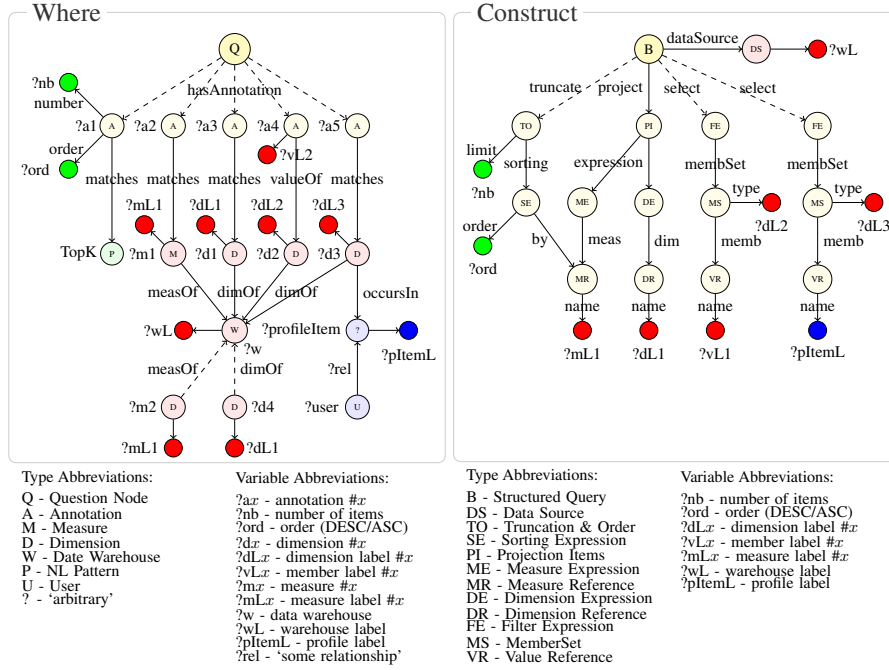


Figure 4.3 – Example for parse graph constraints and mapping rules to generate a structured query

4.2.5 Pattern

We present an example pattern in the appendix, section A on page 139. A *pattern* is the technique used to translate questions in structured queries. It is defined as a set of possibly optional *constraints* to be satisfied by the graphs, plus rules that define how these constraints are translated in structured queries composed of slots to be filled with actual data. Figure 4.3 is an illustration of a pattern in two parts. The left-hand side (entitled ‘Where’) represents the set of constraints that the different graphs must satisfy (i.e. the expected form of the parse graph). The pattern triggers only if the constraints defined there are satisfied. Some constraints defined here can be optional (as it will be explained in section 4.2.5 on the following page). The right-hand side (entitled ‘Construct’) represents the template of the structured query⁴ that will be generated.

Once the parse graph is created for a particular user’s question, the system has to ensure domain- and application-specific constraints. In addition to constraints mentioned on figure 4.1 on page 78 dedicated to BI use-cases, other apply on how entities occur together in the query, the data types of the recognized entities or to which other entities they relate to. Another constraint takes the form of entity recommendation when the user did not include all necessary information to compute a valid query or to add additional filter for personalization. We discuss in the following two types of constraints:

1. *relational constraints* (see section 4.2.5 on the next page) which describe




⁴Note that the syntax of the generated structured query will be introduced in chapter 5 on page 97.

situations like the fact that a dimension and a measure should belong to the same data warehouse

2. *property constraints*, which filter nodes based on property values (see section 4.2.5 on page 86), like the fact that two annotations are overlapping or close to each other.

Then, we discuss a convenient feature of SPARQL to inject additional variables (see section 4.2.5 on page 86), e.g. to generate additional values to be used in the structured query if a certain graph pattern occurs.

Relational Constraints

SPARQL queries are essentially graph patterns. Nodes and edges are addressed by URIs⁵ or are assigned to variables which are bound to URIs or literals by the SPARQL *query processor*. This mechanism of expressing graph constraints and of binding variables eases the configuration of our approach tremendously. Figure 4.3 on the preceding page is a visualized example of complex constraints for selection and mapping rules that are used in our application setting. On the left-hand side stands an excerpt of the constraints and variables used in our BI use-case. The markers attached to a node represent in contrast to figure 4.2 on page 80 assigned variables or URIs. URIs are expressed in short form and do not carry a leading question mark. Edges between nodes and literals refer to `rdfs:label` if they are not marked otherwise. Dashed lines illustrate that a particular part of the graph pattern is optional (implemented through an `OPTIONAL` statement in SPARQL).  depicts the user's question. Below are annotation nodes and left to them assigned variable names (like '?a1') which form the *parse graph*. Other nodes reference metadata graphs (see figure 4.2 on page 80). The nodes like  in figure 4.3 on the previous page represent resources, while nodes like  represent literals, which will be reused later for query composition. As discussed in next section, we map only literal variables to the final query model, to separate the input and output model on conceptual level. Note also that we define much more variables in the real-world use-case, e.g. to handover data types and scores from the question to the query model, which we leave out of the examples for sake of brevity. Our example exhibits constraints for the following situations:

Natural Language Patterns ('?a1') The annotation ('?a1') addresses the natural language pattern for the top-*k* feature (see figure 4.2 on page 80). The top-*k* pattern exports the variables 'number' and 'order', which are bound to '?nb' and '?ord'. This rule might be combined with a rule triggered by phrases like 'order by ...' to assign a variable holding the dimension or measure such that an ordering can be applied. In general, different natural language patterns can be easily combined using *property constraints* as explained in next subsection. Patterns for ranges, custom vocabulary, *my*-questions etc. are treated similarly. In particular in situations related to ranges and the mapping of other data types, we define additional variables for the data type of the of certain objects (e.g. 'Date' or 'Numeric') to handle them separately in the final

⁵Uniform Resource Identifier

query generation. These attributes eventually influence the *serialization* of the structured query.

Data Warehouse Metadata (‘?a2’ and ‘?a3’) The annotations ‘?a2’ and ‘?a3’ refer to a recognized measure and dimension bound via a *matches* relationship in figure 4.3 on page 83 and triggered by questions like “revenue per year”. An arbitrary number of measures and two dimensions are allowed (due to the requirement of rendering charts). By assigning the nodes for the measure (‘?m1’) and dimension (‘?d1’) to the same node for the data warehouse (‘?w’) we ensure that these objects can be used together in a structured query and lead therefore to a valid query. More precisely we check whether recognized dimensions and measures are linked through a fact table (see figure ?? on page ??). For reuse in the structured query, we assign the labels of the recognized objects to variables (i.e. ‘?mL1’ for the measure label and ‘?dL1’ for the dimension label). In some cases the system has to suggest fitting counterparts (e.g. *compatible* dimensions) to not aggregate all facts. In the example in figure ?? on page ?? we choose ‘?d4’ as dimension if the question contains only measures and ‘?m2’ as measure if it contains only dimensions. Thus the system generates multiple interpretations for the user’s question. The SPARQL blocks that contain ‘?d4’ and ‘?m2’ are optional and contain a filter (i.e. a *property constraint* as explained later) such that they are only triggered if either ‘?mL1’ or ‘?dL1’ are not bound. The label of the recommended measure or dimension is finally bound to the respective label variable that would otherwise be unbound (i.e. ‘?mL1’ or ‘?dL1’).

Data Warehouse Values (‘?a4’) Instead of the *matches* relationship, we use the URI *valueOf* to assign the dimension value to the corresponding dimension (i.e. ‘?d2’). For later reuse, we assign the label of the value’s node (‘?vL2’), e.g. ‘2009’ for a year, and the dimension value to a variable (‘?dL2’). In the real-world use-case we consider not only one match situation (like in the example) but a couple of other situations, where the declarative approach is very valuable. For instance, we show here only the case where the matched value does not belong to an already-recognized dimension (i.e. ‘d2’ would be an additional dimension in the query). For the situation where the value belongs to ‘?d1’ – an already-recognized dimensions – we define another optional SPARQL block which is triggered by the *valueOf* relationship between the annotation and the corresponding dimension. We treat single value matches for one dimension differently than matches on multiple values that belong to the same dimension. Our declarative approach eases this, because another set of constraints can be simply defined with separate variables.

Personalization (‘?a5’) Annotation ‘?a5’ shows the personalization feature, which applies a filter for a dimension if a corresponding value is part of the user profile (see ‘my city’ on figure 4.1 on page 78). The constraint captures following situation: an annotation (‘?a5’) refers to a dimension (‘?d3’) that *occursIn* some resource (‘?profileItem’) that has some relationship (‘?rel’) to the user (‘?user’). From the graph pattern, we consider for later processing the label of the dimension (‘?dL3’) and the label of the user profile that occurs in this dimension (‘?pItemL’). Note that constraints for personalization (as shown

in figure 4.3 on page 83) do not refer to the *my*-pattern (shown in figure 4.2 on page 80) due to space constraints. If the constraints would be applied as shown here, we would simply test for every matched dimension whether there is a value mapping to the user profile. These examples highlight the flexibility of using SPARQL graph patterns to manage constraints and variables for query composition in Q&A systems. Additional constraints have to be applied on property or literal level. They are detailed in the following sub-section.

Property Constraints

The following details the use of constraints through SPARQL `FILTER` statements considered in addition to graph patterns. They are less important on conceptual level, but have many practical implications, e.g. to not generate duplicated queries or to add further functionality, which cannot be expressed on graph pattern level. The first obvious additional constraint is to check whether two annotations matching two distinct dimensions are different:

```
FILTER(!sameTerm(?a1, ?a2))
```

It is often crucial to separate objects that matched the same part of the user's question into several structured query; this is even more important for dimension names because they define the aggregation level of the final result. This kind of constraints can be expressed using the metadata acquired during the matching process. Assuming that the position of a match inside the question has been assigned to the variable `?o1` and the offset and length of another annotation are assigned to `?o2` and `?l2`, the filter for ensuring that the latter one does not begin within the range of the first annotation can be expressed by:

```
FILTER(?o2 < ?o1 || ?o2 > (?o1 + ?l2))
```

Property constraints are also used for more complicated query generation problems. It would then apply to `dataType:Numeric`, be triggered by phrases like 'between *x* and *y*' and include a script for normalizing numbers. In combination with matched numeric dimension values, one can define a filter that tests whether two members were matched inside the matched range phrase and generate variables defining the beginning and ending of a structured range query.

Additional Variables

It is often useful to define default values or to bind additional variables. An example for a default value would be to limit the size of the query results if it is not specified by the user. To do so, we add an optional SPARQL block that checks the variable '`?nb`' and assigns a value if it is unbound using:

```
BIND(1000 AS ?nb)
```

There are plenty of other use-cases to inject additional variables, like defining analysis types (which are part of the not-illustrated metadata that is assigned to a structured query). These are indicators used to select the best fitting chart type for a single result. To capture the analysis type, we use certain trigger-words (see [64]) and additional constraints such as the number of measures and

dimensions and the cardinality of dimensions. For instance we would select a *pie chart* if a single measure and dimensions is mentioned in the question and the user is interested in a ‘comparison’ (e.g. triggered by the term ‘compare’ or ‘versus’). However, if the cardinality of the dimension (which is maintained in the metadata graph) would exceed a certain value (e.g. 10), a *bar chart* would be a better fit because a pie chart would be difficult to interpret otherwise.

As result of the mapping step, we get an RDF graph containing all potential interpretations (structured queries) of the user’s question. Since the query model as such reflects the features of the underlying query language L (e.g. projections and different types of selections) it is straightforward to serialize this model to an actual *string* that can be executed on a data source. The constraints defined in previous sections ensure on the one hand how to treat different match situations and on the other hand that the generated queries are valid. The great advantage of this approach is that complex constraints can be defined in a declarative way and that they are to some extend separated from the mapping problem, making the implementation much easier in presence of complex requirements. The generated structured queries must then be scored to provide a usefull ranking of results and to define an order according to which the computed queries are eventually executed.

Application domain

A pattern for Q is a chosen pattern t_Q matching the query Q and must satisfy:

$$t_Q \in \{t \mid t(Q) \neq \emptyset\} \quad (4.9)$$

Let $T = \{t_{Q_1}, \dots, t_{Q_n}\}$ be the set of patterns used in the context of a domain application. This demonstrates that the linguistic coverage of the system is finite (since T is a finite set). $|T|$ is thus a simple way of expressing how broad is the linguistic coverage of the system.

The implementation of the pattern encodes information on how the ranking should be computed. This ranking relies on a confidence score. This score is then incorporated in the query graph (defined in the first section) basing on information defined in the second section (the parse graph). The computation of this confidence score is presented section 4.3.1 on the next page.

4.2.6 Structured queries

We note $B(t, Q)$ the set of queries generated by pattern t for the question Q (see the right-hand side of figure 4.3 on page 83 and entitled ‘Construct’). The set of constraints defined in a pattern matching the query Q is defined as follows:

$$t(Q) = \{f'_1(Q), \dots, f'_k(Q)\} \quad (4.10)$$

where $\forall i \in [1, k] \ f'_i(Q) \subset f_i(Q)$. In other words, a pattern matching a query Q is defined by a subset of annotations in Q . Note that $f'_i(Q)$ can be the empty set.

Result

A *result* is determined by a pattern t and a query $b \in B(t, Q)$. We note $r = (t, b)$.

4.2.7 Query logs

Query logs are used to keep trace of users' queries, what results have been opened, which patterns have triggered which results, etc. These logs are used to compute metrics further detailed section 4.3.4 on the next page. An example of generated query log has been reproduced in the listing B.1 on page 143.

Query logs are of utmost importance for analyzing the behavior of users with respect to the data on the one hand, and the software being evaluated on the other hand. We will focus further on the former in section 5.4 on page 111; in particular we will focus on *patterns* of users' queries in the context of OLAP sessions.

4.3 Ranking the results

Not only one pattern usually matches the parse graph. For instance, some pattern are more *specific* than other ones, i.e. there more constraints in terms of expected annotations in the pattern section defining the constraints. As we expect the best results to appear first, we need to rank the results based on several metrics that we present in this section.

4.3.1 Confidence

Let $c_{i,j} \in [0, 1]$ be the confidence of an annotation (where (i, j) is the position of the annotation in the parse graph) and $d_i \in [0, 1]$ be a weight given to the i -th feature in the parse graph. Then the confidence of a result $r = (t, b)$ triggered by pattern t basing on annotations is given by:

$$s_1(r) = \sum_{i,j} \frac{d_i c_{i,j}}{k |f'_i(Q)|} \quad (4.11)$$

where k is the number of features in the pattern t and $|f'_i(Q)|$ the number of annotations of the i -th feature in the pattern t .

4.3.2 Selectivity

Selectivity is based on the number σ of structured queries that can be generated given a pattern and a question. Let $\sigma = |B(t, Q)|$ be the number of queries that have been generated by the pattern t corresponding to the result $r = (t, b)$. Then, confidence based on selectivity can be computed as:

$$s_2 = \begin{cases} \frac{1}{|B(t, Q)|} & \text{if } \sigma \neq 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.12)$$

The case where $\sigma = 0$ is the one where the pattern does not generate any query (useless pattern).

Note that all results r triggered by the pattern t will have the same selectivity.

4.3.3 Complexity

Complexity of a result r corresponds to the number of *entities* in the generate conceptual structured query (i.e. b). These entities can be dimensions, measures, filters, ...

Pattern	σ
1Measure	5
1Measure_1Dimension	$3 \times 7 + 2 \times 7 = 35$
1Measure_2Dimension	$3 \times C_2^7 + 2 \times C_2^7 = 105$

Table 4.4 – Example of selectivity metrics for the example question (4.3) and the dataset *eFashion* (see figure 1.3 on page 5)

Let $b = \{b_1, \dots, b_m\}$ be the query decomposed in its m entities s.t. $r = (t, b)$. Let T be the entity types (dimension, measure, filter, ...). Let $b' = (\text{count}_t(b))_{t \in T}$ be the vector representing the number of entities of type t in b (if the type t is not represented in the query b , then the vector item for t has the value 0). Then, the complexity of a result r is defined by:

$$s_3(r) = \frac{1}{|T|} \sum_{t \in T} \theta_t b'_{i,t} \quad (4.13)$$

where $0 < \theta_t < 1$ is a weight given to type t and experimentally determined s.t. $\sum_{t \in T} \theta_t < 1$.

4.3.4 Metrics from query logs

The query logs is a rich source of information about implicit user feedback on the result provided by the system. We focus on some metrics defined below.

Popularity

The definition of the popularity of a search result $r = (t, B)$ for user u is given by:

$$p_u(r) = \frac{t_{u,c}(r) - t_{u,o}(r)}{\max_r(t_{u,c}(r) - t_{u,o}(r))} \quad (4.14)$$

where $t_{u,c}(r)$ is the time when the result r was closed by user u and $t_{u,o}(r)$ the time when it was opened. This metrics measures how long a search result has been seen by the user. This metrics should be used in conjunction with a threshold. For instance, user might jump to another application once she gets the desired result. As a result this metrics would be extremely high, because she did not close the search result.

When the system is being used for the first time by user u (i.e. when u has never opened search result r), $t_{u,o}(r)$ and $t_{u,c}(r)$ are undefined. This case corresponds to the *cold-start* phase, and we consider the mean values for users in the social network of u .

Co-occurrence

Co-occurrence measures how likely different database entities could appear together in a query. This measure is also used as a ranking function in the context of auto-completion presented section 6.3 on page 131 and further detailed in [65]. The assumption behind this metrics, is that a pattern should get an higher rank if it generates a query composed of (database) entities with

high co-occurrence (i.e. entities that appear often in user's queries). The co-occurrence between two database entities e_1 and e_2 is given by the Jaccard index of the sets $occ_u(e_1)$ and $occ_u(e_2)$:

$$cooc_u(e_1, e_2) = J(occ_u(e_1), occ_u(e_2)) = \frac{|occ_u(e_1) \cap occ_u(e_2)|}{|occ_u(e_1) \cup occ_u(e_2)|} \quad (4.15)$$

where $occ_u(e)$ is the set queries that contain the entity e (computed from the query logs of user u). The co-occurrence of all entities in a structured query B is given by

$$cooc_u(B) = \binom{2}{|B|} \sum_{b, b' \in B} cooc_u(b, b') \quad (4.16)$$

The co-occurrence metrics for a result is defined as follows:

$$cooc_u(r) = \frac{1}{|B'|} \sum_{B \in B'} cooc_u(B) \quad (4.17)$$

where $B' = \{B \mid r = (t, B)\}$.

Implicit user preference

The popularity metrics (see equation 4.14 on the previous page) is used as a weight for the co-occurrence metrics to define users' implicit preferences:

$$pref_{u,impl}(r) = \frac{1}{|R|} \sum_{r \in R} \alpha_r p_u(r) cooc_u(r) \quad (4.18)$$

where $R = \{r = (t, B)\}$ and α_r is a parameter to be experimentally determined s.t. $\sum_{r \in R} \alpha_r = 1$.

Collaborative user preference

Ranking search results meets a similar goal as providing recommendations (in the sense of recommender systems). The metrics presented equation 4.18 presents a problem for *cold start* users, i.e. those new to the system. Indeed, those users do not have triggered search results, from which co-occurrences can be computed. Collaborative recommender systems have introduced the contribution of other users in the item scoring function to improve the system's coverage and enable the exploration of resources previously unknown (or unused) by the user. We follow the simple linear combination of the user-specific value and the average over the set of all users. Instead of considering "the whole world" where all users have the same role (weight), trust-based recommender systems illustrate the importance of considering users' social network, e.g., favorating users close to the current user. Let $SN(u)$ be the set of users in user u 's social network, filtered in order to keep only users up to a certain maximum distance. The refind user preference can thus be rewritten as:

$$pref_{impl}(u, r) = \alpha \cdot pref_{u,impl}(r) + \frac{\beta}{|SN(u)|} \sum_{u' \in SN(u)} \frac{pref_{u',impl}(r)}{d(u, u')} \quad (4.19)$$

where α and β are to be experimentally adjusted s.t. $\alpha + \beta = 1$ and $d(u, u')$ denotes the distance between both users u and u' .

Explicit user preferences

Explicit user preferences have not yet been implemented in the system (see chapter 3 on page 57). This kind of preference is expressed by users' ratings:

$$pref_{u,expl}(r) = \begin{cases} rating_{u,r} & \text{if } u \text{ has already rated } r \\ rating_u & \text{otherwise} \end{cases} \quad (4.20)$$

where $rating_{u,r} \in [0, 1]$ is the rating of user u for the result r and $\overline{rating_u}$ the overage rating given by u .

User preference

From both implicit (collaborative) user preference and explicit user preference defined equations 4.19 on the facing page and 4.20, we define the global user preference as a simple linear combination of $pref_{impl}(u, t)$ and $pref_{u,expl}(t)$:

$$s_4(r) = \alpha \cdot pref_{impl}(u, r) + \beta \cdot pref_{u,expl}(r) \quad (4.21)$$

where α and β are coefficients to be experimentally determined. This formulation through a linear combination allows us to easily see the impact of the implicit and the explicit preferences in the global metrics. As future work, one could investigate other formulations, but the current one seems to be satisfactory in our use-case.

4.3.5 Overall measure

We combine the different scores s_1 to s_4 to get the final score used as a ranking metrics. The scores s_2 and s_3 depend on the the question Q . In our experiments, we have combined these metrics as a linear combination of equal weight. The reader interested in metrics based on query logs will also find in section 6.3 on page 131 an application of similar metrics to auto-completion.

4.4 Summary & discussion

In this chapter, we have presented the core techniques used to translate users' question (formulated in NL) in structured queries (e.g. SQL or SPARQL). We have proposed a new way of formulating patterns inspired from the IE community as well as a base of patterns, which form together a knowledge-base dedicated to a Q&A system for BI purposes. Patterns define a set of constraints that users' questions must satisfy as well as the data (data models of the warehouses and data themselves) and knowledge bases (domain knowledge and linguistic knowledge for language-dependant NL expressions). These constraints are associated with a template of conceptual structured queries (that is then translated in each target query language as explained in the chapter 3 on page 57). Moreover, we have explained how we define ranking scores for the results that are generated by these patterns. The ranking function combines different metrics which take into consideration the confidence of the named-entity recognition involved in the mapping of words from the user's question to known terms, the complexity of the pattern (i.e. measuring the number of

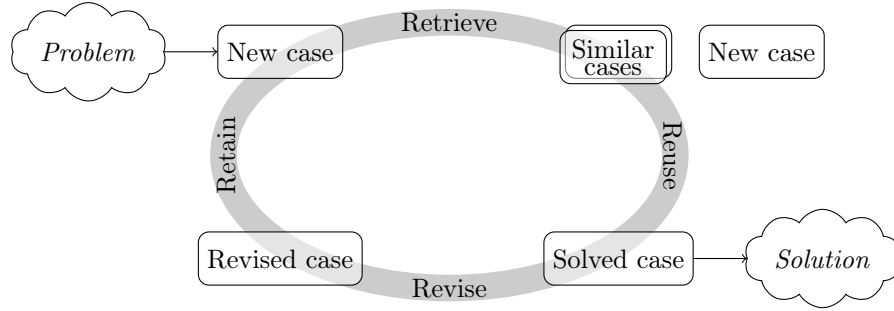


Figure 4.4 – Case-based reasoning approach applied to the problem of pattern learning

entities that are part of the generated structured query) and the specifics of the pattern (i.e. measuring the fact that a pattern is specific or generic).

However, the patterns used to translate users' questions in structured queries are costly resources. Therefore, we like to provide in the following thoughts on how to acquire such patterns. Developing additional patterns (for instance, for a new application domain, or in order to improve the linguistic coverage of the system) is not a straightforward task (see the example pattern reproduced appendix A on page 139). To ease this task, two classic approaches are:

- learning approaches, which are algorithms that base on trained (or labelled) data, on user interaction data or on both
- authoring front-end tools (generally user-friendly GUI) where users can easily generate new patterns on the basis of positive and/or negative examples for validating rules (i.e. patterns) being created

This aspect of the problem has not been fully investigated since we focus further on the implementation of the end-to-end system (see section 7 on page 135). We develop both possible approaches introduced above, as a starting point for future work on the subject.

4.4.1 Learning approaches

Case-based reasoning

Figure 4.4 is an illustration of the case-based reasoning approach for patterns. In this approach, the problem to solve can be formulated as follows:

“Given a parse graph $p(Q)$ for the question Q , what features $f_i(Q)$ shall be considered in the new pattern, and for each of these features, what annotations $a \in f_i(Q)$ from the parse graph shall be included in the pattern. Eventually, what conceptual query shall be associated with the pattern being build.”

New case The case correspond thus to the selection of

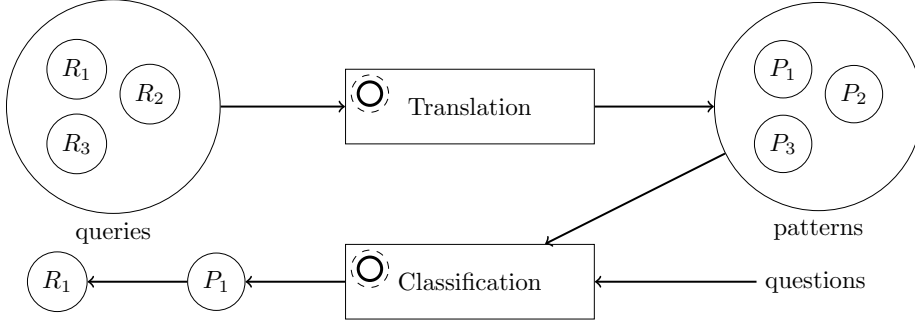


Figure 4.5 – Classification problem involved for a large number of patterns

- relevant features (i.e. the choice of $F \subset \{f_1(Q), \dots, f_k(Q)\}$ where k is the number of features in $p(Q)$)
- a mapping from the chosen features to the annotations, (i.e. $F \rightarrow A$)
- a generated conceptual query referred to as B

Retrieve The ‘Retrieve’ step consists in retrieving similar cases. The retrieving bases on similarity measures.

Reuse The ‘Reuse’ step aims at transforming the set of similar cases retrieved in the preceeding step in order to build a new case (called *Solved case* in the figure) which corresponds to the proposed solution to the problem stated above. Thus, a solved case is made of a pattern – a set of constraints to be satisfied by a parse graph plus a template of structured query.

Revise The solution proposed in the previous step has not been yet validated (to prevent the system from being corrupted). The ‘Revise’ step consists in validating cases proposed in the previous steps. For instance, a case can be validated based on user’s feedback about positive and negative examples.

Retain The ‘Retain’ step consists in storing the case in the system, so that it can be easily re-used in the future when a similar problem arises.

The approach described so far has been employed in the Q&A domain by Ben Mustapha et al. [7]. They propose a system that learns an ontology used to guide the interpretation of users’ questions in the context of semantic search.

Genetic approach

We have investigated the case where there might be a large number of patterns in the repository. This case is relevant when the number of patterns increases over time, for instance through machine learning algorithms. In this case, we need a classification to decide which patterns are closer to the user’s question. The approach that we have investigated is represented figure 4.5. In this approach, the classification algorithm must first be determined. This classification is then used to determine the translation of queries into patterns, using a genetic algorithm and a learning base.

Constraints Translation rules must satisfy following constraints:

- examples must be correctly translated
- generated patterns should be valid

These constraints, to be satisfied by individuals, lead to bonus scores. Figure 4.6 is an illustration of the mutation process and selection of *good* indi-

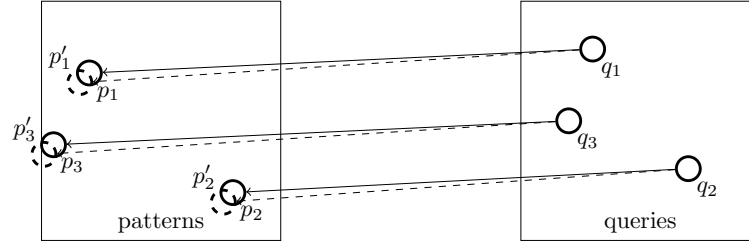


Figure 4.6 – Illustration of how individuals are mutated and rewarded. Dashed circles on the left-hand side (p'_i) are individuals automatically generated out of queries q_i . Individuals that are not patterns (e.g. q'_3) are left out. To be rewarded, generated patterns must be *similar* to original examples (i.e. p_i).

viduals based on their structure (for instance individuals that are not patterns will be left out) and based on how faithfully they reproduce good examples (i.e. how close are p'_i to p_i). On figure 4.7 one illustrates the problem of generating

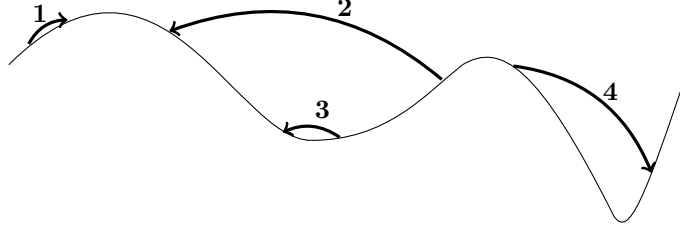


Figure 4.7 – The curve represents both *novelty* (x -axis) and *efficiency* (y -axis) of newly-created individuals being mutated. The problem can be reduced to the search for local maximums. Area ‘1’ leads to better individuals, but they are similar to the known ones; area ‘2’ is both efficient and novel; area ‘3’ is not efficient and not novel and this area should be avoided; area ‘4’ leads to worse individuals. In the latter case, one must decide whether one should pursue or stop the local discovery of new individuals

new individuals, that must be both *efficient* and *novel*. The former property relates to the generality of the generated pattern (with respect to selectivity, see section 4.3.2 on page 88). The latter one depends on the distance with known examples and already-generated individuals.

See also This approach has been further described by Watel [70].

4.4.2 Authoring tool

Authoring tools have been quite recently used in interfaces for Q&A (one the most representative related work is NALIX [42]). The benefits of such tools, is that they allow non-expert users to enrich the system with new semantic rules. Typically, users are assisted graphically in creating semantic mappings, and examples of known pairs of questions and answers are displayed to users, so that they can validate the rules being created. In our proposal (see figure 4.8), the SPARQL pattern would be generated by the tool; the user would simply express graphically a set of constraints based on positive and negative examples.

Pattern acquisition

Revenue in New York in Q2 ☐ ☐

Margin in Texas in Q2 ☐ ☐ OR

Sales revenue in Houston in Q2 ☐ ☐ OR

Days in Chicago in Q2 ☐ ☐

Figure 4.8 – Authoring tool: user loads unresolved questions

Identification of common annotations

First, users are invited to type a serie of questions (or to import them from an external file). All these questions are supposed to be captured by the same pattern. The drawback is that users are supposed to know the data, so that they can think of queries (the system can also keep trace of unresolved question that have been asked in the past, and suggest those questions). For example, the questions “Revenue and margin in New York in Q2” and “Sales revenue and margin in Texas in Q3” seem to correspond to the same pattern (i.e. two measures and two filters). The set of questions are parsed, and the annotations (with common annotation types) are used as annotations in the generated pattern. In the end of this process, users can remove some annotations if they think that they are not relevant for the current pattern.

Graphic construction of the structured query

The semantic information in the pattern lies in the mapping between the set of annotation (the **WHERE** section of the pattern) and the generated structured query (the **CONSTRUCT** section of the pattern). To this end, the graphic editor guides users in designing the conceptual query, based on the annotations that were identified in the previous step. Figure 4.9 on the following page is a screenshot of a mockup.

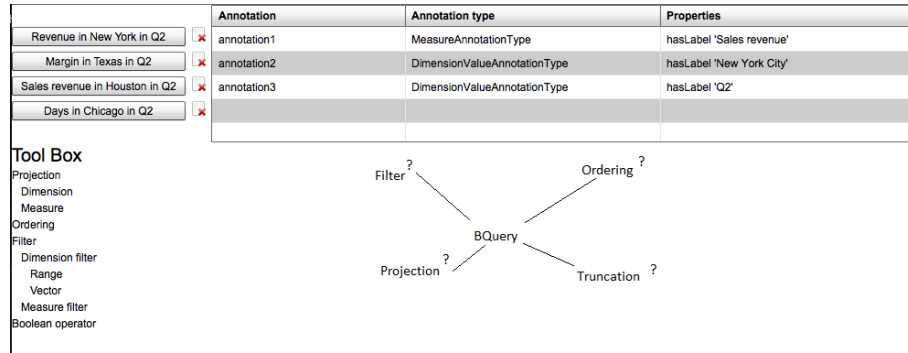


Figure 4.9 – Graphic construction of the pattern

Validation of the candidate pattern

Once user has finished designing the conceptual query, the pattern should be validated to check that the results are the ones that the user expects. The questions used in the first step (see section 4.4.2 on the previous page) are used to test the candidate pattern. The validation process, for each question, is as follows:

1. check that the pattern matches the question
2. execute the query and display the corresponding chart

If users are not satisfied with the pattern execution, they are able to get back to the first step (see section 4.4.2 on the preceding page) or the second one (see section 4.4.2 on the previous page). Eventually, when they are satisfied with the generated pattern, they can add it to the pattern repository. Finally, a last step would be required to test that the newly created pattern does not generate conflicts with respect to other patterns in the system.

Chapter 5

Query Modeling

Outline

5.1	Fact tables and functional dependencies	97
5.2	Multidimensional queries and preferences	98
5.2.1	Formal representation of a query	99
5.2.2	Database queries	102
5.2.3	Personalization of multidimensional queries	103
5.2.4	Example of how patterns are mapped to structured queries	107
5.3	Similarity measure based on preferences	109
5.4	Prediction model	111
5.4.1	Motivating Scenario	111
5.4.2	Architecture	112
5.4.3	Query Clustering	113
5.4.4	Query Prediction	114
5.5	Summary & discussion	116

Multidimensional models have been introduced in chapter 1 on page 1. These models support users to express queries, since terms used to describe model *entities* are terms used by the community (e.g. business terms), while logical terms (e.g. the table names in a relational database schema) are manipulated by database administrators.

A model can be represented as $\mathcal{M} = (A, H, M)$, where $A = \{a_i\}$ is a set of attributes (i.e. dimensions and attributes of dimensions), $H = \{h_i\}$ is a set of hierarchical levels and $M = \{m_i\}$ is a set of measures.

5.1 Fact tables and functional dependencies

The execution of a multidimensional query results in a set of *facts*. Those facts are extracted and aggregated from the fact tables. How these operations are performed depends on the implementation of the database system. In the following, a fact is written f and graphically display as a row, where each column correspond to an attribute of at a given level or to a measure. In the former case, the cell contains the selected attribute value at the given level (or **Members** if no value is selected, i.e. if the attribute is not used as a filter). In

the latter case the cell contains the measure's value corresponding to the filters expressed (or not) in the other cells of the row. Table 5.1 provides an example of

Fact	[Year]	[Title]	(Sales amount quota)
f_1	2001	Pacific Sales Manager	383200
f_2	2001	European Sales Manager	1124400
f_3	2001	North American Sales Manager	1660050
f_4	2001	Members	3167650

Table 5.1 – Facts answering the question “Sales target per department in 2001”

three facts, with the dimension [Title] and the measure (Sales Amount Quota) from the dataset AdventureWorks¹. The last row of the table is an example of aggregation of the facts along the [Title] dimension (**Members** stands for a selection of all possible members of the corresponding dimension). This fact (f_4) would be part, in our case, of an aggregate fact table. There can be more than one attribute and more than one measure for each fact (i.e. on each row of the fact table). Let f_i be the i -th fact in the table 5.1 (i.e. the i -th row). We note $f_i \cdot m_j$ the value of the measure m_j for the fact f_i and $f_i \cdot a_j$ the selected value of the attribute a_j (similar notation as the one used by Golfarelli et al. [24]). Thus, according to table 5.1, $f_1 \cdot (\text{Sales amount quota}) = 38200$; $f_1 \cdot [\text{Title}] = \text{'Pacific Sales Manager'}$; $f_4 \cdot [\text{Year}] = 2001$ and $f_4 \cdot [\text{Title}] = \text{Members}$.

Dimensions and measures that appear in a single fact table are said *compatible* or *functionally dependant*, because they can be used together in a database query (MDX or SQL). Let E be the set of entities (attributes, hierarchy levels and measures). Functional dependency \mathcal{D} is an equivalence relation:

- \mathcal{D} is reflexive: $\forall e \in E \ e \mathcal{D} e$
- \mathcal{D} is symmetric: $\forall e, f \in E \ e \mathcal{D} f \Rightarrow f \mathcal{D} e$
- \mathcal{D} is transitive: $\forall e, f, g \in E \ \begin{cases} e \mathcal{D} f \\ f \mathcal{D} g \end{cases} \Rightarrow e \mathcal{D} g$

We note $d(e)$ the equivalence class to which entity e belongs. Thus, $e \in d(f)$ means that e belongs to the set of entities with which f is compatible. Compatible entities are of interest, for instance to make sure that suggested entities can be used to generate a valid query (this is illustrated in an application for auto-completion in section 6.3 on page 131).

5.2 Multidimensional queries and preferences

A multidimensional query can be expressed as a set of dimensions (at a specific hierarchy level), a set of measures plus constraints (or filters) on members (e.g. selection of a subset of the members, or selection of all members) or on measures (e.g. selection of all members where a specific measure has values in a specific range) plus query modifier (such as ordering clauses or the truncating operator).

¹The dataset can be downloaded at [http://msdn.microsoft.com/en-us/library/ms124623\(v=sql.105\).aspx](http://msdn.microsoft.com/en-us/library/ms124623(v=sql.105).aspx).

In the following, we will present a formal representation of a query section 5.2.1. Then, we will present how this formal representation can be automatically translated into a common representation section 5.2.2 on page 102.

5.2.1 Formal representation of a query

The following structure:

$$Q = \left[\begin{array}{ll} \text{dimensions} & = \{[\text{Department}]\} \\ \text{measures} & = \{(\text{Sales target})\} \\ \text{filters} & = \{[\text{Year}] = 2001\} \\ \text{truncation} & = \emptyset \\ \text{ordering} & = [([\text{Department}], (\text{Sales target}). \uparrow)] \end{array} \right] \quad (5.1)$$

stands for a conceptual representation of a multidimensional query. The different attributes of the structure are:

- dimensions: (unordered) set of dimensions
- measures: (unordered) set of measures
- filters: (unordered) set of filters. A filter is a restriction on the resultset, based on a dimension (selection of some dimension values) or on a measure (where the measure value must verify a specific condition, like being in a range of values)
- truncation: ordered list of truncation clauses. In some cases, only part of the resultset must be returned. In particular, a truncation clause is used in conjunction with an ordering clause in queries like ‘top n ’, where n is the number of facts to be returned (see the **LIMIT** operator in MDX).
- ordering: the facts from the resultset must be ordered along a dimension or measure, in ascending order (\uparrow) or descending order (\downarrow).

We further describe this structure in the following.

Analysis axes or dimensions

Dimensions are used to describe and aggregate facts (see section 5.1 on page 97). Some dimensions belong to the same hierarchy (i.e. they are different levels of the same hierarchy, as introduced in section 1.1.2 on page 6). For instance, there are usually time and geographic hierarchies in multidimensional models. The time hierarchy can be described as follows:

$$\boxed{\text{Year}} \rightarrow \boxed{\text{Quarter}} \rightarrow \boxed{\text{Month}} \rightarrow \boxed{\text{Week}} \rightarrow \boxed{\text{Day}} \dots \quad (5.2)$$

and similarly the geographic hierarchy can be described as:

$$\boxed{\text{Region}} \rightarrow \boxed{\text{Country}} \rightarrow \boxed{\text{State}} \rightarrow \boxed{\text{County}} \rightarrow \boxed{\text{City}} \rightarrow \boxed{\text{Street}} \dots \quad (5.3)$$

Note that hierarchies can be seen as ordered sets of dimensions, where successive dimensions share a hierarchical relation. As a result, there is not a unique decomposition of hierarchies in hierarchy levels (the decomposition depends on the chosen hierarchical relation).

In our model, we introduce a structure called *axis group*, which represents a branch or a *segment* from a hierarchy of the formal representation. This axis group is then broken down into *headers* which is then broken down into *header items*, as represented figure 5.1. We have defined this structure, because it looks

		Axis group 1				
		Header 1 FR Paris	Header 2 FR Levallois-Perret	Header 3 DE Berlin	Header 4 DE Dresden	Header <i>n</i> Item 1 ...
Axis group 2	Header 1 2010	...				
	Header 2 2011					

Figure 5.1 – Decomposition of *axis groups* in the formal query representation

similar to a bi-dimensional table (or *crostable*), where both dimensions are the two axis groups. There can be more axis groups in our structure. Rendering data structures with more than two axis groups is illustrated with an example in the appendix, section D.3 on page 148. Each axis group is decomposed in *headers* which correspond to the selection of members for the different dimensions involved in the hierarchy. For instance, in figure 5.1, **Header 1** stands for the selection of country ‘FR’ and city ‘Paris’ (because [Country] and [City] belong to the same hierarchy). Then, each header is decomposed in a series of items, namely *header items*. Those items correspond to the different hierarchy levels of the hierarchy which corresponds to the axis group.

Measures

Measures values are numeric values (in general integer or double values) that can be aggregated along different dimensions used in the model. A measure is thus always associated to an aggregation function (which can be **sum** or **avg**, but can be also more complex and defined in the data schema). In figure 5.1 which serves as a visual representation, measures’ values would be represented in imaginary multidimensional cells in the body of the table. The dimension of these cells would be the number of measures.

Filters

Filters are restrictions of a resultset based on dimensions’ or measures’ values. For instance, the query (5.4):

$$Q = \left[\begin{array}{ll} \text{dimensions} & = \{[\text{Year}]\} \\ \text{measures} & = \{(\text{Sales revenue})\} \\ \text{filters} & = \{\text{City} \in \{\text{‘New York’}, \text{‘Boston’}\}\} \\ \text{truncation} & = \emptyset \\ \text{ordering} & = [[\text{Year}], [\text{Sales revenue}]. \uparrow] \end{array} \right] \quad (5.4)$$

leads to the database query listing 5.1 on the facing page:

```

1  SELECT
2    sum(Table__2."AMOUNT_SOLD"), Table__7."YR"
3  FROM
4    "EFASHION"."CALENDAR_YEAR_LOOKUP" Table__7
5  INNER JOIN
6    "EFASHION"."SHOP_FACTS" Table__2 ON (
7    Table__2."WEEK_ID"=Table__7."WEEK_ID"
8    )
9  INNER JOIN
10   "EFASHION"."OUTLET_LOOKUP" Table__1 ON (
11   Table__1."SHOP_ID"=Table__2."SHOP_ID"
12   )
13 WHERE
14   Table__1."CITY" IN ('New_York', 'Boston')
15 GROUP BY
16   Table__7."YR"
17 ORDER BY 2

```

Listing 5.1 – SQL query generated from conceptual query (5.4)

In this example, the query is composed of one filter which restricts possible values of dimension [City] to ‘New York’ and ‘Boston’. It is possible to have more than one filter, and filters based on measures’ values. In general, the generated SQL query looks different (the clause **HAVING** is introduced when a join is involved in the condition), but we keep a similar representation in the conceptual query. Query (5.5) is an example of a query with a filter based on the values of a measure:

$$Q = \left[\begin{array}{ll} \text{dimensions} & = \{[Year]\} \\ \text{measures} & = \{(Sales\ revenue)\} \\ \text{filters} & = \{Sales\ revenue > 1,000,000 \$\} \\ \text{truncation} & = \emptyset \\ \text{ordering} & = [[Year], [Year]. \uparrow] \end{array} \right] \quad (5.5)$$

The generated SQL query is reproduced listing 5.2.

```

1  SELECT
2    sum(Table__2."AMOUNT_SOLD"),
3    Table__7."YR"
4  FROM
5    "EFASHION"."CALENDAR_YEAR_LOOKUP" Table__7
6  INNER JOIN "EFASHION"."SHOP_FACTS" Table__2 ON (
7    Table__2."WEEK_ID"=Table__7."WEEK_ID"
8    )
9  GROUP BY
10   Table__7."YR"
11 HAVING
12   sum(Table__2."AMOUNT_SOLD") > 10000000
13 ORDER BY 2

```

Listing 5.2 – SQL query generated from conceptual query (5.4)

Keywords **HAVING** and **WHERE** can be combined in the generated SQL or MDX queries.

Truncation

Truncation clauses are used to select a subset of the resultset. It can be associated with an ordering clause (see below). The truncation clause is composed of an ordering and an integer value, which is the number of items from the resultset to be returned. The ordering specifies along which dimension or measure should the resultset be ordered before being truncated. In MDX, for instance, this truncation clause is translated in the keyword `LIMIT`.

Ordering

The ordering clause specifies how the rows of the resultset should be ordered. There can be more than one ordering clause, but the order of the clauses is important (we use `[]` to note the ordered set). Each item of the clause is a triple of the form $(a_1, a_2, \text{direction})$ where a_1 and a_2 can be measures or dimensions and direction stands for \uparrow or \downarrow (i.e. for ascending or descending ordering respectively). There can be more than one ordering clause. The second clause is considered when there are *equal* rows (in the sense of the order defined by \uparrow and \downarrow). In practice, there are rarely more than two ordering clauses.

5.2.2 Database queries

The most common syntax for multidimensional queries is MDX. An example of a MDX query is reproduced listing 5.3:

```

1 SELECT
2   { [Measures].[Store Sales] } ON COLUMNS,
3   { [Date].[2002], [Date].[2003] } ON ROWS
4 FROM Sales
5 WHERE ( [Store].[USA].[CA] )
```

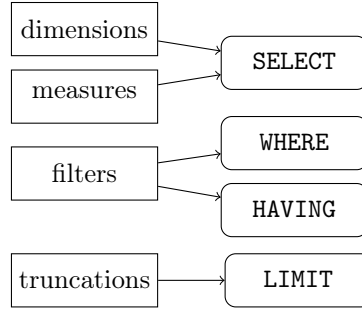
Listing 5.3 – Example of MDX query

In addition, MDX provides several built-in functions, such as date or member functions. MDX or SQL queries can be automatically generated from the data schema of the warehouse. The conceptual query is first decomposed in its clauses, and each clause is associated with a database query fragment. All these fragments are then merged into a single database query. We have experimented SAP BusinessObjects Semantic Layer™ to automatically generate database queries out of conceptual queries. Figure 5.2 on the facing page is an overview of the translation of conceptual queries into database queries (MDX or SQL, depending on the data source).

Semantic layer

The above-mentioned *semantic layer* is a software that interface any DBMS and let users build queries as an object representation, and unburden them from formulating DBMS-specific expressions.

This tool is widely used in BI. We experimented it in an early version of our Q&A system for data warehouses [39].

**Figure 5.2** – Automatic database query generation

5.2.3 Personalization of multidimensional queries

Personalizing multidimensional queries is an approach, where *preferences* can be expressed in queries. These preferences are of different types. On the one hand, quantitative preferences, like the one defined in [1] map objects of preference to a value representing the degree of preference (for instance, a scoring function is used to rank individual tuples based on preferences). On the other hand, qualitative preferences are not associated with scoring functions and can be expressed in various ways, like in [37]. The latter kind of preference is usually expressed in an algebra. Chomicki [11] has shown that qualitative preferences yield a broader expressiveness than quantitative preferences.

In order to offer personalized results, database queries are often enriched with *preferences* [24]. For example, consider the structured query displayed listing 5.4 that has been generated from conceptual query (5.1):

```

1  SELECT
2    NON EMPTY {[Measures].[Sales Amount Quota]} ON COLUMNS,
3    NON EMPTY Hierarchize(Crossjoin({[Employee].[Employee
4  Department].[Department].Members},
5    [Date].[Calendar Year].[Calendar Year].Members))
6  DIMENSION PROPERTIES
7    PARENT_UNIQUE_NAME ON ROWS
8  FROM(
9    SELECT {[Date].[Calendar Year].[2001]}
10  ON COLUMNS FROM [Adventure Works])

```

Listing 5.4 – Structured query generated from conceptual query (5.1)

In some cases, this structured query leads to only one tuple, which probably means that the query should be drilled down along the dimension [Employee Department], that is:

```

1  [Employee].[Employee Department].[Department].Members

```

should be replaced with hierarchy level

```

1  [Employee].[Employee Department].[Title].Members

```

This is a naive example of preferences that can be naturally expressed in addition. In the following, we introduce a framework for expressing this kind of preferences plus additional preferences well suited for BI use-cases.

Expressing multidimensional preferences

The preference in the previous example (see listing 5.4 on the preceding page) states that finest aggregated facts along the given hierarchy are *preferred* than the coarsest ones. This kind of qualitative preferences is well expressed in the MyOLAP algebra [24]:

FINEST([Employee][Employee Department])

The algebra enables the expression of preferences on hierarchies (like in the example above), say $\text{CONTAIN}(h, a)$ (facts including the level a along h are preferred), $\text{NEAR}(h, a_2, a_1)$ (facts which group-by set along h is between a_2 and a_1 are preferred), $\text{COARSEST}(h)$ and $\text{FINEST}(h)$. Preferences on attributes are $\text{POS}(a, c)$ (facts mapping to the member c are preferred) and $\text{NEG}(a, c)$ (facts not mapping to c are preferred). For measures, preferences are $\text{BETWEEN}(m, v_1, v_2)$ (facts which value on m is between v_1 and v_2 are preferred), $\text{LOWEST}(m)$ (facts which value on m is as low as possible) and $\text{HIGHEST}(m)$. In addition, preferences can be composed with two operators: the composition $P_1 \otimes P_2$ (composed preferences P_1 and P_2 are equally important) and prioritization $P_1 \triangleright P_2$ (preference P_1 is prioritized with respect to P_2). The formal definitions and notations introduced above can be found in [24].

Name	Distribution	Comparison	Trend	Composition	Gradient
Tree map				x	
Heat map				x	x
Pie chart				x	
Pie with variable slice depth				x	
Multiple pie chart				x	
Donut chart				x	
Column chart		x			
Bar chart		x			
Column chart with dual axes		x			
Line chart			x		
Line chart with dual axes			x		
Area chart				x	
Combined column and line chart		x	x		
Combined column and line chart with dual axes		x	x		
Stacked column chart		x		x	
100% stacked column chart		x		x	
Stacked bar chart		x		x	
100% stacked bar chart		x		x	
Multiple bar chart		x			
Multiple dual bar chart		x			
Multiple line chart			x		
Multiple dual line chart			x		
Multiple surface chart				x	
3D column chart		x			
Box plot chart		x			

Waterfall chart	x		
Radar chart		x	x
Multiple radar chart		x	x
Tag cloud chart	x		
Scatter chart	x		
Multiple scatter chart	x		
Bubble chart	x		x
Multiple bubble chart	x		
Scatter matrix chart		x	
Polar scatter plot		x	
Polar bubble chart		x	x

Table 5.2 – Some chart types and their associated analysis types

Chart type preference & analysis type

In addition to qualitative preferences from MyOLAP, we consider the chart type as a preference. Indeed, all multidimensional queries that we generate are intended to be visualized as charts. Given the structure of the internal query (5.1), there are many ways to render it as a chart. However, chart types are often associated with an *analysis type* which partly corresponds to the meaning of the chart for the user. This has been outlined in [63].

Table 5.2 on the preceding page outlines different analysis types corresponding to different chart types. This table has been created by hand on the basis of analysis of some existing dashboards. A broader and deeper analysis should be performed, in order to:

1. take into consideration new chart types (that are not part of the dashboards that we have considered)
2. validate the content of the table, which is subject to personal point of view with respect to the semantics of charts belonging to the dashboards

5.2.4 Example of how patterns are mapped to structured queries

For most of the cases like the example given in figure 4.1 on page 78, a structured query contains a *data source*, a set of *dimensions* and *measures*, a set of *filters* and an optional set of result modifiers, e.g. *ordering* or *truncation* expressions. For the example from figure 4.1 on page 78, a structured query could be represented as follows:

$$Q_1 = \left[\begin{array}{ll} \text{data source} & = \text{Resorts} \\ \text{dimensions} & = \{\text{Customer}\} \\ \text{measures} & = \{\text{Revenue}\} \\ \text{filters} & = \left\{ \begin{array}{l} \text{City} = \text{'Palo Alto'}, \\ \text{Age} \geq 20, \\ \text{Age} \leq 30 \end{array} \right\} \\ \text{truncation} & = \{(\text{Revenue}, \downarrow, 5)\} \end{array} \right]$$

In there, curly brackets represent a set of objects, which might have a complex structure (e.g. for filters, which consist of a measure or dimension, an operator and a value). For truncations we use a triple consisting of the dimension or measure on which the ordering is applied, the ordering direction (ascending \uparrow , or descending \downarrow) and the number of items. Another interpretation for the user's question would be Q_2 , which is similar to Q_1 except the proposed measure:

$$Q_2 = \left[\begin{array}{ll} \dots & \\ \text{measures} & = \{\text{Margin}\} \\ \dots & \\ \text{truncation} & = \{\text{Margin}, \downarrow, 5\} \end{array} \right]$$

Since the representation shown above captures only a fraction of the potential queries, we use RDF to capture the structure and semantics of the structured query which is then serialized to an executable query in a subsequent step.

As discussed earlier², we define in the left part of figure 4.3 on page 83 how to derive potential interpretations (i.e. variables and the constraints between them) using a SPARQL **WHERE** clause. Now we need to define the basic structure of a query (in RDF) and how to map variables into this model using a SPARQL **CONSTRUCT** clause (illustrated in the right part of figure 4.3 on page 83). In this way, we separate the pattern matching, which can be quite complex, from the actual mapping problem and ensure a fine-grained flexible control on how to generate structured queries.

Some of the most important concepts of our query model are illustrated in figure 4.3 on page 83. On the top, stands the root node **Q** defining a structured query. Below, dashed lines represent parts that are optional in the left side. These parts of the **CONSTRUCT** clause are only triggered if the respective variables are in the result of the **WHERE** clause, making it easy to describe alternative mappings for different situations as described in the *parse graph*. Besides of the actual query semantics, we attach some metadata nodes to the query node such as the *data source* **DS**. It is bound to the variable ‘?w’ representing the actual data warehouse upon which the generated query shall be executed. Additional nodes are dedicated to: *projection items* **PI**, capturing all projections that are part of the final structured query; *filter items* **FI**, expressing selections on a certain measure or dimension and *truncation and ordering* clauses **TO**. The underlying structures are detailed in the following.

Projections

The most important part of the actual query are projections, which in our use-case consists at least of one measure and dimension. To give a glimpse on our full query model and further detail the example, we define different kinds of expressions (via a common ancestor RDF type) where we depict here the subclasses *measure expression* **ME** and *dimensions expression* **DE**. These nodes capture common metadata (not shown here), such as navigation paths (e.g. for drill-down operations) or confidence scores and refer to the actual object that defines the projection, here the *measure reference* **MR** and *dimension reference* **DR**. They are in our case the labels of recognized objects. It does not matter whether we use the recognized dimensions and measures (derived from ‘m1’ or ‘d1’) or the suggested ones (derived from ‘m2’ or ‘d4’) in the final query since we defined in the **WHERE** clause that suggestions are only made if no user input is available. We plan to include more complex artifacts such as subnodes of the *expression* ancestor node to support for instance computed measures.

Truncation and Ordering

The node **TO** in figure 4.3 on page 83 stands for *Truncation and Ordering*. It represents **ORDER BY** and **LIMIT** clauses of a structured query or of a certain sub-select within such a query. Thus, several nodes **TO** can occur as sub-node of a query node. If the variable ‘?nb’ is not bound by the ‘TopK’ pattern, the default value as described in section 4.2.5 on page 86 will be used and a single **LIMIT** will be generated. The ‘Sorting Expression’ **SE** representing an **ORDER BY** is not being generated in that case because the variable ‘?ord’ is unbound. If the

²We refer in this section to the figure 4.2 on page 80.

user entered a question starting with ‘Top...’ both variables ‘?nb’ and ‘?ord’ would be bound and we would suggest an artifact to apply the ordering (unless the user entered ‘order by ...’, which is parsed by a dedicated pattern). Since *top-k* questions usually relate to a particular measure (even if the query would be ‘top 5 cities’), we can safely apply the order to the recognized or suggested measure by simply relating the node for the ‘Sorting Expression’ (SE) to the one for the measure (MR). Note that in any case every possible interpretation with respect to the ORDER BY assignment would be generated.

Filters

Filter expressions depicted as (FE) represent a set of members or numerical values in the case of measures to be used to filter the actual result. From a data model perspective, filter expressions capture the metadata’s object (either dimension or measure) on which the restriction is applied and a set or range of values that defines the actual restriction. More complex filter expressions can be defined as well (e.g. containing a sub-query). In our example, we show only examples for *member sets* (MS) containing a single member which is represented by a *value reference* (VR). In the first case, a member was directly recognized in the user’s question. The variable ‘?dL2’ originating from the dimension ‘?d2’ is directly assigned to the *member set* and a node for the *value reference* (VR) is generated with a property for the actual value (i.e. ‘?vL1’). Note that we do not need to care whether the respective dimensions will be considered in the projections since this can be handled by constraints (see left part of figure 4.3 on page 83). The second example handles personalization (e.g. “my city”) and uses a filter leveraging the user profile. It works similarly as the one for matched members except that the *value reference* (VR) relates to the label of the object in the user profile that carries a similar value as one of the members of a certain dimension (e.g. ‘Palo Alto’ for the dimension ‘City’). Range queries are conceptually similar to the ones containing a *member set*, no matter whether they are applied on dimensions or measures. The only difference is that a natural language pattern is used for detecting numeric or date range expressions in the user’s question to define variables and that there are two *value references* defining the bounds of the actual *filter expression*.

5.3 Similarity measure based on preferences

In the following, we use the notations from [21]: we note $C = \langle C_1, \dots, C_n, F \rangle$ a n -dimensional cube where n is the number of dimensions and F stands for the fact table. The values of the dimension D_i noted $Dom(D_i)$ are called members and organized in a hierarchy H_i . A reference is a n -tuple $\langle r_1, \dots, r_n \rangle$ where $r_i \in Dom(D_i)$ for $i \in [1, n]$. A MDX query is modeled as a set of references for a given instance of a cube.

The distance between two MDX queries is defined as [21]:

$$d_{MDX}(q_1, q_2) = \gamma \times d_{dim}(q_1, q_2) + (1 - \gamma) \times d_h(q_1, q_2) \quad (5.6)$$

where $\gamma \in [1, n]$ is a parameter, d_{dim} measures the number of different references and d_h the measure inspired from the Hausdorff distance [30] which measures the distance between two sets based on the distance between the elements of the sets.

The similarity measure d_{MDX} (see equation 5.6 on the previous page) does not take into account preferences that appear in MyOLAP queries [24]. To remedy this, we propose an alternative measure d_{MLP} as follows:

$$d_{MLP}(q_1, q_2) = \delta \times d_{MDX}(q_1, q_2) + (1 - \delta) \times d_{pref}(q_1, q_2) \quad (5.7)$$

where $\delta \in [0, 1]$ is a parameter and where $d_{pref}(q_1, q_2)$ measures a distance in terms of preferences held by MyOLAP queries q_1 and q_2 .

Let C_{q_1} be the set of facts in the result of q_1 on the cube C and C_{q_2} be the set of facts in the result of q_2 on the cube C . Notations and definitions can be found in [24]. We define the *priority* p_{P_i} as a binary indicator that two facts share a preference relation (and are not equivalent in terms of preference); i.e.

$$p_{P_i}(f_1, f_2) = \begin{cases} 1 & \text{if } f_1 <_{P_i} f_2 \text{ or } f_2 <_{P_i} f_1 \\ 0 & \text{otherwise} \end{cases}$$

where $f_1 \in C_{q_1}$ and $f_2 \in C_{q_2}$. Let P_1 be the preference of query q_1 and P_2 be the preference of query q_2 ; we define the distance between q_1 and q_2 based on preferences P_1 and P_2 as:

$$d_{pref}(q_1, q_2) = \epsilon \sum_{(f_1, f_2) \in C_{q_1} \times C_{q_2}} \left(\frac{(p_{P_1}(f_1, f_2) + p_{P_2}(f_1, f_2))}{2|C_{q_1}| \cdot |C_{q_2}|} \right) + (1 - \epsilon) d_{pref}(C_{q_1}, C_{q_2})$$

where $\epsilon \in [0, 1]$ is a parameter and $d_{pref}(C_{q_1}, C_{q_2})$ is the preference distance which is not based on individual facts, but on the sets of facts that belong to the same result C_{q_i} . We define $d_{pref}(C_{q_i}, C_{q_j})$ as:

$$d_{pref}(C_{q_i}, C_{q_j}) = \begin{cases} \frac{1}{Jacc(C_{q_i}, C_{q_j})} & \text{if } Jacc(C_{q_i}, C_{q_j}) \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

An example of preference that does not apply to two facts but to a *result* (i.e. to a set of facts) is the preference about which chart type to choose in order to render the result. Let the facts $f \in C_q$ be summarized in table 5.1 on page 98 where q is the question “*Sales target per department in 2001*”. Two popular representations for these facts could be a bar chart or a pie chart representations (these charts have been reproduced figure 3.6 on page 67). The preferred chart type is inferred from both the data, and *annotations* coming from the NL analysis of the question, or even the application which triggers the chart rendering. Bar charts are usually used for comparing data, while pie charts is an analysis of the composition or contribution of data. The analysis types that we consider for some chart types are reported table 5.2 on page 106. The information about which is the preferred chart type (for a given structured query) is stored in the conceptual query (see section 5.2.1 on page 99).

A simple similarity measure between two chart types is based on the common number of analysis types. Let $A = \{a_j\}$ be the set of analysis types (first column of table 5.2 on page 106) and $C = \{c_i\}$ be the set of chart types (headers of table 5.2 on page 106). Let c_i be the chart associated to the result C_{q_i} and $types : C \rightarrow A$ be the function such that $types(c_i) \subset A$ is the set of analysis types corresponding to chart c_i . The Jaccard index on these types is defined as:

$$Jacc(C_{q_i}, C_{q_j}) = Jacc'(c_i, c_j) = \frac{|types(c_i) \cap types(c_j)|}{|types(c_i) \cup types(c_j)|}$$

This index measures the number of common analysis types between two chart types.

Note that we have presented here only one example of preference relating two results (which are defined as sets of facts) for the chart type preference, because is relevant to our application. The metrics $d_{pref}(C_{q_i}, C_{q_j})$ can be further defined taking into account additional preferences. Examples are the size of the result (i.e. which would be the preferred size for the chart?) or the preferred colors of the chart, etc.

5.4 Prediction model

One believe that successive queries in logs follow some pattern, because users usually build queries in an iterative manner [56]. This assertion is hard to prove though, because real query logs are costly resources. For reducing execution time of queries in OLAP sessions, it would therefore be of interest to pro-actively execute queries that would be the next queries triggered by users, according to the pattern mentioned above.

In this section, we present a proposal for a prediction model for predicting the most likely next queries in the context of an OLAP session.

5.4.1 Motivating Scenario

Let us suppose an OLAP user during a typical week at work who interacts with the company data warehouse to perform several tasks:

- She is responsible for making management reports for the daily and weekly sales of one product branch, and for analyzing the effect of publicity on the sales targets in different regions for that specific branch.
- She is also responsible for market research for all products, finding factors that significantly influence sales as they evolve over time.

While performing queries to obtain management reports and doing market research, the user has learned that it gives best results first to get a sales report per region, and then to analyze the effects of publicity on the sales according to region. It is generally hard to determine which factors influence measures like ‘Sales’, and therefore user experience helps in choosing the correct axis of analysis. A good choice is often the time dimension (e.g. ‘Year’), which leads to a first division according to that axis. Of secondary importance are location, publicity, and so on, according to which axis the user will update her query in a next phase. In this phase, the user also makes tryouts of factors, since she is not confident with the factors that influence all products. Contrarily, when working on daily and weekly sales reports of the one product branch she is responsible for, she knows what to query due to her experience.

When working on the management reports, the user asks the data to be represented in a set of bar charts, since reporting styles are fixed in the company. On the other hand, when researching sales factors of products, the user likes to have the data also in numerical table format, since quite often when taking the train home from work, she wants to take a look again at the data and do offline tryout manipulations searching for sales factors on her smartphone or tablet.

From this scenario, we note the following:

- *Querying is contextualized:* User query behavior depends on the context of the user. For example, the order of querying depends on the specific activity the user is performing (management report or market research).
- *Querying is preferenced:* Depending on the context, the preferences of the user also change. For example, the visualization preference changes according to activity.
- *Query structure is similar:* While performing an activity, the structure of a series of queries stays the same. For example, the user checks for factors that influence sales by first querying according to time of year, then location, and so on. So although the specific content of the queries changes, the order in which dimensions are treated in subsequent queries stays the same.

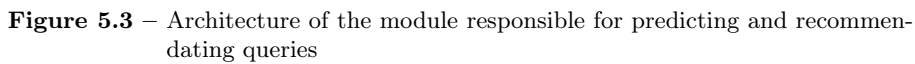
Thus for predicting the most likely next query, context and preferences have to be taken into account together with the current query, as will be described in section 5.4.4 on page 114. However, since query structures can be similar, as illustrated in the last point above, it is important to cluster similar queries for making predictions, which we present in section 5.4.3 on the next page.

5.4.2 Architecture

The first element in our architecture is the context manager (CM), which purpose is to hide the context management complexity by providing a uniform way to access context information. Context information is stored in a knowledge database using a context model. User preferences are also part of this context manager. The query translation engine (QT) is responsible for the automatic transformation of a natural language query to a conceptual query. Next, the query processor (QP) is in charge of processing a conceptual query. Such query represents user's search and preferences, represented in a structure (see for example Q in section 5.2.1 on page 99), according to a specific template. The QP enriches this query with context obtained from CM. This enriched request, represented in MDX format, is then transferred to the query execution module (QE). Then, the query execution module allows the satisfaction of the immediate query by executing it taking into account the given context. Next, the learning module (LM) is responsible for dynamically determining the user's behaviour model (classification step) from the recognized clusters representing similar user's situations (clustering). The user's behaviour model, learned and maintained by the LM, will be then used by the prediction module. Finally, the prediction module (PM), guided by the user's query, preferences and context, is based on the results from the discovery process previously stored in the past history of user queries. From this data, the PM is able to anticipate the future user's needs and to predict, and then execute in a proactive manner, the next query. A triplet

$$\langle \text{query, preferences, context} \rangle \quad (5.8)$$

representing the user situation S is sent to the prediction module. From the user's behaviour model inferred by the LM, PM will determine the user's most



Two main processes compose this query prediction mechanism: the learning process and the prediction process, as illustrated in figure 5.3. In the learning process, similar situations are grouped into clusters during the clustering step, as a way to reduce the size of the history log by looking for recurring situations. A *situation* is defined as a triplet

$$\langle \text{intention, context, preferences} \rangle \quad (5.9)$$

5.4.3 Query Clustering

The first step of our mechanism is the clustering of users' queries. Indeed, as the history log contains a trace of all observed queries of a user for a given situation, it is likely that some of them are similar. Since the size of this history in a dynamic environment can be quite large, clustering similar queries for a user represents an appropriate solution to reduce the data size and arrive to a feasible number of states, which are not too general and not too specific. Also,

the analysis of the clusters allows a better definition on users habits, which can improve the accuracy of our prediction mechanism. The input to this step are vectors representing users queries stored in the history. The main task of the clustering is to detect recurrent and similar queries from all the queries situations in the history. In fact, the clustering is responsible to determine the query that is the closest to a set of queries. This provides us a powerful mechanism to evaluate users' queries, as a user can express the same query in a slightly different way by using queries that have a sufficient similarity.

To calculate the similarity between two queries we use the distance metric defined in the previous section. To build the clusters, we start with randomly selecting x queries from the query logs and making sure that a minimal distance exists between them. They serve as the seeds for the clusters. Then, we treat each query of the logs, and assign it to the cluster for which the associated seed has the minimum distance. In that way, clusters of similar queries are obtained. The clusters are used for recommendation purposes. A BI user that is exploring a dataset and performs a query will possibly be interested in similar queries, so using the distance metric we identify the query cluster which is most similar to the current user query, and select a number of queries contained in that cluster to recommend to the user.

Context information which we consider relevant for our purposes considers, but is not limited to, the following items:

- *User role*: This refers to whether the user is for the moment acting as (real time) decision maker in a business process, or whether she is manager, only interested in overview and periodically generated reports.
- *Current activity*: This context item is related to the user role, but refers to the specific task a user is performing. Tasks include gathering information for periodic sales reports, querying to analyze current customer behavior and so on.
- *Active applications*: The set of applications that are momentarily actively used by the user. For example, if the user is also working on an Excel stylesheet, data representation preferences in context can be inferred from this.
- *Device capabilities*: The technical and hardware specifications of the device the user is working with, or by extent to the surrounding devices, can be taken into account. For example, data representation preferences will be different if the user is working with a smartphone, a tablet or with a desktop computer.
- *Location*: The current location of the user can influence her needs or preferences with regards to geographical information like points of interests, etc.

5.4.4 Query Prediction

We propose an approach predicting users' future query in order to proactively execute a query that can fulfill her future needs. Indeed, this approach is based on the assumption that common situations can be detected, even in dynamic and changing environment. Based on this assumption, this prediction

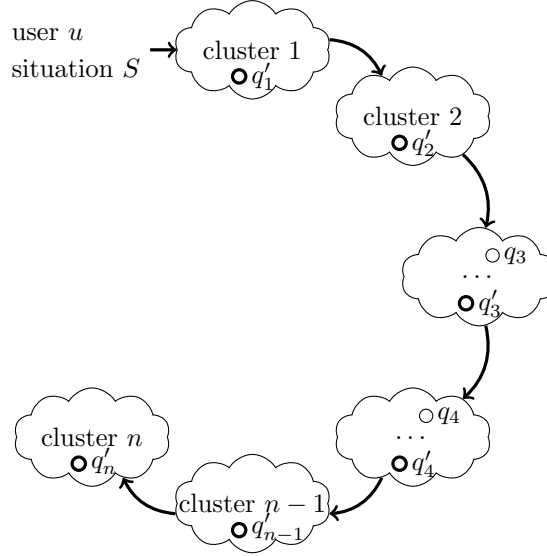


Figure 5.4 – Illustration of the Markov chains. Cloud shapes represent *clusters* of similar queries (e.g. q_3 or q_4). A particular query is selected in each cluster (e.g. q'_1, q'_2, \dots). This particular query corresponds to the most likely next user's query of any query belonging to the preceeding cluster, according to the model.

mechanism considers a time series (as a set of states being the clusters of the previous step) representing the users observed situations. These observations are time stamped and stored in a log file after each query execution (history). Thus, by analysing the history H represented by the triplet

$$\langle \text{intention, context, preferences} \rangle \quad (5.10)$$

the prediction mechanism can learn the users behaviour model M in a dynamic environment, and thus deduce the most likely next query.

We infer the most probable next query by modeling user query behavior as Markov chains³. Each query cluster represents a state in the associated Markov chain. The series of states of the system has the Markov property. A series with the Markov property is such that the probability of reaching a state in the future, given the current and past states, is the same probability as that given only the current state. So past states give no information about future states. If the machine is in state x at time n , the probability that it moves to state y at time $n + 1$, depends only on the current state x and not on past states (see figure 5.4 for an illustration of the model). The transition probability distribution can be represented as a matrix P , called a transition matrix, with the (i, j) -th element of P equal to:

$$P_{ij} = Pr(X_{n+1} = j | X_n = i) \quad (5.11)$$

³This choice is motivated by the fact that queries from query logs are supposed to be iteratively created. Thus each cluster of the Markov model corresponds to an intermediate step of data analysis.

The initial probability $Pr(X_{n+1} = j|X_n = i)$ is $\frac{1}{m}$ where m is the number of queries that can follow the current query Q_i . $Pr(X_{n+1} = j|X_n = i)$ could be updated by counting how often query Q_j is preceded by query Q_i and dividing this number by the total amount of queries that were observed as following query Q_i . This means however that the past is as important as the present. In most cases the series of queries a user performs will evolve through time. For example, if the history considers queries performed by the user during the last six months, then we want the more recent queries having relatively more influence on the user behavior model than older queries. Consequently, the transition probability function should be updated in a way that recent transitions have more importance than older ones. For that, the following exponential smoothing method can be used so that the past is weighted with exponentially decreasing weights:

$$P_{ij} = \alpha \times x_j + (1 - \alpha)P'_{ij} \quad (5.12)$$

P'_{ij} represents the old probability and x_j is the value for the choice taken at query Q_i with respect to query j . x_j is zero or one. If $x_j = 1$ then query Q_j was executed after query i , $x_j = 0$ if not. Using this method, the sum of all outgoing probabilities remains 1, as it is required for a transition probability matrix. The parameter α is a real number between 0 and 1 that controls how important recent observations are compared to history. If α is high, the present is far more important than history. In this setting, the system will adapt quickly to the behavior of the user. This can be necessary in a rapidly changing environment or when the system is deployed and starts to learn. In a rather static environment, α can be set low. In conclusion, by incorporating the learning rate, we make sure the user behavior model is dynamic and evolves together with changing habits or preferences of the user.

5.5 Summary & discussion

In this chapter, we presented results of multidimensional queries. These results are displayed in fact tables. Headers of the table are the measures and dimensions involved in the multidimensional query. Then, we have introduced the functional dependencies, as entities that belong to the same hierarchy (i.e. dimensions of different levels in the same hierarchy) or entities that are valid in the sense that they can appear together in the same query. As we will describe in chapter 6 on page 119, this relation is useful in some application, for instance for suggesting valid entities when auto-completing queries. In the answering framework that has been detailed in chapter 3 on page 57, we have mentioned conceptual queries as abstract multidimensional queries. In this chapter, we described the conceptual model of these queries, and the links between this conceptual model and the visualization of data. Indeed, one aims at rendering query results as charts or table, as it will be shown in the chapter 6 on page 119. Then, we have made explicit the translation from conceptual queries to actual implementation of query languages (e.g. SQL or MDX). In our case, this is performed by underlying BI systems. The second part of the chapter is dedicated to personalization of multidimensional queries. Therefore, we have integrated some existing approach for qualitative personalization (the MyMDX approach [24]) with other kinds of preferences (which are domain-

specific), such as chart preference. These preferences are intended to be used in some human-computer interaction process, where users formulate queries and refine them in an iterative way (i.e. in the context of a session of multidimensional queries). The proposal is thus to suggest queries of interest for the user, as being the range of most likely next queries. The prediction is made by a prediction model, that is trained based on a corpus of queries, and computed distance between multidimensional queries (expressing preferences as explained above).

The conceptual query model introduced above has been fully integrated in the answering framework (see also chapter 3 on page 57). The scenario which recommends queries of interest based on a prediction model is currently being evaluated. This topic is interesting in practice, since predicting most likely next queries and executing them in an pro-active manner reduces execution time of the system.

Chapter 6

Experiments & evaluation

Outline

6.1	Evaluating an IR system	119
6.1.1	Classic evaluation metrics	120
6.2	Evaluation proposal	120
6.2.1	US Census Bureau data	121
6.2.2	Evaluation corpus	122
6.2.3	Performance results	128
6.2.4	Evaluation results	128
6.3	Auto-completion – an experiment on the search platform	131
6.3.1	Usage statistics in BI documents	131
6.3.2	Collaborative co-occurrence measure	132
6.3.3	Query expansion	133
6.3.4	Results of the experimentation	134

In the area of IR, there are many methods and techniques for achieving a similar goal. For instance, a popular survey [3] about NL interfaces to structured data (which is a tiny area within IR) reports not less than six ranges of approaches for translating NL in database queries. As a result, much effort has been devoted to promoting evaluation metrics on the one hand, and campaigns on the other hand, such that the wide range of systems can be compared.

We present in section 6.1 classic evaluation metrics and campaigns for IR. Then, we describe section 6.2 on the following page our evaluation protocol and the results that we got. Besides the Q&A system itself, we detail in section 6.3 on page 131 the experiment that we led in the context of the search platform.

6.1 Evaluating an IR system

Evaluating an IR system is not an easy task. Indeed, considering classic metrics of *precision* and *recall* (see section 6.1.1 on the following page), we believe that for the sake of objectivity, queries from the test corpus should not be formulated by users who know well the data schema. Moreover, there is no dataset nor goldstandard queries that can be used besides those from TREC evaluation, but are not suitable for BI purposes.

6.1.1 Classic evaluation metrics

Main evaluation metrics in IR are *precision* and *recall* and their variants. Their definition is given below.

Precision and recall

Precision, on the one hand, measures relevancy taking into account the total number of retrieved documents, and thus estimates the brevity of the system's response. It is defined as:

$$p = \frac{|\{\text{retrieved documents}\} \cap \{\text{relevant documents}\}|}{|\{\text{retrieved documents}\}|} \quad (6.1)$$

Recall, on the other hand, measures relevancy but in terms of completeness: the more relevant results are returned, the better the recall metric is. It is defined as:

$$r = \frac{|\{\text{retrieved documents}\} \cap \{\text{relevant documents}\}|}{|\{\text{total relevant documents}\}|} \quad (6.2)$$

In many applications, recall is preferred than precision, because the precision metric is not relevant in the case of system that returns huge number of results (e.g., search systems over the Web). In other cases, precision is much more interesting than recall, for instance when there is at most one relevant document (thus recall is always 0 or 1).

Precision at k

In some cases, especially when recall is not relevant, an interesting metric is the *rank* of correct results. *Precision-at- k* (noted $p@k$) is such a metric:

$$p@k = \frac{\min(k, |\{\text{retrieved documents}\} \cap \{\text{relevant documents}\}|)}{\min(|\{\text{retrieved documents}\}|, k)} \quad (6.3)$$

It measures the precision taking into account *at best* the k first results triggered by any search procedure.

6.2 Evaluation proposal

In this section, we apply the metrics introduced in the previous section, and describe the results that we have got.

We propose an evaluation where:

- the document repository (i.e. the structured data) are external data (i.e. they are not the data that have been used for experimenting the system in the first place)
- queries are formulated by real users, but not users who took part to the implementation of the system
- the document repository can be used to compare the system to other similar system(s)

We introduce the data (i.e. the document repository) as well as the queries in the following.

Figure 6.1 – A few tables from the census dataset

6.2.1 US Census Bureau data

The public dataset from the US Census Bureau¹ (called Census dataset in the following) is composed of many demographic and economic facts. This dataset is available as a relational database which we have integrated in our DBMS (we use the SAP HanaTM database²). Figure 6.1 is an extract of some of the 48 tables that we have in our database dump. On this figure, we can see that the table and field names are not the terms that can be used by real users to query the dataset. For that reason, we have designed a multidimensional data model which defines a mapping from the database elements to query terms. As we can see on table 6.1, the same query term can be used for more than

Logic name	Data model term
HHDFAM	Family households
HINCY00_10	Household income 0\$-10,000\$
STATENAME	State
PSTATABB	State
PLS02CRT	Sulfur oxide combustion output emission rate

Table 6.1 – Comparison of logical names and terms used in the data model of the Census dataset

one database element. For instance, the term “State” maps to the table field **STATENAME** from the table **Community** and to the table field **PSTATABB** from the table **Plant**.

The Census dataset is quite voluminous. To give an idea of the size of the dataset, we provide table 6.2 on the next page the count of rows for the tables that are included in the dataset.

These data are geographic, demographic and social census data. We have integrated the Census dataset in our data warehouse and created manually the data schemas corresponding to these dataset.

¹See <http://www.census.gov>.

²See <http://www.saphana.com>.

Table	Size
DISTANCETOPOWERPLANTS	329802
FREEZIPCODEDATABASE	65535
HOUSING_AGG_AVM	44113
HOUSING_AGG_COMMUNITY_PROFILE	30223
HOUSING_DATAAMENITIES_2011	3907451
HOUSING_DATAAVM_NOV11	68844541
HOUSING_DATACOMMUNITY_PROFILE_2011	33416
HOUSING_DATADISTRICTS_Q32011	15258
HOUSING_DATAID_LOOKUP_SEP11	96658
HOUSING_DATAMEASURERESULTS_Q32011	535365
HOUSING_DATAPROGRAMS_Q32011	661936
HOUSING_DATASALESAGG_OCT11	286502
HOUSING_DATASALES_OCT11	380875
HOUSING_DATASCHOOLRATINGS_SEP11	81242
HOUSING_DATASCHOOLREVIEWS_SEP11_2	658026
HOUSING_DATASCHOOLS_Q32011	115243
HOUSING_DATAZONEON_AMENITIES_LKP_SEP11	3907451
HOUSING_DATAZONEON_GEOS_SEP11	85961
HOUSING_DATAZONEON_PROFILE_SEP11	48703
HOUSING_DATAZONEON_RELATES_SEP11	88868
HOUSING_DATAZONEON_SALESAGG_SEP11	320311
HOUSING_DATAZONEON_SCHOOLS_LKP_SEP11	71764
HOUSING_DATAZONEON_TO_BG_LKP_SEP11	246731
HOUSING_DATAZONEON_TO_ZIP_LKP_SEP11	124273
HOUSING_EGRD_EGRDPLT_V2	4998
HOUSING_EGRD_PLANT	4998
HOUSING_EGRD_PLANTMEASURES	9839
HOUSING_SAMPLE_COMMUNITY_PROFILE_2011	1757
HOUSING_SAMPLE_MEASURERESULTS_Q12011	8811
HOUSING_SAMPLE_PROGRAMS_Q12011	3002
HOUSING_VIEWS_ZIPS_2	29905
HOUSING_WALMART_RSS	33272
HOUSING_WEATHERFORECAST	1109137
WALMART_LOCATIONS	4411

Table 6.2 – Comparison of the sizes of tables from Census dataset

6.2.2 Evaluation corpus

As introduced section 6.2 on page 120, the evaluation corpus should be composed of queries that have not been formulated by the same users as those who had designed the data warehouse schema.

We present below the external collaboration platform ManyEyes™ on the one hand, and the *goldstandard queries* that have been extracted from this platform on the other hand.



Figure 6.2 – ManyEyes collaborative platform

Collaborative visualization platform

ManyEyesTM is a collaborative platform³ where users share both datasets and visualizations (e.g., charts) corresponding to these datasets. Besides, users are invited to rank datasets and visualizations. Figure 6.2 is a screenshot of the homepage of the platform. When exploring the available datasets used by contributors, we observed that many datasets come from the US Census Bureau, which we have described in section 6.2.1 on page 121.

Goldstandard queries

The dataset presented in section 6.2.1 on page 121 can be used to evaluate BI systems. Indeed, when modeling this dataset in a multidimensional schema, we end up with hundreds of dimensions and measures (see table 6.2 on the facing page). Besides, we have observed that this dataset is very popular on search systems for publicly available data sources (see popular tags on Figure 6.3 on the next page). For example, WolframAlphaTM made part of the Census data⁴ available.

We have randomly selected titles of charts corresponding to Census dataset, and suggest those titles as *goldstandard queries* that could be used as a test corpus in order to evaluate any search system. These queries have been used to evaluate our proposal (see section 6.2.4 on page 128).

The evaluation corpus is composed of the titles of the 50 best ranked visualizations (corresponding to the census dataset). This ensures that queries

³See <http://www-958.ibm.com/>.

⁴About WolframAlphaTM: see <http://www.wolframalpha.com/>. See also <http://blog.wolframalpha.com/2012/05/09/compute-american-community-survey-data-for-every-geographic-area/>.



Figure 6.3 – Popular dataset tags on ManyEyes

are not formulated by agents who designed the data schema of the warehouse. We present some of the *goldstandard* queries Table 6.2.2 on page 127. We have measured performance in terms of execution time on the one hand, and IR metrics on the other hand.

Our experiment shows that the system behaves to some extent similarly as WolframAlpha™, which is a well-proven system. However, in the following we like to discuss optimizations that would further improve our performance for the given questions. The second column of table 6.2.2 (‘updated query’) depicts modification required for the system to correctly interpret users’ requests. In the last column (‘comment’) we provide a brief explanation on how the system could be easily improved in order to correctly interpret users’ initial requests.

For instance, the second question “Home ownership by State” fails, because the term ‘ownership’ is not part of the terminology of the data warehouse; but the term ‘dwellings’ appears in some measures. Thus, basic linguistic resources (like WordNet) could be used to relate synonyms or terms with similar meanings. The fifth question (“And the whitest name in America is”) also requires little effort to be understood by the system. Indeed, the base form of the word ‘whitest’ is ‘white’ (which is known to the system). Thus adding a stemming component would lead to a successful answered question in that case.

An interesting question is “Where are rich people?”. It would require a little more effort in order to be correctly processed by the system. To answer this question, it requires to attach additional semantics to the data warehouse to determine locations that would be recognized by the term ‘where’. In addition, one would configure a range filter (e.g, using natural language patterns) to declare the meaning of ‘rich’ (i.e. a certain income range). The question “of Americans covered by health insurance” is of similar kind, because the term ‘cover’ can be translated to a filter on the fact table for “health insurance”. The question “500MW+ Power Plants” would need a special natural language pattern to correctly parse the expression “500MW+”.

Query	Updated query	Entities	Comment
State Population Change		{State, Population change}	
Home Ownership by State	Owner-occupied dwellings by state	{State, Owner-occupied dwellings}	home ~ dwellings
USA States information		{State}	
Generation Y in 2010 (Ages 10-32)	population by year for ages 10-32	{Year, Population}	generation ~ demographic information
And the whitest name in America is	names white percent	{Name, White percent}	whitest ~ white
40+ Population Projections by Age		{Age, Population projection}	
Average Time Spent Commuting by State		{State, Median travel time to work}	
Percent Hispanic by State		{State, Hispanic population}	
Population by ethnicity	population by race	{State, White race count, Other race count, American indian Eskimo and Aleut race count, Asian and pacific islander race count, Black, race count}	ethnicity ~ race*
Change in city & town populations		{Town name, Population change}	
Population		{State, County, Town name, Population}	
Domestic Net Migration		{State, Domestic net migration}	
Population (by County)		{County, Population}	

US surnames	US names	{Count, Name}	surnames ~ names
Age distribution by US population		{Age, Male number, Female number}	
Where are rich people?	highest household income	{State, Average household income, Median household income}	rich ~ household income
People covered and not covered by Health Insurance by State		{State, Covered}	
Southeast Asian American Population by US County		{County, Asian and Pacific islander race count}	
Black vs. white college experience percentage by state		{State, Percent white males with at least some college, Percent black males with at least some college}	
Dirtiest states from coal pollution	Plant name and carbon dioxide emissions	{State, Carbon dioxide emissions}	dirtiest ~ emissions
50MW + Power Plants	Plant name with nameplate capacity > 500MW	{Plant name, Nameplate capacity}	500MW + ~ nameplate capacity
Emissions Per State Per Capita		{State, Methane emissions, Nitrogen oxide emissions, Mercury emissions}	
US Violent Crime		{Crime rate}	
Deaths per Year in the United States		{Cause, Per year}	
Home Valuation of Major Cities in US		{City, Min valuation, Max valuation}	
of Americans without health insurance	Americans not covered per state	{State, Without health insurance}	health insurance ~ covered
Percent of men in Mass., 15 and over, never married		{Percent men}	

Marriages in the United States per 1,000 women by state	{State, Marriage rage per 1,000 women}	
Percentage of population aged 85+, by county	{County, Percent population aged 85+}	
Population by Age	Estimate by age	population ~ numeric estimate
Civilians Employed by Occupation	Amount by category	occupation ~ category
Percent of Population 18+ (by state)	{State, Percent population of age 18+}	
Population categorized by ages and areas	Population categorized by ages and states	areas ~ states

6.2.3 Performance results

We have measured the processing time of the overall answering system (see figure 6.4). On this figure, we represent the processing time before rendering the charts (plots “•”) and after rendering the charts (plots “x”).

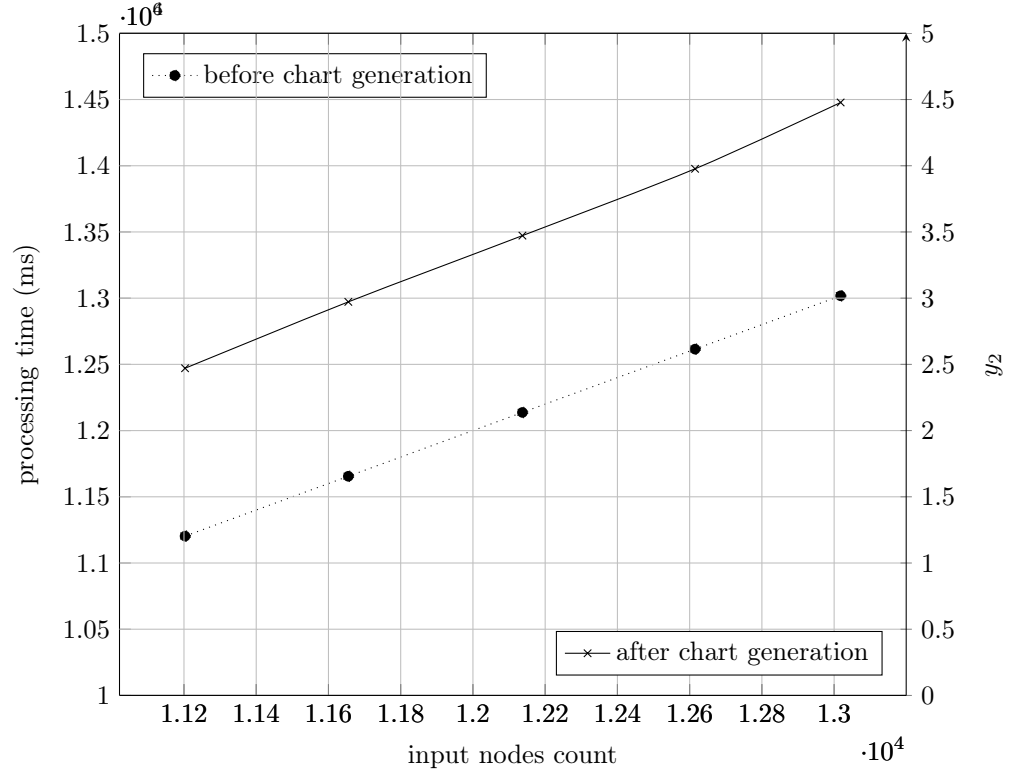


Figure 6.4 – Processing time before and after the chart generation process as a function of the schema input nodes count

On this figure, we see that as expected the processing time seems to be a proportion of the size of the graph used as input of the pattern matching algorithm. The part of the execution time dedicated to rendering the chart is approximatively a third of the global execution time. This is due to the fact that datasets that are rendered as charts are too voluminous.

6.2.4 Evaluation results

We consider the following protocol:

1. Build a data schema for a goldstandard dataset (presented in previous section)
2. Run the queries of the test corpus, and compute following evaluation metrics:
 - a) average precision for all goldstandard queries

- b) precision-at- k (or precision@ k)
- c) overall processing time, and before chart generation of these queries

Recall is not a metrics of interest in our case, because the goldstandard queries correspond to exactly one chart, i.e. one database query. We thus consider a measure derived from precision called *success at k* that measures how far is the first relevant answers within the list of results. The advantage of this protocol, is that the considered queries during the evaluation (i.e. the test corpus) have been formulated by real users. The major drawback is that many of such queries contain noise (first because the “queries” were not supposed to be queries, but titles). Therefore, we have only considered titles corresponding to actual data in the dataset.

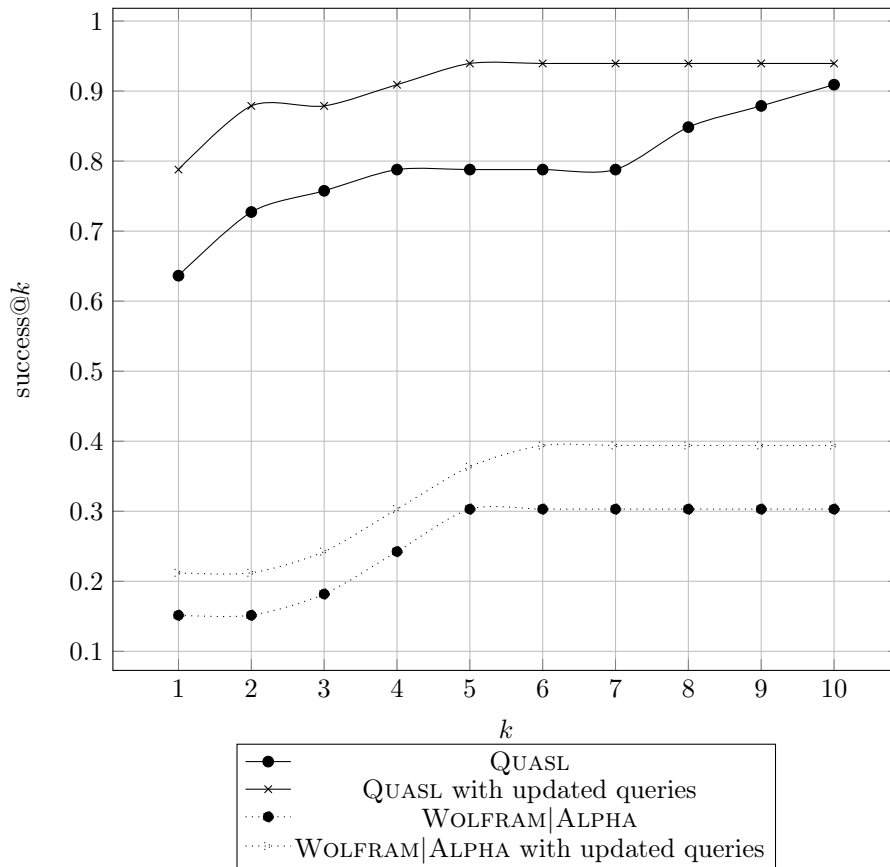


Figure 6.5 – Success of answering goldstandard queries compared to WolframAlpha™

Figure 6.5 compares the success for k varying from zero to ten for the answering system and WolframAlpha™. *Corrected* success stands for results where the query has been modified in such a way that the system can better respond. For instance, we have observed that WolframAlpha™ provides better results if some queries are prefixed or suffixed with “US” or “Census” to explicit a restriction to a subset of available data (“US”) or to the dataset

itself (“Census”). Given the goldstandard queries, our system answers better than WolframAlpha™. However, our observation is that one of the reasons why we perform better is that WolframAlpha™ does not include the whole Census dataset. Therefore, we have computed a secondary success measure, which takes into account if the dataset is known or not (i.e. if the system is able to answer the question). For WolframAlpha™, this has been determined by reformulating the questions several times until the expected result comes up. If not, we considered the dataset unknown by the system. The results have been

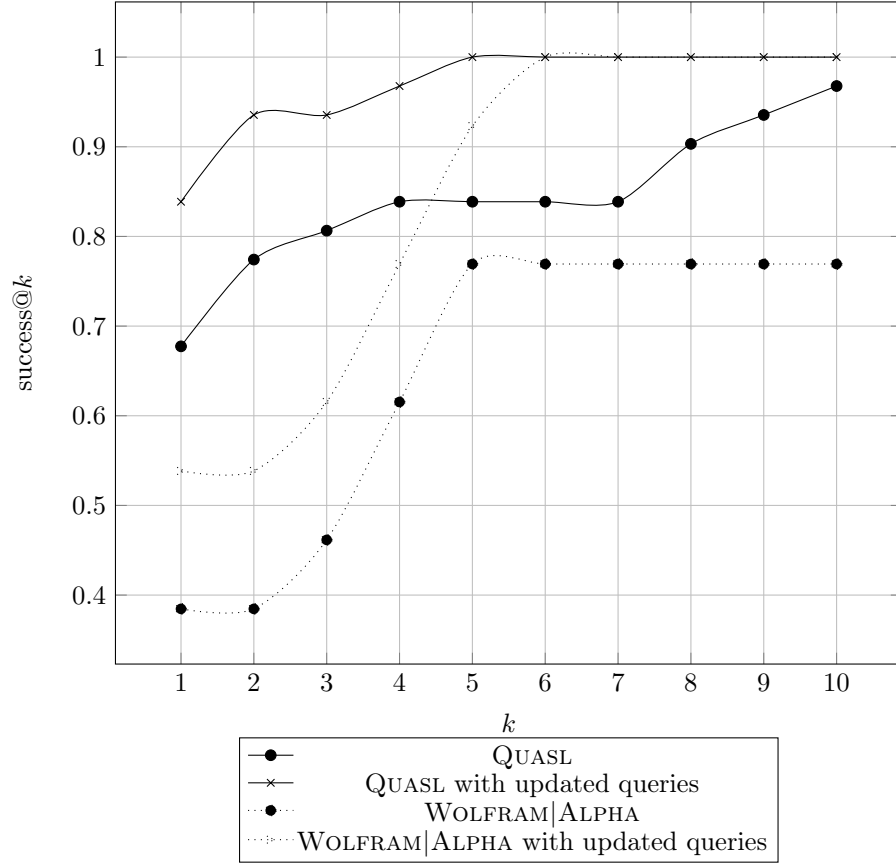


Figure 6.6 – Variant of success of answering goldstandard queries compared to WolframAlpha™

plotted Figure 6.6. On this figure, we see that WolframAlpha™ performs better (under the assumption presented above) from $k = 4$. This can be explained by the fact that WolframAlpha™ has a better average precision than QUASL (see table 6.4). Table 6.4, values have been computed based on the gold-standard

QUASL	WolframAlpha™
0.26	0.43

Table 6.4 – Average precisions of QUASL and WolframAlpha™

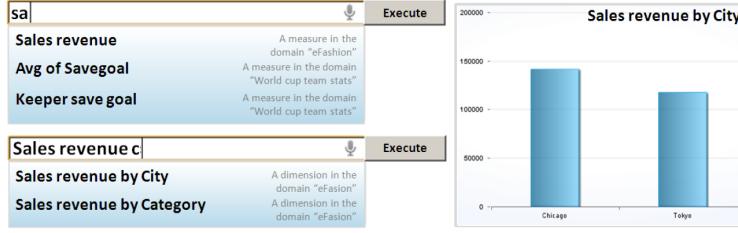


Figure 6.7 – Search-box of the answering system’s front-end with an auto-completion implementation based on collaborative metrics

queries. The definition of average precision can be found in [48]. It roughly corresponds to the proportion of good answers in the k first search results.

6.3 Auto-completion – an experiment on the search platform

Besides the answering system itself (described in chapter 3 on page 57), we have investigated *auto-completion*, i.e. the technique that allows to automatically suggest expected words or keywords in the search box, while the user is typing her query.

The front-end application of the system is composed of a search box, where users are supposed to enter their query (see figure 6.7). This experiment aims at providing suggestions on how to complete the current user’s query (before the user validates the query). The problem of auto-completing a query is formalized as follows [65]:

$$QE : (u, q, params) \mapsto \{(q_1, r_1), \dots, (q_n, r_n)\} \quad (6.4)$$

where (q_i, r_i) is the collection of scored queries such that for all i from 1 to n , $|q_i| = |q| + 1$. The idea in this formulation is to find candidate *entities* (i.e. dimensions and measures) that are best associated with query q .

6.3.1 Usage statistics in BI documents

Functional dependencies and hierarchies previously presented provide very structural knowledge regarding associations between BI entities. Beyond this, some BI platforms propose repositories of documents like reports or dashboards which can be used to compute actual usage statistics for measures and dimensions. This kind of information is extremely valuable in our use case, since query expansion implies to find the best candidate to associate to a given set of measures and dimensions.

Structure of BI documents and co-occurrences

We use the structure of BI documents to define co-occurrences between measures and dimensions. For instance, BI reports are roughly composed of sections which may contain charts, tables, text areas for comments, etc. Charts and tables define important units of sense. Measures and dimensions associated

in a same table/chart are likely to be strongly related and represent an analysis of specific interest to the user. Similarly, dashboards can be composed of different pages or views which contain charts and tables. More generally, any BI document referencing measures and dimensions could be used to derive consolidated co-occurrences or usage statistics.

Personal co-occurrence measure

BI platforms provide access control rules for business domain models and documents built on top of them. Consequently, different users may not have access to the same models and at a more fine-grained level to the same measures and dimensions. Besides, repositories contain documents generated by and shared (or not) between different users of the system. As a result, the measure of cooccurrence that we define in this section is inherently personalized. Let us consider a user u and let $occ_u(e_1)$ denote the set of charts and tables – visible to the user u – referencing a BI entity e_1 , measure or dimension. Note that in the following we only consider compatible entities (i.e. $e_1, e_2 \in d$).

We define the co-occurrence of two entities e_1 and e_2 as the Jaccard index of the sets $occ_u(e_1)$ and $occ_u(e_2)$:

$$cooc_u(e_1, e_2) = J(occ_u(e_1), occ_u(e_2)) = \frac{|occ_u(e_1) \cap occ_u(e_2)|}{|occ_u(e_1) \cup occ_u(e_2)|} \quad (6.5)$$

The Jaccard index is a simple but commonly used measure of the similarity between two sample sets.

6.3.2 Collaborative co-occurrence measure

Cold-start users and coverage

In recommender systems, the *coverage* is the percentage of items that can actually be recommended, similar to the recall in information retrieval. Formula 6.5 presents a problem for *cold-start* users, i.e. those new to the system. Indeed, these users do not have stored documents from which co-occurrences can be computed. Collaborative recommender systems introduce the contribution of other users in the item scoring function to improve the system's coverage and enable the exploration of resources previously unknowned (or unused) by the user. A simple approach consists in using a linear combination of the user-specific value and the average over the set of all *users*.

Using the social/trust network

The simple approach previously described broadens the collaborative contribution to “the whole world” and all users have the same weight. Trust-based recommender systems have illustrated the importance of considering the user's social network and, e.g., favoring users close to the current user [36]. Narrowing the collaborative contribution down to close users presents benefits at two levels: (a) results are more precisely personalized and (b) potential precomputation is reduced. Let us note $SN(u)$ the set of users in u 's social network which can be filtered, e.g., to keep only users up to a certain maximum distance. We propose the following refined co-occurrence measure, where α and β

are positive coefficients to be adjusted experimentally such that $\alpha + \beta = 1$:

$$cooc(u, e_1, e_2) = \alpha \cdot cooc_u(e_1, e_2) + \frac{\beta}{|SN(u)|} \cdot \sum_{u' \in SN(u)} \frac{1}{d(u, u')} cooc_{u'}(e_1, e_2) \quad (6.6)$$

This measure $cooc(u, e_1, e_2)$ is defined for entities e_1 and e_2 exposed to the user u by access control rules. The contribution of each user u' is weighted by the inverse of the distance $d(u, u')$. Relations between users can be obtained from a variety of sources, including popular social networks on the Web. However, this does not necessarily match corporate requirements since users of the system are actual employees of a same company. In this context, enterprise directories can be used to extract, e.g., hierarchical relations between employees. Clearly, other types of relations may be considered but the actual construction of the social network is not of the scope of this thesis.

User preferences

We distinguish *explicit* and *implicit* preferences, respectively noted $pref_{u,expl}$ and $pref_{u,impl}$. For a given entity e , we define the user's preference function $pref_u$ as a linear combination of both preferences, for instance simply:

$$pref_u(e) = \frac{1}{2} (pref_{u,impl}(e) + pref_{u,expl}(e)) \quad (6.7)$$

Explicit preferences are feedback received from the user, e.g., in the form of ratings (in $[0, 1]$) assigned to measures, dimensions. Let us note $r_{u,e}$ the rating given by u to e and \bar{r}_u the average rating given by u . We define $pref_{u,expl}(e) = r_{u,e}$ if u has already rated e , and $pref_{u,expl}(e) = \bar{r}_u$ otherwise.

Implicit preferences can be derived from a variety of sources, for instance by analyzing logs of queries executed in users' sessions [20]. In our case, we consider occurrences of BI entities in documents manipulated by the user as a simple indicator of such preferences:

$$pref_{u,impl}(e) = \frac{|occ_u(e)|}{\max_{e'(|occ_u(e')|)} |occ_u(e')|} \quad (6.8)$$

6.3.3 Query expansion

The aim of our system is to assist the user in the query design phase by ordering suggestions of measures and dimensions she could use to explore data. When she selects a measure or a dimension, it is added to the query being built and suggestions are refreshed to form new consistently augmented queries.

Ranking

To complete a given query $q = \{e_1, \dots, e_n\}$ with an additional measure of dimension, we need to find candidate entities and rank them. Candidate entities c_j , $j \in [1, p]$ are those defined in the same domain and compatible with every e_i determined using functional dependencies. We then use the following personalized function to rank each candidate c_j :

$$rank_u(c_j, q) = \begin{cases} pref_u(c_j) & \text{if } q = \emptyset \\ pref_u(c_j) \cdot \frac{1}{n} \sum_{i=1}^n cooc(u, c_j, e_i) & \text{otherwise} \end{cases} \quad (6.9)$$

Measure	Dimension	Co-occurrence
Sales revenue	Quarter	0,38
	State	0,25
	Year	0,25
	Category	0,25
	Lines	0,22

Table 6.5 – Top-5 dimensions that most co-occur with the “Sales revenue” measure

The query expansion component can thus be defined as:

$$QE : (u, q, params) \mapsto \{(q_1, rank_u(c_1, q)), \dots, (q_p, rank_u(c_p, q))\} \quad (6.10)$$

Parameters

Beyond ranking, suggestions of the query expansion component can be fine-tuned using various parameters:

- The maximum number of results.
- The type of suggested entities can be limited to measures and/or dimensions.
- The domain can be restricted to a list of accepted models.
- Suggested dimensions can be grouped by and limited to certain hierarchies.

This may be used to reduce the number of suggestions and encourage the user explore varied axis of analysis.

6.3.4 Results of the experimentation

We consider the query designer which simply presents a search text box to the user (see figure 3.3 on page 61). As she types, candidate measures and dimensions are proposed to the user as auto-completion suggestions. Figure 6.7 on page 131 shows measures (from distinct domain models) suggested when the user starts typing ‘sa’: ‘Sales revenue’, ‘Avg of savegoal’ and ‘Keeper save goal’. On the right side of the figure, the user has selected the first suggestion (i.e. ‘Sales revenue’) and keeps typing ‘c’. The system suggests the two dimensions ‘City’ and ‘Category’. The auto-completion initialization requires that the user roughly knows the names of objects she wants to manipulate, which may be a barrier to adoption. To help her get started and explore available data, suggestions can be surfaced to the user before she even starts typing. For instance, the most commonly used measures and dimensions of various domain models could be suggested to start with. Table 6.5 presents the five dimensions that most co-occur with a given measure named Sales Revenue.

Chapter 7

Conclusion

Outline

7.1	Summary	135
7.2	Challenges	138
7.2.1	Personalized patterns	138
7.2.2	Linguistic coverage	138
7.2.3	Prediction model	138

First, we sum up our contribution to the state of the art. Secondly, we discuss our results, and suggest follow-up research topics as well as perspectives to the work presented in this thesis.

7.1 Summary

Business users of today's retrieval systems face the problem of finding relevant documents in a short time. Indeed, there are many access points where information can be found (e.g. applications like the corporate portal, wikis, forums, etc.) and these applications do not co-operate correctly (e.g. underlying query languages are different, and documents are of different types). More specifically in BI, today's retrieval systems are able to offer reports or dashboards that have been annotated beforehand, but annotating these documents is a tedious task. Users may also want to execute queries over data warehouses, because the information they are looking for may not already be in a report or a dashboard. A range of retrieval systems on the Web are dedicated to seeking for information over structured data belonging to various domains (such as WolframAlpha). These systems are dedicated to non-expert users (standard users); and provide not only one answer (as it would be in Q&A systems in most of the cases) but a set of answers, corresponding to each interpretation of user's question (different interpretations correspond roughly to the domain). In response to the issue mentioned beforehand, that finding information in corporate settings is still a pain, retrieval systems in BI have emerged in conjunction with advances in visualization techniques aiming at delivering concise answers related to huge amount of data.

Numberous approaches and techniques were developed in order to interface databases in NL, that we have sum up in chapter 2 on page 15. In particular, systems which aggregate different data sources and which are domain-independant focus today’s researchers’ attention (more specifically WATSON [16]). A typical approach consists in translating NL in an intermediate query language (called *pivot* language in the NLP community), which is then translated in the target (database) query language. Two other very recent systems dedicated to BI questions, have also focused our attention are SODA [8] and SAFE [51]. The former system is a keyword-based search system over data warehouses. It uses some kinds of patterns to map keywords and some operators in the user’s query to rules to generate SQL fragments. It integrates various knowledge sources like a domain ontology etc. However, this system does not focus on “using natural language processing to understand the input” [8], nor any contextual information that should be taken into account in modern system [33]. The latter system is an answering system dedicated to mobile devices in the medical domain. It uses patterns in order to relate keyword queries and NL queries. It uses semantic technologies to overcome the problem of matching users’ terminology terms and database terms. They propose a variant of auto-completion, they suggest terms and relationships from the ontology (and not only from the database). While authors insist on the necessity of getting answers rapidly due to the domain (medical domain), more than 30% of questions presented in their evaluation are answered in more than 50s.

The system that we propose is in the form of a search interface available on desktop as well as on mobile devices, and has been described in chapter 3 on page 57. Users type or pronounce queries in their own terms, and search results are eventually displayed in a dashboard. We focus in this thesis on the translation of questions in NL in database queries. The system can be easily configured:

- any data warehouse or more generally any repository of structured data can be interface to our system with limited configuration effort. This corresponds to the implementation of a new ‘search plugin’
- the system has been implemented for three languages (english, german and french) but we are confident on the feasibility of supporting additional languages. The main limitation is imposed by the named entity recognizer, that supports more than 30 languages. In terms of effort, supporting a new language means translating data schema in the new language, integrating named entity recognition rules for the new language and translating sentences to be synthesized by the siri-like mobile application.

We have implemented an end-to-end Q&A framework for BI. This framework can be augmented with various search plugins, i.e. for supporting additional data sources. Moreover, the different algorithms involved in this framework have been implemented in a parallel way, in order to optimize the execution time required by the overall system.

In chapter 4 on page 75, we define the *patterns* that are involved in the translation of NL questions in conceptual queries (i.e. pivot language mentioned above). These patterns define constraints that must be satisfied:

- in user's question: words used by the user and (database) entities in her query
- by the user's context: device which is used to access the search application, user's profile (like her job title, her preferences, etc.)

Some of these constraints can be optional (for instance, a single pattern may define that there should be *at least* one dimension in user's query). Each pattern triggers translation rules with *slots*, to be filled with actual values of user's query. This approach is inspired from the form-based Q&A approach, but is much more flexible, since:

- there is no strong requirement in the order according to which the *features* must be listed in the pattern
- the constraint problem is solved by a NER, which allows variant (such as synonyms), and not only well-formed terms with respect to the data schema of the warehouse

The query model has been presented in chapter 5 on page 97. This model is a high-level representation of any multidimensional query, that can be automatically translated in MDX or SQL (depending on the underlying data warehouse).

We have evaluated our proposal in chapter 6 on page 119. This evaluation consists in an original approach that we believe could be generalized to any retrieval system in the BI domain. Indeed, in addition to an evaluation protocol, we suggest some *gold-standard* queries (formulated by real users) which formulation vary from precise to very vague. Some additional metrics have also been introduced, in particular distance metrics between patterns, which could be the starting point to future work on machine learning.

The contribution to the state of the art can be thus summarized as follows:

- a set of constraint-matching algorithms, as a solution proposal to the graph-matching problem already observed in keyword-based search interfaces to structured data
- a framework for Q&A in the BI domain that can be easily configured for new domains of application and for new kinds of data warehouses (e.g. via the implementation of new *search plugins*)
- an evaluation framework for BI retrieval systems, through evaluation metrics and gold-standard queries from the Census dataset
- a basis for machine learning algorithms that would improve the system that we have built
- a query model that we believe could be successfully associated to a prediction model, and whose main goal would be to decrease significantly the response time of the entire system in executing pro-actively well-chosen queries

We detail in the next section our proposal for future work, in order to improve and go further the work presented in this thesis.

7.2 Challenges & problems to be addressed in future works

All the problems related to mapping NL queries to structured queries for data warehouses have not been solved. We review below some research directions to be considered as follow-up to the work presented in this thesis.

7.2.1 Personalized patterns

In our work, we do not consider personalized patterns, because we haven't seen any case where a pattern should be personalized. The case where it might be of interest, is where the system could learn users terminology, and build personalized patterns to this end.

7.2.2 Linguistic coverage

The current system has a good linguistic coverage, according to the evaluation results presented in the previous chapter, but is currently limited. An immediate follow-up to the current implementation is then to use machine learning techniques to improve the current capabilities over time, basing for instance on failing queries from query logs.

7.2.3 Prediction model

We have introduced in section 5.4 on page 111 a module performing query prediction as an application to the personalization of multidimensional queries. The implementation of this component is still in progress, and therefore we suggest to further investigate both prediction and recommendation of multidimensional queries.

Appendix A

Pattern

We reproduce here an example of a pattern.

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX data-binding: <http://viceversatech.com/rdfbeans/2.0/>
4 PREFIX bquery-model: <http://research.sap.corp/pattern/>
5 PREFIX grepo: <urn:grepo#>
6 PREFIX query-tree: <urn:grepo/query-tree#>
7 PREFIX slayer: <urn:grepo/slayer#>
8 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
9
10 CONSTRUCT {
11   bquery-model:bquery data-binding:bindingClass
12   "com.sap.research.questionanswering.pattern.model.BusinessQuery"
13   .
14   bquery-model:dimensionProjectionItem
15     data-binding:bindingClass
16     "com.sap.research.questionanswering.pattern.model.DimensionProjectionItem"
17   .
18   bquery-model:projectionItem data-binding:bindingClass
19     "com.sap.research.questionanswering.pattern.model.ExpressionProjectionItem"
20   .
21   bquery-model:measureReference data-binding:bindingClass
22     "com.sap.research.questionanswering.pattern.model.MeasureReference"
23   .
24   bquery-model:measure data-binding:bindingClass
25     "com.sap.research.questionanswering.pattern.model.Measure"
26   .
27   bquery-model:memberClass data-binding:bindingClass
28     "com.sap.research.questionanswering.pattern.model.Dimension"
29   .
30   bquery-model:memberFilter data-binding:bindingClass
31     "com.sap.research.questionanswering.pattern.model.MemberFilter"
32   .
33   bquery-model:vectorMemberset data-binding:bindingClass
34     "com.sap.research.questionanswering.pattern.model.VectorMemberSet"
35   .
36   bquery-model:member data-binding:bindingClass
37     "com.sap.research.questionanswering.pattern.model.Member"
```



```

22  bquery-model:expressionProjectionItem
    data-binding:bindingClass
    "com.sap.research.questionanswering.pattern.model.ExpressionProjectionItem"

23  _:bquery rdf:type <http://research.sap.corp/pattern/bquery> .
24  _:bquery bquery-model:bquery/analysisType "comparison" .
25  _:bquery bquery-model:bquery/dataSource ?universeName .
26  _:bquery bquery-model:bquery/hasProjectionItem
    _:dimensionProjectionItem_1 .
27  _:bquery bquery-model:bquery/hasProjectionItem
    _:measureProjectionItem_1 .
28  ?pattern <urn:grepo/pattern#matches> _:query .
29  ?pattern rdfs:label ?patternLabel .
30  ?pattern rdf:type ?patternType .
31  _:query rdf:type <urn:grepo/pattern#Query> .
32  _:query <urn:grepo/pattern#hasBusinessQuery> _:bquery .
33  _:bquery bquery-model:bquery/hasFilterClause
    _:memberFilterClause .
34  _:memberFilterClause rdf:type bquery-model:memberFilter .
35  _:memberFilterClause bquery-model:memberFilter/hasMemberSet
    _:memberSet .
36  _:memberSet rdf:type bquery-model:vectorMemberset .
37  _:memberSet bquery-model:typedMemberset/hasMemberClass
    _:memberDimension .
38  _:memberSet bquery-model:vectorMemberset/hasMember
    _:member_1 .
39  _:dimensionProjectionItem_1 rdf:type
    bquery-model:dimensionProjectionItem .
40  _:dimensionProjectionItem_1 bquery-model:dimension
    _:dimension_1 .
41  _:measureProjectionItem_1 rdf:type
    bquery-model:expressionProjectionItem .
42  _:measureProjectionItem_1
    bquery-model:expressionProjectionItem/hasExpression
    _:measureExpression .
43  _:measureExpression rdf:type bquery-model:measureReference .
44  _:measureExpression bquery-model:measureReference/hasMeasure
    _:measure_1 .
45  _:measure_1 rdf:type bquery-model:measure .
46  _:measure_1 rdfs:label ?measureLabel_1 .
47  _:measure_1 bquery-model:measure/hasMeasureId
    ?measureOriginUri_1 .
48  _:measure_1 rdfs:isDefinedBy ?measureUri_1 .
49  _:dimension_1 rdf:type bquery-model:memberClass .
50  _:dimension_1 rdfs:label ?dimensionLabel_1 .
51  _:dimension_1 bquery-model:memberClass/hasDimensionId
    ?dimensionOriginUri_1 .
52  _:dimension_1 rdfs:isDefinedBy ?dimensionUri_1 .
53  _:member_1 rdf:type
    <http://research.sap.corp/pattern/member> .
54  _:member_1 rdfs:label ?memberLabel .
55  _:member_1 bquery-model:member/hasValue ?memberLabel .

```

```

56 _:memberDimension rdfs:type
    <http://research.sap.corp/pattern/memberClass> .
57 _:memberDimension rdfs:label ?memberDimensionLabel .
58 _:memberDimension bquery-model:memberClass/hasDimensionId
    ?memberOriginUri_1 .
59 _:memberDimension rdfs:isDefinedBy ?memberUri_1 .
60 _:bquery <http://research.sap.corp/pattern#referencesEntity>
    _:measure_1 .
61 _:bquery <http://research.sap.corp/pattern#referencesEntity>
    _:dimension_1 .
62 _:bquery <http://research.sap.corp/pattern#referencesEntity>
    _:memberDimension .
63 _:bquery bquery-model:bquery/hasConfidence ?confidence .
64 }
65
66 WHERE {
67 ?pattern rdfs:label ?patternLabel .
68 ?pattern rdfs:type ?patternType .
69 ?queryUri query-tree:hasAnnotation ?annotationUri_1 .
70 ?annotationUri_1 query-tree:hasAnnotationType
    <urn:grepo/features/instances#MeasureAnnotationType> .
71 ?annotationUri_1 query-tree:referencesResource ?measureUri_1
    .
72 ?annotationUri_1 query-tree:confidence ?measureConfidence_1 .
73 ?measureUri_1 rdfs:label ?measureLabel_1 .
74 ?measureUri_1 grepo:originUri ?measureOriginUri_1 .
75 ?queryUri query-tree:hasAnnotation ?annotationUri_2 .
76 ?annotationUri_2 query-tree:hasAnnotationType
    <urn:grepo/features/instances#DimensionAnnotationType> .
77 ?annotationUri_2 query-tree:referencesResource
    ?dimensionUri_1 .
78 ?annotationUri_2 query-tree:confidence
    ?dimensionConfidence_1 .
79 ?dimensionUri_1 rdfs:label ?dimensionLabel_1 .
80 ?dimensionUri_1 grepo:originUri ?dimensionOriginUri_1 .
81 ?queryUri query-tree:hasAnnotation ?annotationUri_3 .
82 ?annotationUri_3 query-tree:hasAnnotationType
    <urn:grepo/features/instances#DimensionValueAnnotationType>
    .
83 ?annotationUri_3 query-tree:referencesResource ?memberUri_1 .
84 ?annotationUri_3 rdfs:label ?memberLabel .
85 ?annotationUri_3 query-tree:confidence ?memberConfidence_1 .
86 ?memberUri_1 rdfs:label ?memberDimensionLabel .
87 ?memberUri_1 grepo:originUri ?memberOriginUri_1 .
88 ?measureUri_1 slayer:hasUniverse ?universeId .
89 ?dimensionUri_1 slayer:hasUniverse ?universeId .
90 ?memberUri_1 slayer:hasUniverse ?universeId .
91 ?universeId rdfs:label ?universeName .
92 ?universeId slayer:hasLanguage ?language .
93 LET (?confidence := ?measureConfidence_1 +
    ?dimensionConfidence_1 + ?memberConfidence_1 + 4)
94 }

```

Listing A.1 – Pattern matching queries like “Revenue per state in 2001?”

Appendix B

Query logs

In this appendix, we reproduce a short example of query logs captured by our Q&A framework.

```
1  {
2    "userQuery": "revenue_per_city",
3    "clickedDaslResultQuery": "NonEmpty((Revenue)_IN_
      City::(Revenue.DESC));",
4    "PatternName": "1Measure_1Dimension_2Filters",
5    "Universe": "S360_CLUB_EN",
6    "Dimension": "City",
7    "Measure": "Revenue",
8    "resultType": "daslResult",
9    "currentRank": "1.34",
10   "resultopenTime": "1341495982562",
11   "resultcloseTime": "1341496006179"
12 }
13 {
14   "userQuery": "revenue_per_city",
15   "clickedDaslResultQuery": "NonEmpty((Revenue)_IN_
      Resort::(Revenue.DESC));",
16   "PatternName": "1Measure_1Dimension_2Filters",
17   "Universe": "S360_CLUB_EN",
18   "Dimension": "Resort",
19   "Measure": "Revenue",
20   "resultType": "daslResult",
21   "currentRank": "0.81",
22   "resultopenTime": "1341496007538",
23   "resultcloseTime": "1341496009286"
24 }
25 {
26   "userQuery": "revenue_per_city",
27   "clickedDaslResultQuery": "NonEmpty(([Sales_revenue])_IN_
      State::([Sales
28 _revenue].DESC));",
29   "PatternName": "1Measure_1Dimension_2Filters",
30   "Universe": "eFashionS360_EN",
31   "Dimension": "State",
32   "Measure": "Sales_revenue",
33   "resultType": "daslResult",
```

```

34  "currentRank": "0.66",
35  "resultopenTime": "1341496011790",
36  "resultcloseTime": "1341496013598"
37  }
38  {
39    "userQuery": "revenue_per_city",
40    "clickedDaslResultQuery": "NonEmpty([Sales_revenue])_IN_
      State::([Sales_revenue].DESC));",
41    "PatternName": "1Measure_1Dimension_2Filters",
42    "Universe": "eFashionS360_EN",
43    "Dimension": "State", "Measure": "Sales_revenue",
44    "resultType": "daslResult",
45    "currentRank": "0.66",
46    "resultopenTime": "1341496014521",
47    "resultcloseTime": "1341496016764"
48  }

```

Listing B.1 – Query log example

This log keeps trace of the user's query, the pattern that were used, database entities, the generated structured queries as well as clickthrough data (i.e. what time and how long results were opened by the user).

Appendix C

Multidimensional queries

C.1 Automatic generation of SQL/MDX queries

We reproduce listing C.2 the automatic SQL generation for the following query:

$$Q = \left[\begin{array}{ll} \textit{dimension} & = \{[City]\} \\ \textit{measure} & = \{(Sales\ revenue)\} \\ \textit{filters} & = \emptyset \\ \textit{ordering} & = \emptyset \\ \textit{truncation} & = \emptyset \end{array} \right] \quad (C.1)$$

The datasource is a relational database, and for that reason the query generation ends up to an SQL query.

```

1  SELECT DISTINCT
2      "EFASHION" . "OUTLET_LOOKUP" . "CITY" AS DASL_1,
3      sum( "EFASHION" . "SHOP_FACTS" . "AMOUNT_SOLD" ) AS DASL_2
4  FROM
5      "EFASHION" . "OUTLET_LOOKUP"
6  INNER JOIN
7      "EFASHION" . "SHOP_FACTS"
8  ON
9      ( "EFASHION" . "OUTLET_LOOKUP" . "SHOP_ID" = "EFASHION" . "SHOP_FACTS" . "SHOP_ID" )
10 GROUP BY "EFASHION" . "OUTLET_LOOKUP" . "CITY"
```

Listing C.1 – SQL query generation (1)

The following query:

$$Q = \left[\begin{array}{ll} \textit{dimension} & = \{[Year]\} \\ \textit{measure} & = \{(Margin)\} \\ \textit{filters} & = \{State = 'Texas'\} \cup \{State = 'NewYork'\} \\ \textit{ordering} & = \{(Year, (Margin).DESC)\} \\ \textit{truncation} & = \emptyset \end{array} \right] \quad (C.2)$$

ends up into the following technical query:

```

1  SELECT DISTINCT
2      sum( "EFASHION" . "SHOP_FACTS" . "MARGIN" ) AS DASL_1,
3      "EFASHION" . "CALENDAR_YEAR_LOOKUP" . "YR" AS DASL_2
4  FROM
```

```
5         "EFASHION" . "CALENDAR_YEAR_LOOKUP"
6 INNER JOIN
7         "EFASHION" . "SHOP_FACTS"
8 ON (
9         "EFASHION" . "SHOP_FACTS" . "WEEK_ID"
10        =
11        "EFASHION" . "CALENDAR_YEAR_LOOKUP" . "WEEK_ID"
12    )
13 INNER JOIN "EFASHION" . "OUTLET_LOOKUP"
14 ON (
15        "EFASHION" . "OUTLET_LOOKUP" . "SHOP_ID"
16        =
17        "EFASHION" . "SHOP_FACTS" . "SHOP_ID"
18    )
19 WHERE
20        "EFASHION" . "OUTLET_LOOKUP" . "STATE" IN ( 'Texas' , 'New_
                York' )
21 GROUP BY
22        "EFASHION" . "CALENDAR_YEAR_LOOKUP" . "YR"
23 ORDER BY 1 DESC
```

Listing C.2 – SQL query generation (2)

Appendix D

Application screenshots

D.1 Desktop application

Figure D.1 is a screenshot of the desktop application.



Figure D.1 – Screenshot of the desktop application

D.2 iPhone™/iPad™ application

Figure D.2 is a screenshot of the iPhone™ application. The component on the

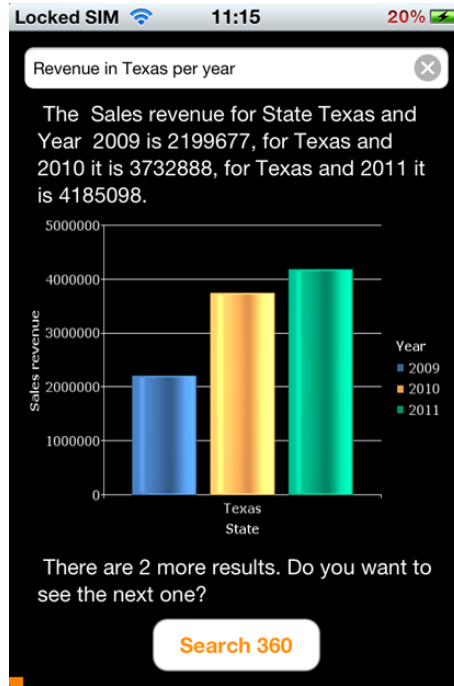


Figure D.2 – Screenshot of the iPhone™ application

top is the search box, where users are invited to type their queries. Questions can also be pronounced and recognized (the rounded rectangle entitled “Search 360” is indeed a button); in that case a colored bar shows the amplitude of the voice and let users know that voice recognition service is running and ready for use.

D.3 Search results

We have introduced section 3.1.3 page 66 different kinds of results used in the different applications. Besides the standard fact table (an example can be found in the section mentionned above), cross tables can be generated. Table D.1 is an example of cross-table. The difference with a standard fact table is that it is

<i>City</i>	<i>Store name</i>	<i>Year</i>		
		2009	2010	2011
New York	Magnolia	1,023,060.70\$	1,687,359.10\$	1,911,434.30\$
New York	5th	644,635.10\$	1,076,144.00\$	1,239,578.40\$
Miami	Sundance	405,985.10\$	661,249.80\$	811,923.60\$
Washington	Tolbooth	693,210.50\$	1,215,158.00\$	1,053,581.40\$

Table D.1 – Example of a cross-table with two hierarchies

possible to have two hierarchies (on both axis, respectively the first hierarchy projected on the level “Year” and the second hierarchy projected on the level “City”) while there can be only one hierarchy on a classic fact table.

In order to represent an additional hierarchy, the table structure is not sufficient, and *reports* can be used to render the facts. On figure D.3, we reproduce

2010		
City	[Sales revenue]	Margin
Washington	1215158,00	457230,60
Los Angeles	1581616,00	604780
Miami	661249,80	266670,00
New York	2763503,10	1104277,60
Dallas	739368,70	279651,60
Chicago	1150658,80	465477,90
Houston	1990449,20	797854,30
Austin	1003070,70	382055,70
Colorado Springs	768389,50	294482,60
San Francisco	1201063,50	471748,00
Boston	157718,70	63657,20

2011		
City	[Sales revenue]	Margin
Miami	811923,60	318131,80
Houston	2246198,40	855541,80
Chicago	1134085,40	439865,00
Colorado Springs	843584,20	309966,00
Washington	1053581,40	385414,60
Austin	1135479,10	424790,00
Dallas	803420,80	286146,40
San Francisco	1336003,30	502120,90
New York	3151021,70	1189165,50
Los Angeles	1656675,70	619367,60
Boston	887169,20	336574,10

Figure D.3 – Example of report with two hierarchies (time and geographic hierarchies)

an example report output for the query composed of following entities:

- measures *Sales revenue* and *Margin*
- geographic hierarchy with the dimension *City*
- time hierarchy with the dimension *Year*

Appendix E

Source code

We reproduce in this section the main classes that are part of our implementation.

```
1
2 public class Pattern2DaslSearchPlugin implements
   ISearchEnginePlugin {
3
4   private static Map<String , DaSlProcessor> daslProcessors =
     new HashMap<String ,
5   DaSlProcessor>();
6
7   protected HashMap<String , Object> alreadyAddedQueries;
8
9   @Override
10  public ResultIterator search(QueryTree queryTree ,
11    final SessionContext sessionContext) {
12    try {
13      IGraph queryGraph = null ,
14      featureGraph = null , stfGraph = null , geoGraph = null ,
15      timeGraph = null , slGraph = null , repoGraph = null ,
16      patternGraph = null , userGraph = null;
17
18      IGraphRepoFactory factory = (IGraphRepoFactory)
19      OSGiHelper.getOSGiService(Activator.getContext() ,
20      IGraphRepoFactory.class);
21      IGraphRepo graphRepo =
22      factory.getGraphRepo(sessionContext.getUserToken());
23
24      queryGraph = graphRepo.getGraph(QueryTreeVocab.NS_DATA);
25
26      stfGraph = graphRepo.getGraph(STFVocab.NS_DATA);
27      featureGraph =
28      graphRepo.getGraph(NlpFeatureVocab.NS_DATA);
29      geoGraph = graphRepo.getGraph(GKVocab.NS_GEOGRAPHIC_DATA);
30      timeGraph = graphRepo.getGraph(GKVocab.NS_TIME_DATA);
31      slGraph = graphRepo.getGraph(SLVocab.NS_DATA);
32      repoGraph = graphRepo.getGraph(RepoVocab.NS_UNKNOWN_DATA);
33      userGraph = graphRepo.getGraph(STFUserVocab.NS_DATA);
34      patternGraph = graphRepo.getGraph(PatternVocab.NS_DATA);
```

```

32
33     Model joinedModel = ModelFactory.createDefaultModel();
34     joinedModel = (Model) queryGraph.getHandler();
35     joinedModel = ModelFactory.createUnion(joinedModel,
36         (Model)
37         stfGraph.getHandler());
38     joinedModel = ModelFactory.createUnion(joinedModel,
39         (Model) featureGraph.getHandler());
40     joinedModel = ModelFactory.createUnion(joinedModel,
41         (Model) geoGraph.getHandler());
42     joinedModel = ModelFactory.createUnion(joinedModel,
43         (Model) timeGraph.getHandler());
44     joinedModel = ModelFactory.createUnion(joinedModel,
45         (Model) slGraph.getHandler());
46     joinedModel = ModelFactory.createUnion(joinedModel,
47         (Model) userGraph.getHandler());
48     joinedModel = ModelFactory.createUnion(joinedModel,
49         (Model) patternGraph.getHandler());
50
51     final Model model = joinedModel;
52     INodeQuery queryNode =
53         NodeQueryFactory.withType(SLVocab.CLASS_Universe);
54     Iterator<INode> universeIterator =
55         slGraph.getNodes(queryNode);
56
57     while (universeIterator.hasNext()) {
58         INode universeNode = universeIterator.next();
59         try {
60             // set up connection to universe
61             UniverseSessionHelper sessionHelper = new
62                 UniverseSessionHelper(
63                 universeNode.getAttribute(RepoVocab.PROP_DATA_originUri,
64                     null)
65             );
66             DaSlProcessor processor = new
67                 DaSlProcessor(sessionHelper,
68                 universeNode.getLabel());
69
70             daslProcessors.put(universeNode.getLabel(), processor);
71         } catch (Exception e) {
72             logger.error("Could_not_instantiate_dasl_processor_for_"
73                 + universeNode.getLabel() + "_due_to_"
74                 + e.getMessage()
75             );
76         }
77     }
78
79     final INode bundleNode = stfGraph.createNode(
80         STFVocab.CLASS_Search360PlugInSearch
81     );
82     bundleNode.setLabel("Pattern2Dasl_Search_Bundle");
83
84     final HashMap<String, INode> parameterSettings = new
85         HashMap<String, INode>();

```

```

79 parameterSettings.put(
80     "?queryUri", FakeQueryNodeGenerator.doIt(queryTree,
81         queryGraph)
82 );
83 final PriorityQueue<ScoredResultModel> constructedModels
84     = new PriorityQueue<ScoredResultModel>();
85 Thread patternExecutorThread = new Thread() {
86     @Override
87     public void run() {
88         PatternExecutor.applyPatterns(model, constructedModels,
89             sessionContext.getUserToken(), parameterSettings,
90             bundleNode
91         );
92     }
93 };
94 patternExecutorThread.start();
95
96 final PriorityQueue<BusinessQuery> bqueries = new
97     PriorityQueue<BusinessQuery>();
98 Thread objectWrapperThread = new Thread() {
99     @Override
100     public void run() {
101         BQueryObjectWrapper.getBQueryObjects(bqueries, constructedModels,
102             model);
103     }
104 };
105 objectWrapperThread.start();
106
107 BufferedResultIterator resIt = new BufferedResultIterator(
108     this.getClass().getName(), sessionContext
109 ) {
110     boolean isRunning;
111     boolean isQueueDone = false;
112
113     int NTHREADS =
114         evalConfiguration.getProperty("EVAL_MODE").toBoolean()
115         ? 1 : Runtime.getRuntime().availableProcessors() * 2;
116     final ExecutorService exec =
117         Executors.newFixedThreadPool(NTHREADS);
118     final Queue<Result> res = new
119         LinkedBlockingQueue<Result>();
120     Map<String, DaslResult> alreadyAddedQueries = new
121         HashMap<String, DaslResult>();
122     IGraphRepoFactory factory = (IGraphRepoFactory)
123         OSGiHelper
124             .getOSGiService(Activator.getContext(), IGraphRepoFactory.class);
125
126     IGraphRepo graphRepo =
127         factory.getGraphRepo(sessionContext.getUserToken());
128     IGraph slGraph = graphRepo.getGraph(SLVocab.NS_DATA);
129
130     public void run() throws Exception {
131         Thread schedulerThread = new Thread() {

```

```

122     @Override
123     public void run() {
124         while (!isQueueDone) {
125             while (bqueries == null || bqueries.peek() == null) {
126                 try {
127                     Thread.sleep(50);
128                 } catch (InterruptedException e) {
129                 }
130             }
131             while (bqueries.peek() != null) {
132                 BusinessQuery businessQuery = bqueries.poll();
133                 if
134                     (businessQuery.getDatasource().equals("$SIGNAL_DONE$"))
135                     {
136                         isQueueDone = true;
137                         break;
138                     }
139                 DaslResult daslQueryAndChartType;
140                 try {
141                     daslQueryAndChartType =
142                         BQueryTranslation.bQueryToDaslQuery(
143                             businessQuery, sessionContext.getUserToken(),
144                             slGraph, true
145                         );
146                     if (alreadyAddedQueries.containsKey(
147                         daslQueryAndChartType.getDaslQuery()
148                         + daslQueryAndChartType.getUniverse())
149                     ) {
150                         DaslResult updatedDaslResult = alreadyAddedQueries
151                             .get(daslQueryAndChartType.getDaslQuery()
152                             + daslQueryAndChartType.getUniverse());
153                         if (daslQueryAndChartType.getScore() >
154                             updatedDaslResult.getScore()) {
155                             updatedDaslResult.setScore(daslQueryAndChartType.getScore());
156                             updatedDaslResult.setPatternName(
157                                 daslQueryAndChartType.getPatternName()
158                             );
159                         }
160                     }
161                     continue;
162                 }
163                 scheduleQueryForExecution(daslQueryAndChartType);
164             } catch (Exception e) {
165                 logger.error(e.getMessage(), e);
166             }
167         }
168     }
169     exec.shutdown();
170 }
171
172 private void scheduleQueryForExecution(
173     final DaslResult daslResult) {

```

```

171     alreadyAddedQueries.put(
172         daslResult.getDaslQuery() + daslResult.getUniverse(),
173         daslResult
174     );
175     try {
176         logger.debug("executing_query="
177             + daslResult.getDaslQuery() + "_with_score_"
178             + daslResult.getScore()
179         );
180         Runnable requestHandler = new Runnable() {
181             public void run() {
182                 try {
183                     daslProcessors.get(daslResult.getUniverse()).processQuery(daslResult);
184                     res.offer(daslResult);
185                 } catch (UnsupportedOperationException e) {
186                     logger.debug("query_" + daslResult.getDaslQuery()
187                         + "_encountered_an_error"
188                     );
189                 } catch (Exception e) {
190                     logger.error(
191                         "dasl_encountered_an_error_trying_to_execute_"
192                         + daslResult.getDaslQuery() + "in_universe_" +
193                         daslResult.getUniverse()
194                         + "_from_pattern_" + daslResult.getPatternName(),
195                         e
196                     );
197                 }
198             }
199         };
200         exec.execute(requestHandler);
201     } catch (Exception e) {
202         logger.error("query=" + daslResult.getDaslQuery() + "_
203             failed", e);
204     }
205     }
206     }
207
208     @Override
209     public void remove() {
210         exec.shutdownNow();
211     }
212
213     @Override
214     public Map<String, Object> getStatistics() {
215         return null;
216     }
217
218     @Override
219     protected List<Result> moreResults(Session session) {
220         if (isRunning == false) {
221             isRunning = true;
222             try {

```



```

220         run();
221     } catch (Exception e) {
222         logger.error(e);
223     }
224 }
225 while (res.peek() == null && !hasFinishedExecution()) {
226     try {
227         Thread.sleep(50);
228     } catch (InterruptedException e) {
229     }
230 }
231
232 List<Result> intermResult = new ArrayList<Result>();
233 while (res.peek() != null) {
234     intermResult.add(res.poll());
235 }
236 return intermResult;
237 }
238
239 private boolean hasFinishedExecution() {
240     return exec.isTerminated() && res.peek() == null;
241 }
242
243 @Override
244 protected boolean hasMoreResults() {
245     if (isRunning == false) {
246         isRunning = true;
247         try {
248             run();
249         } catch (Exception e) {
250             e.printStackTrace();
251         }
252     }
253     boolean finished = hasFinishedExecution();
254     return !finished;
255 }
256 };
257
258 // trigger run method
259 resIt.hasNext();
260 return resIt;
261
262 } catch (Exception e) {
263     logger.error("", e);
264 }
265
266 return new ListResultIterator(
267     this.getClass().getName(), sessionContext, new
268     ArrayList<Result>());
269 }
270
271 public static void populatePatternGraph(
272     final IGraphRepo repo, final Model model, final INode
273     bundleNode,

```

```

272     final IGraph slGraph, final IGraph repoGraph, final
273         String queryLabel
274 ) {
275     final IGraph patternGraph =
276         repo.getGraph(PatternVocab.NS_DATA);
277     Thread populator = new Thread() {
278         @Override
279         public void run() {
280             String query = ""
281                 + "PREFIX rdf:_
282                 <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n_"
283                 + "PREFIX rdfs:_
284                 <http://www.w3.org/2000/01/rdf-schema#>\n_"
285                 + "SELECT ?query_ ?patternLabel_ ?bquery_ ?entityUri_
286                 ?universeName\n_"
287                 + "WHERE {
288                 ?pattern_rdf:type_<urn:grepo/pattern#Pattern>_._"
289                 + "?pattern_rdfs:label_?patternLabel_._"
290                 + "?pattern_<urn:grepo/pattern#matches>_?query_._"
291                 + "?query_rdf:type_<urn:grepo/pattern#Query>_._"
292                 + "?query_<urn:grepo/pattern#hasBusinessQuery>_?bquery_
293                 ._"
294                 + "?bquery_rdf:type_
295                 <http://research.sap.corp/pattern/bquery>_._"
296                 + "?bquery_
297                 <http://research.sap.corp/pattern#referencesEntity>_
298                 ?entity_._"
299                 + "?entity_rdfs:isDefinedBy_?entityUri_._"
300                 + "?bquery_
301                 <http://research.sap.corp/pattern/bquery/datasource>_
302                 ?universeName_._"
303                 + "}";
304
305             Query q = QueryFactory.create(query);
306             QueryExecution qEx = QueryExecutionFactory.create(q,
307                 model);
308             ResultSet rs = qEx.execSelect();
309
310             int bQueryCount = 0;
311             while (rs.hasNext()) {
312                 QuerySolution sol = rs.next();
313                 // patternnode
314                 String patternUri = PatternVocab.NS_DATA + "/"
315                     +
316                     URLEncoder.encode(sol.getLiteral("patternLabel").toString());
317                 INode patternNode = patternGraph.getNode(patternUri);
318                 INode queryNode =
319                     patternGraph.createNode(PatternVocab.CLASS_Query);
320                 queryNode.setLabel(queryLabel);
321                 patternGraph.createStatement(
322                     patternNode, PatternVocab.PRED_Pattern_matches,
323                     queryNode, bundleNode
324                 );

```

```

311 // bquery
312 String bqueryUri = PatternVocab.NS_DATA + "/"
313 + URLEncoder.encode(sol.get("bquery").toString())
314 );
315 INode bqueryNode = patternGraph.getNode(bqueryUri);
316
317 if (bqueryNode == null) {
318     bqueryNode = patternGraph.createNode(
319         bqueryUri, PatternVocab.CLASS_BusinessQuery
320     );
321     bqueryNode.setLabel("bquery_" + bQueryCount++);
322     patternGraph.createStatement(
323         patternNode,
324         PatternVocab.PRED_Pattern_hasBusinessQuery,
325         bqueryNode, bundleNode
326     );
327 }
328 INode entityNode =
329     slGraph.getNode(sol.get("entityUri").toString());
330 patternGraph.createStatement(
331     bqueryNode, PatternVocab.PRED_BQuery_referencesEntity,
332     entityNode, bundleNode
333 );
334 }
335 };
336 populator.start();
337 }
338 }

```

Listing E.1 – Search plugin for the pattern approach

Bibliography

- [1] Rakesh Agrawal and Edward L. Wimmers. A framework for expressing and combining preferences. *SIGMOD Rec.*, 29(2):297–306, May 2000.
- [2] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Masque/sql - an efficient and portable natural language query interface for relational databases. In *Proceedings of the 6th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, pages 327–330. Gordon and Breach Publishers Inc, 1993.
- [3] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases - an introduction. *CoRR*, cmp-lg/9503016, 1995.
- [4] Douglas E. Appelt and Boyan Onyshkevych. The common pattern specification language. In *Proc. TIPSTER 1998*, pages 23–30.
- [5] J.M. Christian Bastien and Dominique L. Scapin. Ergonomic criteria for the evaluation of human-computer interfaces. Technical Report RT-0156, INRIA, June 1993.
- [6] Madeleine Bates and Robert J. Bobrow. Information retrieval using a transportable natural language interface. In *Proceedings of the 6th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '83, pages 81–86, New York, NY, USA, 1983. ACM.
- [7] Nesrine Ben Mustapha, Hajer Baazaoui Zghal, Marie-Aude Aufaure, and Henda ben Ghezala. Semantic search using modular ontology learning and case-based reasoning. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 3:1–3:12, New York, NY, USA, 2010. ACM.
- [8] Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. Soda: Generating sql for business users. In *Proceedings of the 38th International Conference on Very Large Databases*, 2012.
- [9] Falk Brauer, Michael Huber, Gregor Hackenbroich, Ulf Leser, Felix Naumann, and Wojciech M. Barczynski. Graph-based concept identification and disambiguation for enterprise search. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 171–180, 2010.
- [10] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26(1):65–74, March 1997.

- [11] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, December 2003.
- [12] Philipp Cimiano, Peter Haase, and Jörg Heizmann. Porting natural language interfaces between domains: an experimental user study with the orakel system. In *Proceedings of the 12th international conference on Intelligent user interfaces*, IUI '07, pages 180–189, New York, NY, USA, 2007. ACM.
- [13] Hamish Cunningham, Hamish Cunningham, Diana Maynard, Diana Maynard, Valentin Tablan, and Valentin Tablan. Jape: a java annotation patterns engine, 1999.
- [14] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, January 2001.
- [15] Antonio Ferrández and Jesús Peral. The benefits of the interaction between data warehouses and question answering. In Florian Daniel, Lois M. L. Delcambre, Farshad Fotouhi, Irene Garrigós, Giovanna Guerrini, Jose-Norberto Mazón, Marco Mesiti, Sascha Müller-Feuerstein, Juan Trujillo, Traian Marius Truta, Bernhard Volz, Emmanuel Waller, Li Xiong, and Esteban Zimányi, editors, *EDBT/ICDT Workshops*, ACM International Conference Proceeding Series. ACM, 2010.
- [16] David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.
- [17] David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.
- [18] Michal Finkelstein-landau and Emmanuel Morin. Extracting semantic relationships between terms: Supervised vs. unsupervised methods. In *Proceedings of International Workshop on Ontological Engineering on the Global Information Infrastructure*, pages 71–80, 1999.
- [19] Vanessa Lopez Garcia, Enrico Motta, and Victoria Uren. Aqualog: an ontology-driven question answering system to interface the semantic web. In *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume: demonstrations*, NAACL-Demonstrations '06, pages 269–272, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.
- [20] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. A framework for recommending olap queries. In *Proceedings of the ACM 11th international workshop on Data warehousing and OLAP*, DOLAP '08, pages 73–80, New York, NY, USA, 2008. ACM.

- [21] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. Recommending multidimensional queries. In *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '09, pages 453–466, Berlin, Heidelberg, 2009. Springer-Verlag.
- [22] Alessandra Giordani and Alessandro Moschitti. Syntactic structural kernels for natural language interfaces to databases. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, ECML PKDD '09, pages 391–406, Berlin, Heidelberg, 2009. Springer-Verlag.
- [23] Matteo Golfarelli and Stefano Rizzi. A methodological framework for data warehouse design. In *Proceedings of the 1st ACM international workshop on Data warehousing and OLAP*, DOLAP '98, pages 3–9, New York, NY, USA, 1998. ACM.
- [24] Matteo Golfarelli, Stefano Rizzi, and Paolo Biondi. Myolap: An approach to express and evaluate olap preferences. *IEEE Trans. on Knowl. and Data Eng.*, 23(7):1050–1064, July 2011.
- [25] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '61 (Western), pages 219–224, New York, NY, USA, 1961. ACM.
- [26] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '61 (Western), pages 219–224, New York, NY, USA, 1961. ACM.
- [27] Ralph Grishman. Response generation in question answering systems. In *Proceedings of the 17th annual meeting on Association for Computational Linguistics*, ACL '79, pages 99–101, Stroudsburg, PA, USA, 1979. Association for Computational Linguistics.
- [28] Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin, and Fernando C. N. Pereira. Team: an experiment in the design of transportable natural-language interfaces. *Artif. Intell.*, 32:173–243, May 1987.
- [29] Yong-Jin Han, Tae-Gil Noh, Seong-Bae Park, Se Young Park, and Sang-Jo Lee. A natural language interface of thorough coverage by concordance with knowledge bases. In *Proceedings of the 15th international conference on Intelligent user interfaces*, IUI '10, pages 325–328, New York, NY, USA, 2010. ACM.
- [30] Felix Hausdorff. *Grundzüge der mengenlehre*. Von Veit, Leipzig, 1914.
- [31] David Hawking. Challenges in enterprise search. In Klaus-Dieter Schewe and Hugh E. Williams, editors, *ADC*, volume 27 of *CRPIT*, pages 15–24. Australian Computer Society, 2004.

- [32] Bruce Hayes, Susan Curtiss, Anna Szabolcsi, Tim Stowell, Edward Stabler, Dominique Sportiche, Hilda Koopman, Patricia Keating, Pamela Munro, Nina Hyams, and Donca Steriade. *Linguistics: An Introduction to Linguistic Theory*. Wiley-Blackwell, February 2001.
- [33] Marti A. Hearst. ‘natural’ search user interfaces. *Commun. ACM*, 54(11):60–67, November 2011.
- [34] L. Hirschman and R. Gaizauskas. Natural language question answering: the view from here. *Nat. Lang. Eng.*, 7:275–300, December 2001.
- [35] David A. Hull. Xerox trec-8 question answering track report. In *Proceedings of the 8th Text REtrieval Conference (TREC-8)*, 1999.
- [36] Mohsen Jamali and Martin Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’09, pages 397–406, New York, NY, USA, 2009. ACM.
- [37] Werner Kieß ling. Foundations of preferences in database systems. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB ’02, pages 311–322. VLDB Endowment, 2002.
- [38] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. Explaining structured queries in natural language. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *ICDE*, pages 333–344. IEEE, 2010.
- [39] Nicolas Kuchmann-Beauger and Marie-Aude Aufaure. A natural language interface for data warehouse question answering. In *Proceedings of the 16th international conference on Natural language processing and information systems*, NLDB’11, pages 201–208, Berlin, Heidelberg, 2011. Springer-Verlag.
- [40] Nicolas Kuchmann-Beauger and Marie-Aude Aufaure. Structured data-based q&a system using surface patterns. In *Proceedings of the 9th international conference on Flexible Query Answering Systems*, FQAS’11, pages 37–48, Berlin, Heidelberg, 2011. Springer-Verlag.
- [41] Yunyao Li, Ishan Chaudhuri, Huahai Yang, Satinder Singh, and H. V. Jagadish. Danalix: a domain-adaptive natural language interface for querying xml. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD ’07, pages 1165–1168, New York, NY, USA, 2007. ACM.
- [42] Yunyao Li, Huahai Yang, and H. V. Jagadish. Nalix: an interactive natural language interface for querying xml. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD ’05, pages 900–902, New York, NY, USA, 2005. ACM.

- [43] Yunyao Li, Huahai Yang, and H. V. Jagadish. Constructing a generic natural language interface for an xml database. In Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Böhm, Alfons Kemper, Torsten Grust, and Christian Böhm, editors, *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, pages 737–754. Springer, 2006.
- [44] Vanessa Lopez, Enrico Motta, and Victoria S. Uren. Poweraqua: Fishing the semantic web. In York Sure and John Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2006.
- [45] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
- [46] Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 55–61, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.
- [47] Michael Minock. C-phrase: A system for building robust natural language interfaces to databases. *Data Knowl. Eng.*, 69:290–302, March 2010.
- [48] Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1):2:1–2:27, December 2008.
- [49] Rani Nelken and Nissim Francez. Querying temporal databases using controlled natural language. In *Proceedings of the 18th conference on Computational linguistics - Volume 2*, COLING '00, pages 1076–1080, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.
- [50] Donald A. Norman. How might people interact with agents. *Commun. ACM*, 37(7):68–71, July 1994.
- [51] Giorgio Orsi, Letizia Tanca, and Eugenio Zimeo. Keyword-based, context-aware selection of natural language query patterns. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 189–200, New York, NY, USA, 2011. ACM.
- [52] Fernando Pereira. Extraposition grammars. *Comput. Linguist.*, 7(4):243–256, October 1981.
- [53] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern natural language interfaces to databases: composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.
- [54] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 149–157, New York, NY, USA, 2003. ACM.

- [55] Oscar Romero, Diego Calvanese, Alberto Abelló, and Mariano Rodríguez-Muro. Discovering functional dependencies for multidimensional design. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, DOLAP '09, pages 1–8, New York, NY, USA, 2009. ACM.
- [56] Carsten Sapia. Promise: Predicting query behavior to enable predictive caching strategies for olap systems. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, DaWaK 2000, pages 224–233, London, UK, UK, 2000. Springer-Verlag.
- [57] E. Saquete, P. Martínez-Barco, R. Muñoz, and J. L. Vicedo. Splitting complex temporal questions for question answering systems. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.
- [58] Ben Shneiderman and Pattie Maes. Direct manipulation vs. interface agents. *interactions*, 4(6):42–61, November 1997.
- [59] Eriks Sneiders. *Automated Question Answering: Template-based Approach*. PhD thesis, Royal Institute of Technology, Sweden, 2002.
- [60] Eriks Sneiders. Automated email answering by text pattern matching. In Hrafn Loftsson, Eiríkur Rögnvaldsson, and Sigrún Helgadóttir, editors, *IceTAL*, volume 6233 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2010.
- [61] Radu Soricut and Eric Brill. Automatic question answering using the web: Beyond the factoid. *Inf. Retr.*, 9(2):191–206, March 2006.
- [62] Linda Stadtmüller. The coming storage armageddon: What your database administrator isn't telling you. Technical report, IBM, 2011.
- [63] Raphaël Thollot. *Dynamic Situation Monitoring and Context-Aware BI Recommendations*. PhD thesis, École Centrale Paris, 2012.
- [64] Raphaël Thollot, Falk Brauer, Wojciech M. Barczynski, and Marie-Aude Aufaure. Text-to-query: dynamically building structured analytics to illustrate textual content. In *EDBT '10: Proceedings of the 2010 EDBT/ICDT Workshops*, pages 1–8, New York, NY, USA, 2010. ACM.
- [65] Raphaël Thollot, Nicolas Kuchmann-Beauger, and Marie-Aude Aufaure. Semantics and usage statistics for multi-dimensional query expansion. In Sang-goo Lee, Zhiyong Peng, Xiaofang Zhou, Yang-Sae Moon, Rainer Unland, and Jaesoo Yoo, editors, *Database Systems for Advanced Applications*, volume 7239 of *Lecture Notes in Computer Science*, pages 250–260. Springer Berlin / Heidelberg, 2012.
- [66] Cynthia A. Thompson and Raymond J. Mooney. Acquiring word-meaning mappings for natural language interfaces. *J. Artif. Int. Res.*, 18(1):1–44, January 2003.

- [67] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 639–648, New York, NY, USA, 2012. ACM.
- [68] Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. Panto: A portable natural language interface to ontologies. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, ESWC '07, pages 473–487, Berlin, Heidelberg, 2007. Springer-Verlag.
- [69] David H. D. Warren and Fernando C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *Comput. Linguist.*, 8:110–122, July 1982.
- [70] Dimitri Watel. Apprentissage de patrons de questions. Master's thesis, Université Paris-Sud 11, 91400 Orsay, 2011.
- [71] W. A. Woods. Transition network grammars for natural language analysis. *Commun. ACM*, 13(10):591–606, October 1970.
- [72] W. A. Woods. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*, AFIPS '73, pages 441–450, New York, NY, USA, 1973. ACM.
- [73] W. A. Woods. Progress in natural language understanding: an application to lunar geology. In *AFIPS '73: Proceedings of the June 4-8, 1973, national computer conference and exposition*, pages 441–450, New York, NY, USA, 1973. ACM.
- [74] W A Woods. Readings in natural language processing. chapter Semantics and quantification in natural language question answering, pages 205–248. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986.
- [75] Alexander Yates, Oren Etzioni, and Daniel Weld. A reliable natural language interface to household appliances. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 189–196, New York, NY, USA, 2003. ACM.
- [76] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, pages 658–666. AUAI Press, 2005.